UNIX Communication

This chapter concentrates on the UNIX communication utilities. It describes the commands available for communicating with other users on the system, reading the news about the system, and broadcasting messages to all users. It explains the UNIX electronic mail (e-mail) facilities and shows the commands and options available. This chapter also describes how the shell and other variables affect your e-mail environment, and it shows you how to make a startup file that customizes use of the e-mail utilities.

In This Chapter

10.1 WAYS TO COMMUNICATE

UNIX provides an array of commands and capabilities for communicating with other users. You can have a simple interactive communication with another user by sending and receiving mail through the mail delivery system or you can broadcast messages to everyone on the system.

Be sure to follow some basic guidelines for communication with other users in the system:

- Be polite; do not use profanity.
- Think before sending. Do not send mail that you may regret later.
- Save a copy of all your outgoing mail.

10.1.1 Using Two-Way Communication: The write Command

You can use the **write** command to communicate with another user. This communication is interactive, from your terminal to another terminal, so the receiving terminal must be a logged-on user. The message you send appears on the receiving user's screen. Then that user can send you a reply by issuing the **write** command from his or her terminal. Using the **write** command, two users can effectively have a conversation through their terminals.

Let's follow an example step by step to see how **write** works. Suppose your user ID is `david`, and you want to chat with Daniel, whose user ID is `daniel`.

Type **write daniel** and press **[Return]**.

If Daniel is not logged on, you see the message

```
daniel not logged on.
```

If Daniel is logged on, he sees a message similar to this on his screen:

```
Message from david on (tty06) [Thu Nov 9:30:30]
```

On your terminal, the cursor is placed on the next line, and the system waits for you to type your message. Your message may contain many lines, and each line you type is transmitted to Daniel when you press **[Return]**. You signal the end of your message by pressing the **[Ctrl-d]** key at the beginning of an empty line. This terminates your **write** and sends an EOT (end of transmission) message to Daniel.

Figure 10.1 shows two screens, depicting a typical conversation. The top screen is David's terminal and the bottom one is Daniel's.

To use the **write** *command, you must know the user ID of the person with whom you want to communicate. You use the* **who** *command (discussed in Chapter 3) to obtain the user IDs of the logged-on users.*

Daniel can reply by using the **write** command from his terminal, but he does not have to wait for you to finish your message. When he sees the initial message that you are writing to him, he can send you messages while you send him messages if he types **write david** and presses **[Return]**.

See page x for an explanation of the icons used to highlight information in this chapter.

Figure 10.1
Typical Screen Conversation: The Top Screen is David's and the Bottom Screen is Daniel's

```
$write daniel [Return]
Hello Dan [Return]
Is today's meeting still on? [Return]
[Ctrl-d]
<EOT>
$
```

```
Message from david on (tty06) [Thu Nov 9:30:30] ...
Hello Dan
Is today's meeting still on?
<EOT>
$
```

With **write** simultaneously active on both terminals, you and Daniel can carry on a two-way conversation. Sometimes this two-way exchange becomes confusing, so it is useful to establish a protocol for using **write.** The common protocol for UNIX users is to end message lines with the character *o* (for over) to inform the receiving party that a message is finished and you are (possibly) waiting for a reply. When you intend to end the conversation, type **oo** (for over and out).

If you are receiving a **write** message from another user, the message appears on your terminal, regardless of what you are doing. If you are using the vi editor and are in the middle of an editing job, the write message appears on the screen where the cursor is. But don't be alarmed. This is a terminal-to-terminal communication, and what **write** produces does not damage your editing file. It simply overwrites information on your screen, and you can continue with your editing or whatever other job you were doing.

Nevertheless, receiving messages while you are concentrating on a job is inconvenient, not to mention the mess it makes on your screen. You can prevent your terminal from accepting messages coming from the **write** command.

10.1.2 Inhibiting Messages: The mesg Command

You can use the **mesg** command as a toggle to stop receiving messages from the **write** command or to reactivate receiving messages. **mesg** without an argument shows the current status of your terminal in this respect.

The following command sequence shows you how to protect yourself from annoying messages:

```
$ mesg [Return]  . . . . . Check status of your terminal.
is y  . . . . . . . . . . . It is set to YES, accepting messages.
$ mesg n [Return]  . . . . Set it to NO, and stop receiving messages.
$ mesg [Return]  . . . . . Check again.
is n  . . . . . . . . . . . Now it is set to NO.
$_  . . . . . . . . . . . . Return to the prompt.
```

To practice the communication commands, you usually want to have another user to participate in the exercises. However, you can practice most of these commands by yourself. You can send and receive messages using your own user ID.

10.1.3 Displaying News Items: The news Command

You can use the **news** command to find out what is happening in the system. **news** gets its information from the system directory where the news files are placed, usually in `/usr/news`. Without any options, **news** displays all the files you have not seen from the `news` directory. It refers to and updates a file called `.news_time` in your HOME directory. This file is created in your HOME directory the first time you use the **news** command and remains an empty file. The **news** command uses its access time to determine the last time you gave the **news** command.

1. *You press the interrupt key (usually* **[Del]***) to stop displaying one news item and continue with the next item.*
2. *You press the interrupt key twice to quit (terminate) the* **news** *command.*

To check the latest news, type **news** and press **[Return]**. Figure 10.2 shows some sample news items.

Figure 10.2
The **news** Command

```
$ news [Return]
david  (root) Mon Nov      28    14:14:14  2005
   Let's congratulate david; he got his B.S. degree.
   Friday night party on second floor. Be there!

books (root) Mon Nov       28    14:14:14  2005
   Our technical library is growing.
   New set of UNIX books is now available.
$_
```

Each news item has a header that shows the filename, file owner, and the time that the file was placed in the news directory.

news Options

The **news** options are summarized in Table 10.1. The options do not update your `.news_time` file.

The following command sequences show how the **news** command options work:

- List the current news items, using the **-n** option: type **news -n** and press **[Return]**. UNIX shows only the headers of the files that contain the news items. The header shows the filename, the owner of the file, and the time it was created. Figure 10.3 shows the output of the command.

Table 10.1
The **news** Command Options

Option	Operation
-a	Displays all the news items, old or new files.
-n	Lists only the names of the files (headers).
-s	Displays the number of the current news items.

Figure 10.3
The **news** Command and the **-n** Option

```
$ news -n [Return]
   david  (root) Mon Nov  28  14:14:14  2005
   books  (root) Mon Nov  28  14:14:14  2005
$_
```

❑ To display a specific news item, type **news david** and press **[Return]**, for example. Figure 10.4 shows the output of the sample news item.

Figure 10.4
The **news** Command with Specified News Item

```
$ news david [Return]
david  (root) Mon Nov     28  14:14:14  2005
   Let's congratulate david; he got his B.S. degree.
   Friday night party on second floor. Be there!
$_
```

The name of a news item is its filename in the news directory.

10.1.4 Broadcasting Messages: The wall Command

You can use the **wall** (write all) command to send messages to all currently logged on users. The **wall** command reads from the keyboard (standard input) until you enter **[Ctrl-d]** at the beginning of a blank line to signal the end of the message. The **wall** executable file is usually placed in the `/etc` directory and is not defined in the *PATH* standard variable—which means you have to type the full pathname to invoke it.

The **wall** command is usually used by the system administrator to warn users of imminent events. You may not have access to it.

Assuming your login name is `david` (and you have access to **wall**), send a message to all users:

❑ Type **/etc/wall** and press **[Return]**.

❑ Type **Alert ...** and press **[Return]**.

❑ Type **Lab will be closed in 5 minutes. Time to log out.** and press **[Return]**.

❑ Press **[Ctrl-d]**. The system responds as shown in Figure 10.5.

Figure 10.5
Invoking the **wall** Command

```
$ /etc/wall [Return]
   Alert ... [Return]
   Lab will be closed in 5 minutes. Time to log out. [Return]
[Ctrl-d]
$_

Broadcast message from david
   Alert ...
   Lab will be closed in 5 minutes. Time to log out.
```

1. *The message is also sent to its sender, so you see your own broadcast message.*
2. *The* **wall** *message you send is not received by currently logged-on users who set* **mesg** *to* **n**.
3. *The system administrator can override access denial.*
4. *Your messages carry your user ID; thus you cannot send an anonymous message.*

10.1.5 Using Two-Way Communication: The talk Command

You can use the **talk** command to communicate with another logged-on user. This command is similar to the **write** command. When you enter **talk,** your display will be divided into two sections. The top section displays what you type and the bottom section displays what the other person types. For example, if you type **talk daniel** and press the **[Return]** key, your screen splits into two sections, and the following message appears on your screen:

```
[waiting for your party to respond]
```

and the following message appears on Daniel's screen:

```
Message from Talk_daemon@xyz at 111:22 ..
talk: Connection requested by david@xyz
talk: Respond with: talk david@xyz
```

and Daniel responds by typing

```
talk david@xyz
```

Now, Daniel's screen is divided into two sections, and the conversation begins.

To end the session, either of the parties involved can press **[Ctrl-c]** and the following message is displayed on both terminals.

```
[Connection closing. Exiting]
```

The following message is displayed on your terminal if Daniel has used the **mesg** command to block messages:

```
[Your party is refusing messages]
```

The following message is displayed on your terminal if Daniel is not logged on:

```
[Your party is not logged on]
```

Once more, if you are David and want to communicate with Daniel, Figure 10.6 shows the screen conversation.

Figure 10.6
Screen Conversation: The Top Screen is David's; the Bottom One is Daniel's

```
[Connection established]                          david's screen
Hi Dan!
Is today's meeting still on?
_____________________________________________________________

Hi Dave
Yes. Be there!
Bye.
```

```
[Connection established]                          daniel's screen
Hi Dave!
Yes. Be there!
Bye.
_____________________________________________________________

Hi Dan!
Is today's meeting still on?
```

10.2 ELECTRONIC MAIL

Electronic mail (*e-mail*) is an essential part of the office environment. E-mail gives you the capability to send and receive messages, memos, and other documents to and from other users. The main difference between sending mail using the e-mail service and using the **write** command is that with the **write** command you see only the messages sent to you if you are logged on. But with e-mail, your mail is automatically kept for you until you issue the command to read it. E-mail service is more convenient and is faster than conventional mail service, and it does not interrupt the receiving party the way phone calls do.

Under UNIX, either the **mail** or **mailx** command can be used to send or read e-mail. The **mailx** utility is based on Berkeley UNIX **mail** and has more powerful features than **mail,** enabling you to manipulate (review, store, dispose of, etc.) your mail easily and efficiently. **mailx** is the e-mail command discussed in this book. It has a large number of features and options; using some of them requires advanced UNIX experience. In this chapter, we describe enough to make you feel comfortable using it and interested in looking for more information about it.

Where do you find more information? Well, how about the **man** *command? (In case you have forgotten,* **man** *was discussed in Chapter 3.)*

You use the **mailx** command to send mail to other users or read mail that is sent to you. The **mailx** operation involves a number of files; the way it appears and functions depends on the environment variables that are set up in files, and it needs files for storing your mail.

10.2.1 Using Mailboxes

In the UNIX mail system, you have two kinds of mailboxes: a system mailbox and a private mailbox.

Your System Mailbox

Every user of the system has a *mailbox,* which is a file with the same name as your login name (user ID). This file is typically stored in `/usr/mail`. Mail sent to you is stored in this file, and when you read a message, **mailx** reads from your mailbox. If your login name is `david`, the full pathname to your mailbox could be `/usr/mail/students/david`.

You can use the **set** command (discussed in Chapter 9) to see your system mailbox pathname. The variable *MAIL* is set to the filename that receives your mail.

Your Private Mailbox: The *mbox* File

After you read your mail, **mailx** automatically appends a copy of it to a file called `mbox` in your `HOME` directory. If an `mbox` file does not exist, then **mailx** creates one in your `HOME` directory the first time you read your mail. This file contains mail that you have read but that `$HOME/mbox` has not deleted or saved elsewhere. The variable *MBOX* controls the filename. The default value is `HOME/mbox`. For example, the following command changes the default setting and sets up your private mailbox in the `email` directory.

```
MBOX=$HOME/EMAIL/mbox
```

The explicit saving of a message or using the x (exit) command to exit mailx disables the automatic saving of your messages.

The Customizable mailx Environment

You can customize your **mailx** environment by setting up appropriate variables in two startup files: the `mail.rc` file in the system directory and the `.mailrc` file in your `HOME` directory.

When you call **mailx,** it first checks for a startup file called `mail.rc`. The full pathname to this file is similar to the following:

```
/usr/share/lib/mailx/mail.rc
```

This file is usually created and maintained by the system administrator. Variables set in this file are applicable to all the system users.

The second file that **mailx** looks for is a file called `.mailrc` in your `HOME` directory. You can change the **mailx** environment that the system administrator has set up in the `mail.rc` file by setting variables in your `.mailrc` file. This file is not necessary, and **mailx** works fine without it, as long as you are happy with the system administrator's arrangement. Ways to customize your **mailx** environment are discussed in more detail in Section 10.5.

10.2.2 Sending Mail

In order to send mail to another person, you must know that person's login name. For example, if you want to send mail to a user identified by the login name `daniel`, you type **mailx daniel** and press **[Return]**.

By default, input to **mailx** (your message) comes from the keyboard (standard input). Depending on how your system environment variables are set, **mailx** may show the **Subject:** prompt. If it does, you type the subject for your message, and **mailx** changes to the *input mode* and waits for you to enter the rest of your message. You signal the end of your message by pressing **[Ctrl-d]** at the beginning of a line. **mailx** shows **<EOT>** (for end of transmission), and your message is transmitted.

While in the input mode, **mailx** provides you with a large number of commands, enabling you to compose your message with ease and efficiency. All input mode commands start with a tilde (**~**), and they are called *tilde escape commands*. (The tilde escape commands are explained later in this chapter.)

To send a message to Daniel (login name `daniel`), do the following:

```
$ mailx daniel [Return]  . . . . Send a message to daniel.
Subject: meeting [Return]  . . . Enter the subject.
Hi, Dan [Return]
Let me know if tomorrow's meeting is still on. [Return]
Dave [Return]
[Ctrl-d]. . . . . . . . . . . . . Signal the end of the message.
EOT  . . . . . . . . . . . . . . . mailx shows end of transmission.
$_ . . . . . . . . . . . . . . . . Return to the prompt.
```

1. *The subject field is optional. Just press* **[Return]** *to skip the* **Subject:** *prompt.*
2. *Signal the end of a message by pressing* **[Ctrl-d]** *at the beginning of a blank line.*
3. *Mail is delivered to the other user's mailbox regardless of whether he or she is logged on.*
4. *Recipients are informed that they have mail as soon as they log in. The following message appears on their terminal:*

```
You have mail
```

10.2.3 Reading Mail

In order to read your mail, you type **mailx** with no argument. If you have mail in your mailbox, then **mailx** shows two lines of information followed by a numbered list of headers of messages in your mailbox and then the **mailx** prompt, which by default is a question mark. At this point, **mailx** is in command mode, and you can issue commands to delete, save, or reply to messages. You press **q** at the **?** prompt to exit **mailx.**

The list of headers consists of one line for each mail item in your mailbox. The format is as follows:

```
> status message # sender date lines/characters subject
```

Each field in the header line conveys certain information about your mail:

- The **>** indicates that the message is the current message.
- The *status* is **N** if the message is new. This means that you have not read this mail.

- The *status* is **U** (unread) if the message is not new. That means you have seen the message header before, but you have not read the mail itself yet.
- The *message #* indicates the sequence number of the mail in your mailbox.
- The *sender* is the login name of the person who sent you the mail.
- The *date* shows the date and time that mail arrived in your mailbox.
- The *line/characters* shows the size of your mail by the number of lines and number of characters.

Suppose you are Daniel and you have just logged in and you want to read your mail.

❑ The system informs you that you have mail:

```
You have mail
```

❑ To read your mail, type **mailx** and press **[Return]**; UNIX responds:

```
mailx version 4.0 Type ? for help.
"/usr/students/mail/daniel": 1 message 1 new
> N  1  david .... Thu Nov 28 14:14 8:126 meeting
```

1. *The first header line shows your* **mailx** *version number and informs you that you can press* **?** *to get help.*
2. *The second header line shows* `/usr/mail/daniel`, *your system mailbox, followed by the number and status of your messages. In this case, you have one message, and* **N** *indicates that this is the first time you are reading it.*

The **?** prompt shows that **mailx** is in command mode. You specify the mail you want to read by typing its associated message number. You can also press **[Return]** to start reading from the current mail (indicated by the **>** sign on the header) and continue reading your mail in sequence by pressing **[Return]** after the **?** prompt.

```
? . . . . . mailx is in command mode.
? 1 [Return] . . Display message 1, the only message in your mailbox.
Message 1:
From: david  Mon, 28 Nov 01 14:14 EDT  2005
To: daniel
Subject: meeting
Status: R
Hi, Dan
Let me know if tomorrow's meeting is still on.
Dave
? . . . . . Ready for the next command.
? q [Return] . . Exit mailx.
Saved 1 message in /usr/students/david/mbox
$_ . . . . . Return to the shell.
```

When you use **q** *to quit from* **mailx,** *it saves a copy of the mail that you have read in* `mbox`, *your private mailbox in your* `HOME` *directory. (Thus, at this point, your system mailbox is empty.)*

Suppose you want to read your mail again. Type **mailx** and press **[Return]**; UNIX responds:

```
No mail for daniel
```

Check what you have in mbox.

```
$ cat mbox [Return]. . . . . . Check what you have in your mbox.
From: david  Mon, 28 Nov 01 14:14 EDT  2005
To: daniel
Subject: meeting
Status: R
Hi, Dan
Let me know if tomorrow's meeting is still on.
Dave
```

As you expected, mbox contains a copy of your mail.

10.2.4 Exiting mailx: The q and x Commands

You can exit **mailx** by typing the **q** (*quit*) or **x** (*exit*) command at the **?** prompt. Although both commands cause you to exit from the **mailx,** they do so in a different manner.

The **q** command causes the automatic removal of the mail that you have read from your system mailbox. By default, a copy of the removed mail is kept in your private mailbox (any filename assigned to the MAIL variable).

The **x** command does not remove the mail you have read from your system mailbox. In fact, when you use **x**, nothing changes in your mailbox. Even deleted messages remain intact.

mailx Options Table 10.2 summarizes the **mailx** options. These options are used in the command line when you invoke **mailx** for reading or sending mail. The following command sequences show the use of the **mailx** options:

Table 10.2
The **mailx** Command Options

Option	Operation
-f ***filename***	Reads mail from the specified *filename* instead of the system mailbox. If no file is specified, it reads from mbox.
-H	Displays a list of the message headers.
-s ***subject***	Sets the subject field to the string *subject.*

```
$ mailx -H [Return]  . . . . . Display the message headers only.
N 1 daniel         Thu ESP 30  12:26 6/103  Room
N 2 susan          Thu Sep 30  12:30 6/107  Project
N 3 marie          Thu Sep 30  13:30 6/70   Welcome
$_. . . . . . . . . . . . . . . . . Back to the shell.
```

Type **mailx -f mymail** and press **[Return]** to read mail from the specified file mymail instead of your system mailbox; UNIX responds:

```
/usr/students/daniel/mymail: No such file or directory
```

The **mailx** by default reads mail from your system mailbox. With the **-f** option, it reads your mail from a specified file, such as your old mail files. In this case, you specified `mymail`, and the message shows `mymail` is not in your current directory.

If you type **mailx -f** and press **[Return]**, with no filename specified, UNIX defaults to your private mailbox, so you see a display like the following:

```
mailx version 4.0 Type ? for help.
  "/usr/students/mail/daniel": 1 message 1 new
> N  1  david Thu Nov 28 14:14 17/32 meeting
```

To use the **-s** option, set the *subject* string as part of the command line, and send mail to Daniel (whose user ID is `daniel`), do the following:

```
$ mailx -s meeting daniel [Return] . . . .  Send mail to daniel;
                                             set the subject field to
                                             the string meeting.
_ . . . . . . . . . . . . . . . . . . . . .  Compose your
                                             message to Daniel.
[Ctrl-d] . . . . . . . . . . . . . . . . . .  End your message.
EOT . . . . . . . . . . . . . . . . . . . . .  mailx shows end of
                                             transmission.
$_. . . . . . . . . . . . . . . . . . . . . .  Return to the shell.
```

When Daniel reads your message, the subject field shows **Subject: meeting.**

Use quotation marks around the subject string if it contains spaces.

10.3 mailx INPUT MODE

While **mailx** is in the *input mode* (composing the mail to be sent), a variety of commands are at your disposal. These commands all start with the tilde (~); that lets you temporarily escape from the input mode and issue certain commands, which is why they are referred to as *tilde escape commands.* Table 10.3 summarizes some of these commands.

Some of these commands are quite important. Imagine you want to write a message that contains more than just a few lines. Using the primitive **mailx** editor is cumbersome and just not up to the job. Instead of using it, you can invoke the vi editor, compose your message using all the ease and power of vi, and then, when you have finished composing your message, exit vi and return to the **mailx** input mode. Then you can give other commands or send the message.

1. **Tilde escape commands are applicable only when mailx is in the input mode.**
2. **All tilde commands must be entered at the beginning of a line.**

The following command sequences show the use of the tilde escape commands while **mailx** is in the input mode. Assume your login name is `david` and you are sending mail to yourself.

```
$ mailx -s "Just a Test" david [Return].....Send mail to
                                             yourself.
```

mailx is invoked with the **-s** option. The quotation marks are necessary because of the embedded spaces in the specified *subject* string.

Table 10.3
The **mailx** Tilde Escape Commands

Command	Operation
~?	Displays a list of all tilde escape commands.
~! *command*	Lets you invoke the specified shell *command* while composing your message.
~e	Invokes an editor. The editor to be used is defined in the mail variable called *EDITOR*. vi is the default.
~p	Displays the message currently being composed.
~q	Quits input mode. Saves your partially composed message in the file called `dead.letter`.
~r *filename*	Reads the file named *filename* and adds its contents to your message.
~< *filename*	Reads the file named *filename* (using the redirection operator) and adds its contents to your message.
~<! *command*	Executes the specified command and places its output into the message.
~v	Invokes the default editor, the vi editor, or uses the value of the mail variable *VISUAL,* which can be set up for other editors.
~w *filename*	Writes the message currently being composed to the specified *filename*.

```
$_ . . . . . .  mailx is in the input mode.
~! date [Return] . . . . . .  Invoke the date command.
Mon, Nov 28  16:16 EDT 2005
_ . . . . . .  mailx is ready for input.
```

At this point you are using **~!** and executing the **date** command. You can execute any command you wish. The output of the command does not become part of the message you are composing.

```
$~< ! date [Return] . . . . . .  Invoke the date command and redirect the output of the
                                 date command to be included in your message.
"date" 1/29. . . . . .  Feedback message is given.
_ . . . . . .  Ready.
```

The feedback message indicates that one line consisting of 29 characters (the output of the **date** command) is added to your text.

```
This is a test message to explore mailx. [Return]
$~v [Return] . . . . . .  Now use the vi editor to compose the rest
                          of your message.
```

At this point, you have invoked the vi editor and your partially composed message is the input file to the vi editor.

```
Mon, Nov 28  16:16 EDT 2005
This is a test message to explore mailx capabilities.
~
~
~
"/tmp/Re26485"  2 lines, 69 characters
```

Now all the power and flexibility of the vi editor are at your disposal. You can delete, modify, or save your text. You can execute commands or import another file and continue composing your message.

```
Mon, Nov 28  16:16 EDT 2005
This is a test message to explore mailx capabilities.
This message is composed using the vi editor.
:wq [Return]  . . . . . Exit vi.
3 lines, 115 characters  . . . Get vi feedback.
(continue)  . . . . . Feedback message.
_  . . . . . You are back to mailx input mode.
~w first.mail [Return] . . . . Save your mail in a file called
                                first.mail.
"first.mail" 3/115 . . . . . Get feedback message.
```

The feedback message indicates the size of `first.mail`: It contains 3 lines and 115 characters.

The **~w** is used to save the currently composed message in the specified file `first.mail`. This is a good habit to practice so that you have a copy of your transmitted messages. If you set the **mailx** record variable, then your outgoing mail is automatically saved.

```
~q [Return] . . . . . Quit mailx input mode.
$_ . . . . . Return to the shell.
```

Using **~q** to quit **mailx** input mode also saves your partially composed message in a file called `dead.letter` in your HOME directory (or any filename assigned to the DEAD variable).

Let's start again and complete the sending of your mail. At this point the scenario is this: you are `david` and you want to send mail to yourself. You have a copy of your message in a file called `first.mail` (you used **~w** to save it) and another copy in `dead.letter` (you used **~q** to exit input mode).

Send `first.mail` to `david` using the input redirection sign (**<**) on the command line.

```
$ mailx david < first.mail [Return] . . . . Send mail.
$_ . . . . . . . . . . . . . . . . . . . . . Done.
```

The **<** sign directs the shell to pass the specified filename (`first.mail`) as input to the **mailx** command.

You can specify more than one input file.

By default, the Bourne shell checks every 10 minutes for your new mail, so you have to wait a while before you can read the mail you just sent to yourself. When there is new mail in your mailbox, UNIX displays the message *you have mail* before the next prompt.

To send `first.mail` to `david` using the tilde escape command, do the following:

```
$ mailx david [Return] . . . . Send mail to yourself.
Subject:. . . . . Ready for you to type your message.
~< first.mail [Return] . . . . Read in the contents of the file
                                first.mail.
"first.mail" 3/115 . . . . . Get feedback message.
```

```
[Ctrl-d] . . . . . . . . . . . . . Transmit it.
EOT  . . . . . . . . . . . . . . . mailx  shows EOT.
$_ . . . . . . . . . . . . . . . . Return to the shell.
```

The **~<** *reads in the specified file, in this case* `first.mail`*, and adds its contents to the message you are currently composing. You can also use the* **~r** *command to achieve the same results.*

Get your partially composed message that **mailx** saved in the `dead.letter` file, complete the message, and send it to `david`:

```
$ mailx david [Return] . . . . . . . . . . . . Send message to david.
Subject:. . . . . . . . . . . . . . . . . . . . Ready to go.
~r dead.letter [Return]  . . . . . . . . . . . Read in (import) dead.letter file.
~v [Return] . . . . . . . . . . . . . . . . . . Invoke the vi editor.
Mon, 28 Nov 01 16:16 EDT 2005
   This is a test message to explore mailx capabilities.
   This message is composed using the vi editor.
   ~
   ~
   "/tmp/Re265" 3 lines and 115 characters . . . . . . The vi editor feedback message.
```

Let's assume you want to add a few lines to your message:

```
Mon, 28 Nov 01 16:16 EDT 2005 . . . . . . . . Complete the message.
This is a test message to explore mailx capabilities.
This message is composed using the vi editor.
This is the first time I am using e-mail. Maybe I should save this text,
get a copy of it, and frame it!
~
~
:wq [Return]  . . . . . . . . . . . . . . . . . Save and quit.
(continue). . . . . . . . . . . . . . . . . . . Get feedback message.
```

At this point your message is not displayed, but you can see what you have composed before sending it by typing **~p** and pressing **[Return]**; UNIX displays the whole message one page at a time:

```
~p [Return]  . . . . . . . . . Display the composed message.
Mon, 28 Nov 01 16:16 EDT 2005
This is a test message to explore mailx capabilities.
This message is composed using the vi editor.
This is the first time I am using e-mail. Maybe I should
save this text, get a copy of it, and frame it!
[Ctrl-d] . . . . . . . . . . . Indicate the end of your message.
EOT. . . . . . . . . . . . . . This is the end of transmission.
$_ . . . . . . . . . . . . . . Return to the shell.
```

10.3.1 Mailing Existing Files

You do not have to compose your messages using the **mailx** editor at all. Maybe you have a memo already written that you want to send to another user. In that case, you use

the shell input redirection operator to redirect the **mailx** input from the default input device (the keyboard) to an existing file.

Send a file called `memo` to Daniel, whose user ID is `daniel`.

```
$ mailx daniel < memo [Return]  . .  Mail memo to daniel.
$_  . . . . . . . . . . . . . . . .  Return to the shell.
```

Assuming that your message is in a file called `memo` and you want to send it to the user ID `daniel`, this command will do the job.

10.3.2 Sending Mail to a Group of Users

What if you want to send your `memo` file to Daniel and a few others? It would be very inconvenient to type the **mailx** command to send the same message to 10 different users. In that case, you specify a list of the user IDs of the people you intend to receive the mail, and **mailx** sends the mail to all of them.

To send `memo` to user IDs `daniel`, `susan`, and `emma`, type:

```
mailx daniel susan emma < memo [Return]
```

The user IDs are separated by spaces.

If you send mail often to a specific group of people, you can save yourself a lot of typing by defining a name for your list and using the defined name instead of typing the whole list of user IDs every time. To do this, you use the **alias** command, and the format looks like this:

```
alias [name] [userID01] [userID02] [userID03] [userID04]
[userID05] [userID06]
```

where *name* is the name you want to type when you want to send mail to all the people on this list.

For example, if you type:

```
alias friends daniel david gabe emma [Return]
```

UNIX assigns the name `friends` to this list of user IDs. Now, instead of typing all those user IDs, you can just type **friends.** If you place **alias** commands like this in your `.mailrc` file, they become part of your **mailx** environment and you do not need to assign them every time you want to use **mailx.**

Assuming your friends' user IDs are assigned to the name `friends`, mail your `memo` to your friends by typing **mailx friends < memo** and pressing **[Return]**.

10.4 mailx COMMAND MODE

When you are reading mail, **mailx** is in command mode, and the question mark prompt means it is waiting for your commands. While **mailx** is in command mode, a large number of commands are at your service, enabling you to copy, save, or delete your

Table 10.4
The **mailx** Commands Available in Command Mode

Command	Operation
!	Lets you execute the shell commands (the shell escape).
cd ***directory***	Changes to the specified directory, or to the HOME directory if none is specified.
d	Deletes the specified messages.
f	Displays the headlines of the messages.
q	Exits **mailx** and removes the messages from the system mailbox.
h	Displays active message headers.
m ***users***	Sends mail to specified *users*.
R ***messages***	Replies to the sender of the *messages*.
r ***messages***	Replies to the sender of the *messages* and all the other recipients of the same *messages*.
s ***filename***	Saves (appends) the indicated messages to the *filename*.
t ***messages***	Displays (types) the specified *messages*.
u ***messages***	Undeletes the specified *messages*.
x	Exits **mailx**; does not remove messages from system mailbox.

mail. You can reply to the sender of a message or send mail to a specific user without leaving the command mode. Table 10.4 summarizes some of these commands.

Scenario The command sequences in the following sections show how **mailx** reads your mail and what commands and options are applicable in the command mode. They assume that your user ID is `david`, you have three messages in your system mailbox, and you want to read, display, and delete your mail. The **mailx** command provides you with the capabilities to manipulate your mail in different ways. Your only problem would be the selection of appropriate commands for the job at hand.

10.4.1 Ways to Read/Display Your Mail

The **mailx** command lets you read and display your mail in several different ways: each piece of mail one at a time, as a specified range of messages, or as a specified single message. Generally, regardless of how you plan to read your mail, you will want to display a list of the mail, which you do simply by invoking **mailx**. Pressing **[Return]** at the **?** prompt shows the current message.

Read/display your mail:

```
$ mailx [Return]  . . . . .  Invoke mailx and display a list of the mail.
mailx version 4.0  Type ? for help.
"/usr/student/mail/david": 3 messages 3 new
```

```
> N 1  daniel                Fri Sep 30 12:26 6/103 Room
  N 2  susan                 Fri Sep 30 12:30 6/107 Project
  N 3  marie                 Fri Sep 30 13:30 6/79  Welcome
? [Return]  . . . . . . . . . . . Display the current message.
Message 1:
From: daniel Fri Sep 30 12:26 EDT 2005
To: david
Subject: Room
Status: R

Room 707 is reserved for your meetings.

? . . . . . . . . . . . . . . Ready for the next command.
```

Pressing **[Return]** at the **?** prompt shows the current message, in this case message 1.

```
? 3 [Return] . . . . . . . Display message 3.
Message 3:
From: marie Fri Sep 30 12:30 EDT 2005
To: david
Subject: Welcome
Status: R

Welcome back!

? . . . . . . . . . . . . . . Ready for the next command.
? t 1-3 [Return] . . . . . Type messages 1 through 3.
Message 1:
From: daniel Fri Sep 30 12:26 EDT 2005
To: david
Subject: Room
Status: R

Room 707 is reserved for your meetings.

Message 2:
From: susan Fri Sep 30 12:26 EDT 2005
To: david
Subject: Project
Status: R

Your project is in trouble! See me ASAP.

Message 3:
From marie Fri Sep 30 12:30 EDT 2005
To: david
Subject: Welcome
Status: R
Welcome back!
? . . . . . . . . . . . . . . You are ready for the next command.
```

The **t** (type) command displays messages one after another. The range of messages is indicated by the low and high limit numbers separated by a hyphen. In this case, **t 1-3** means to display message numbers 1, 2, and 3.

```
? t 1 3 [Return] . . . . . Display messages 1 and 3.
Message 1:
From: daniel Fri Sep 30 12:26 EDT 2005
To: david
Subject: Room
Status: R

Room 707 is reserved for your meetings.
```

```
Message 3:
From: marie Fri Sep 30 12:30 EDT 2005
To: david
Subject: Welcome
Status: R
Welcome back!
?_ . . . . . . . . . . . . . You are ready for the next command.
```

The **t** *command also shows any specified mail indicated by the mail sequence numbers. If more than one mail sequence number is indicated, the numbers must be separated by a space. In this case,* **t 1 3** *means display messages 1 and 3.*

```
? n [Return] . . . . . . . Show the next message.
At EOF . . . . . . . . . . Get feedback message.
?_ . . . . . . . . . . . . Prompt returns for the next command.
```

The **n** (next) command shows the next message in your mailbox, just like pressing **[Return]**. In this case, there is no next message; thus, the *End of the File* message is displayed.

```
? n10 [Return] . . . . . . Show the tenth message.
10: . . . . . . . . . . . . invalid message number.
```

You can indicate a message number if you want a specific message to be displayed. In this case, **n10** means to display message 10. But there is no message 10, so the error message is displayed.

```
? f [Return] . . . . . . . Show the headline of the current message.
. 3 marie Fri Sep 30 12:30 EDT 2005
?_ . . . . . . . . . . . . The prompt is back.
```

The **f** command shows the headline of your current message, in this case message 3.

```
? x [Return] . . . . . . . Exit mailx.
$_ . . . . . . . . . . . . Return to the shell prompt.
```

If you use the **x** command to exit from **mailx,** all your messages remain intact in your mailbox.

10.4.2 Ways to Delete Your Mail

The **mailx** command lets you delete your mail one message at a time, delete all of the messages at one time, delete a specified range of messages, and recover messages you deleted by mistake. Let's look at some examples. The scenario remains the same. You are David, and there are three messages in your system mailbox.

Delete your mail:

```
$ mailx [Return] . . . . . Read your mail.
mailx version 4.0  Type ? for help.
"/usr/student/mail/david": 3 messages 3 new
> N 1  daniel              Fri Sep 30 12:26 6/105   Room
  N 2  susan               Fri Sep 30 12:30 6/107   Project
  N 3  marie               Fri Sep 30 13:30 6/79    Welcome
? d [Return] . . . . . . . Delete the current message.
```

```
? d3 [Return]. . . . . Delete message 3.
? h . . . . . Show only the headlines.
> N 2 . . . . . susan Fri Sep 30 12:30 6/107 Project
```

The **h** command displays the headlines of the messages in your mailbox. You have deleted messages 1 and 3, and the remaining message (message 2) is displayed.

```
? u1 [Return]. . . . . Undelete message 1.
? u3 [Return]. . . . . Undelete message 3.
```

The **u** *command undeletes the specified message, in this case messages 1 and 3.*

```
? h [Return] . . . . . Check to see whether all messages are in
                       your mailbox.
  1 daniel             Fri Sep 30 12:26 6/103 Room
  2 susan              Fri Sep 30 12:30 6/107 Project
  3 marie              Fri Sep 30 13:30 6/79  Welcome
? d 1-3 [Return] . . . . Delete messages 1 through 3.
? h [Return] . . . . . Check if they are deleted.
No applicable messages
? u* [Return]. . . . . Undelete all the deleted messages.
```

Using the **u** command with the * wild card undeletes all messages in your mailbox.

```
? d/vacation [Return] . . Delete all messages with the word vacation
                          in their subject field.
No applicable messages
```

Using */string* with the delete command, you can delete mail that has the specified string as part of the subject field. In this case, no message in your mailbox matches the string *vacation,* and so no message is removed.

```
? d daniel [Return] . . . Delete all messages from daniel.
```

You can specify the sender's user ID with the delete command to delete all the mail sent by the specified person.

```
? x [Return] . . . . . Exit mailx.
$_. . . . . Back to the shell prompt.
```

There are many ways to delete unwanted mail. However, when you exit from **mailx** using the **x** command, all messages remain in your mailbox, *even the messages you have deleted.* If you use the **q** command to exit **mailx**, then your mailbox is permanently updated according to the commands you have issued.

After you quit mailx using q, you can no longer use the undelete command to restore deleted messages, so make sure that you have deleted only the messages you intended to delete before you quit mailx.

10.4.3 Ways to Save Your Mail

The **mailx** command lets you save your messages in a specified file while you are reading them. You can save all your messages, a single message, or a range of messages. Let's look at some examples. The scenario remains as before: `david` is your login name, and there are three messages in your system mailbox.

```
$ mailx [Return] . . . . . . Read your mail.
mailx version 4.0  Type ? for help.
"/usr/student/mail/david": 3 messages  3 new
> N 1  daniel              Fri Sep 30 12:26 6/103 Room
  N 2  susan               Fri Sep 30 12:30 6/107 Project
  N 3  marie               Fri Sep 30 13:30 6/79  Welcome
? s mfile [Return] . . . . . Append the current message to mfile.
"mfile" [New file] 12/86 . . Get feedback message.
```

The **s** command saves your indicated message in the specified file. At this point `mfile` contains your current message, message 1, indicated by the > sign.

```
? s 2 3 mfile [Return] . . . Append messages 2 and 3 to mfile.
"mfile" [Appended] 18/286 . Get feedback message.
? s 1-3 mfile [Return] . . . Append messages 1 through 3 to mfile.
"mfile" [Appended] 18/286 . Get feedback message.
? x [Return] . . . . . . . . Exit; mailbox remains the same.
$_. . . . . . . . . . . . . . Return to the shell prompt.
```

At this point, all your messages are appended to `mfile`. You can use the **mailx** command option **-f** to read your mail from `mfile`.

1. *Do not confuse* **mailx** *command option* **-f** *(to which you add a filename to read a specified file) with* **mailx** *command option* **f** *(which displays only the headline of only the current message).*
2. *Do not confuse* **mailx** *command option* **-s** *(which sets the subject field to the specified characters) with* **mailx** *command option* **s** *(which saves your messages in the specified file).*

10.4.4 Ways to Send a Reply

You can send a reply to the sender of a message while reading your mail. This is convenient because you can send a reply right after you read a message.

To send a reply while in the **mailx** read mode, do the following:

```
$ mailx [Return] . . . . . . Read your mail.
mailx version 4.0  Type? for help.
"/usr/student/mail/david": 3 messages 3 new
> N 1  daniel              Fri Sep 30 12:26 6/103 Room
  N 2  susan               Fri Sep 30 12:30 6/107 Project
  N 3  marie               Fri Sep 30 13:30 6/79  Welcome
? R [Return] . . . . . . . . Reply to the current message.
Subject: RE: Room  . . . . . The subject prompt refers to Room.
_ . . . . . . . . . . . . . . The cursor appears at the beginning of  the
                             line, ready for you to type your reply.
Thank you . . . . . . . . . . Compose your reply.
[Ctrl-d]  . . . . . . . . . . End your message.
EOT . . . . . . . . . . . . . Message transmitted.
? . . . . . . . . . . . . . . Ready for the next command.
? R3 [Return]. . . . . . . . Respond to message 3.
? r3 [Return]. . . . . . . . Respond to message 3 and all the others
                             who received a copy of it.
```

Reply command **R** sends your reply only to the author (originator) of the message or a specified list of users. Reply command **r** sends your reply to the author of the message and to all other users who have received the same message.

You can also use the **m** command to send mail to other users. The **m** command places **mailx** into input mode so you can compose your mail. With the **m** command, you can specify one user or a list of users to receive your message.

While in the **mailx** command mode, send mail to a specific user by doing the following:

```
? m daniel [Return]  . . . . Send mail to daniel.
? m daniel susan [Return] . Send mail to daniel and susan.
? x [Return] . . . . . . . . Exit mailx.
$_  . . . . . . . . . . . . . The shell prompt appears.
```

10.5 CUSTOMIZING THE mailx ENVIRONMENT

You can customize the **mailx** environment by setting the **mailx** variables in the `.mailrc` file. To define these variables, you use the **mailx set** command. The **mailx** command also recognizes some of the shell's standard variables.

10.5.1 Shell Variables Used by mailx

Some of the standard shell variables are used by **mailx,** and their values affect the **mailx** behavior: *HOME* (which defines your HOME directory) and *MAILCHECK* (which defines the frequency with which **mailx** checks your mail). For example, if you want arrival of mail in your mailbox to be checked once a minute, then you define *MAILCHECK* as follows:

```
MAILCHECK=60
```

MAILRC is another shell variable used by **mailx.** This variable defines the startup file that **mailx** checks each time you invoke it. If this variable is undefined, then the default value of `$HOME/.mailrc` is used. For example, you can define *MAILRC* in your `.profile` file (the startup file for your login shell) as follows:

```
MAILRC=$HOME/E-mail/.mailrc
```

The *HOME, MAILCHECK,* and *MAILRC* shell variables are used by mailx, but you cannot change them while in mailx.

A large number of **mailx** variables can be manipulated to tailor the **mailx** environment to your specifications. You can set these variables from within **mailx** or set them in `.mailrc`. You use the **set** command to set up **mailx** variables and the **unset** command to reverse their settings. The format of the **set** command and the way you create and set up your `.mailrc` file are similar to those for `.exrc` (the vi editor startup file).

These variables can be set in the startup file `.mailrc` *or internally from* **mailx.**

append When you terminate reading your mail, if **append** is set, then **mailx** appends messages to the end of the `mbox` file instead of the beginning.

You have two choices for the **append** setting:

- Type **set append** and press **[Return]** to add messages to end of `mbox`.
- Type **unset append** and press **[Return]** to add messages to the beginning of `mbox`. This is the default.

asksub When **asksub** is set, **mailx** prompts you for the **Subject:** field. It is set by default.

crt and PAGER The *crt* variable is set to the number of lines on your screen. Messages that have lines more than the set number are piped through the command defined by the *PAGER* variable.

To set up **mailx** so that, if a message is longer than 15 lines, it is piped to the **pg** command and displayed one page at the time, do the following:

- Type **set PAGER='pg'** and press **[Return]**. (This is the default value of the *PAGER* variable, unless your system administrator has changed it.)
- Type **set crt=15** and press **[Return]** to set the number of lines to 15.

You can use the **pg** command options to scan messages up and down (see Chapter 8).

DEAD Your partially composed messages that were interrupted (or messages that for some reason or another are undeliverable) are stored in the specified filename. The default is as follows:

```
set DEAD=$HOME/dead.letter
```

To change the default file specified by the *DEAD* variable, you could type

```
set DEAD=$HOME/E-mail/dead.mail [Return]
```

EDITOR The *EDITOR* variable sets the editor invoked when you use the edit (or **~e**) commands. The default is as follows:

```
set EDITOR=ed
```

To change the default *EDITOR* variable, you could type

```
set EDITOR=ex [Return]
```

It is changed to an editor called *ex*.

escape The *escape* variable lets you change the **mailx** escape character. The default value is the tilde (**~**).

To change the default *escape* variable character to @, you can type

```
set escape=@ [Return]
```

folder The *folder* variable makes a specified directory the standard directory for **mailx**. All mail files are saved in the directory specified by the folder variable. There is no default value for the folder variable.

To specify the `EMAIL` file in the `HOME` directory for mail files, type

```
set folder=$HOME/EMAIL [Return]
```

This command does not create the EMAIL *directory. It just assigns the specified directory to the variable folder. You use the* **mkdir** *command to create the directory.*

header If the *header* variable is set, as it is by default, **mailx** displays the header of messages when you are reading mail.

To unset the header variable, type

```
unset header [Return]
```

MBOX The *MBOX* variable saves your read messages automatically in the specified file. The default filename is `$HOME/mbox`.

To change the specified filename for read messages, you could type

```
set MBOX=$HOME/EMAIL/mbox [Return].
```

Saving a message in another file or using the **x** *command overrides the automatic saving process.*

PAGER The *PAGER* variable is set to a paging command and works with the setup of the *crt* variable. The default paging command is **pg**.

To change the paging command to **more,** type

```
set PAGER=more [Return]
```

record The *record* variable is set to the filename that captures all your outgoing mail automatically. There is no default value for this variable.

To save outgoing mail in `keep`, type

```
set record=$HOME/EMAIL/keep [Return]
```

This command does not create the EMAIL *directory. It just assigns the specified directory to the record variable. You use the* **mkdir** *command to create the directory.*

SHELL The *SHELL* variable is set to the shell program you intend to use. This is applicable when you use **!** or **~!** to issue a command to the shell while you are in the **mailx** environment. The default is the *sh* shell.

To change the default *SHELL* variable, you could type

```
set SHELL=csh [Return].
```

Now the shell is changed to the *C* shell.

VISUAL The *VISUAL* variable is set to the screen editor you intend to use when **mailx** is in the input mode and you use the **~v** command. The default is the vi editor.

10.5.2 Setting Up the `.mailrc` File

The `.mailrc` startup file in your HOME directory contains the commands and variables you set to tailor the **mailx** according to your preference. You can use the vi editor or the **cat** command to create a `.mailrc` file.

Figure 10.7 shows a `.mailrc` sample file. In the sample file, *friends* and *chess* are the names assigned to two groups of user IDs. The sample file is also set up so that if a message is longer than 20 lines, it is piped to the **pg** command (the default PAGER value). Finally, your private mailbox, as set up in the sample file, is in the `EMAIL` directory, and your outgoing mail is saved in a file called `record` in the `EMAIL` directory.

Figure 10.7
The `.mailrc` Sample File

```
$ cat .mailrc
alias friends daniel david marie gabe emma
alias chess emma susan gabe
set crt=20
set MBOX=$HOME/EMAIL/mbox
set record=$HOME/EMAIL/record
$_
```

10.6 COMMUNICATIONS OUTSIDE THE LOCAL SYSTEM

This chapter has discussed the UNIX communication utilities that enable you to send mail to users who have login accounts on your local host. You can also send mail to users on other UNIX computers. If you are on a UNIX network, as big companies and universities usually are, you can use the same commands. However, you must give more information. For example, the destination of your message needs to have the name of the computer (node name on the network) in addition to the user ID of the person on the specified computer. For example, if the computer nodes between you and David on the network were named X, Y, and Z, then you would type the following to send mail to David:

```
mailx X\!Y\!Z\!david
```

The details of communicating with other UNIX systems are outside the scope of this book. The commands in this chapter are all the basic skills you need; the rest is just a matter of looking up the commands in a reference book.

COMMAND SUMMARY

This chapter focused on the UNIX communication utilities and discussed the following commands and options.

mailx

This utility provides the electronic mail system for the users. You can send messages to other users on the system, regardless of whether they are logged on or not.

Option	Operation
-f *filename*	Reads mail from the specified *filename* instead of the system *mailbox*. If no file is specified, it reads from `mbox`.
-H	Displays a list of the message headers.
-s *subject*	Sets the subject field to the string *subject*.

mailx command mode

When you invoke **mailx** to read your mail, it places itself in command mode. The prompt for this mode is the question mark (?).

Command	Operation
!	Lets you execute the shell commands (the shell escape).
cd *directory*	Changes to the specified directory, or to the `HOME` directory if none is specified.
d	Deletes the specified messages.
f	Displays the headlines of the current message.
q	Exits **mailx** and removes the messages from the system mailbox.
h	Displays active message headers.
m *users*	Sends mail to specified *users*.
R *messages*	Replies to the sender of the *messages*.
r *messages*	Replies to the sender of the *messages* and all the other recipients of the same *messages*.
s *filename*	Saves (appends) the indicated messages to the file named *filename*.
t *messages*	Displays (types) the specified *messages*.
u *messages*	Undeletes the specified *messages*.
x	Exits **mailx** does not remove messages from the system mailbox.

mailx tilde escape commands

When you invoke **mailx** to send mail to others, it places itself in input mode, ready for you to compose your message. The commands in this mode start with a tilde (~) and are called tilde escape commands.

Command	Operation
~?	Displays a list of all the tilde escape commands.
~! *command*	Lets you invoke the specified shell *command* while composing your message.
~e	Invokes an editor for editing your message. The editor to be used is defined in the mail variable called *EDITOR*.
~p	Displays the message currently being composed.
~q	Quits input mode. Saves your partially composed message in the file called `dead.letter`.
~r *filename*	Reads the file named *filename* and adds its contents to your message.
~< *filename*	Reads the file named *filename* (using the redirection operator) and adds its contents to your message.
~<! *command*	Executes the specified command and places its output into your message.
~v	Invokes the default visual editor, the vi editor, or uses the value of the mail variable *VISUAL*, which can be set up for other editors.
~w *filename*	Writes the message currently being composed to the specified *filename*.

mesg

This command is set to **n** to prohibit unwanted **write** messages. It is set to **y** to receive messages.

news

This command is used to look at the latest news in the system. It is used by the system administrator to inform others of the events happening.

Option	Operation
-a	Displays all the news items, whether they are in old or new files.
-n	Lists only the names of the news files (headers).
-s	Displays the number of the current news items.

talk

This command is used for terminal-to-teminal communication. The receiving party must be logged on.

wall
This command is used mostly by the system administrator to warn users of some imminent events.

write
This command is used for terminal-to-terminal communication. The receiving party must be logged on.

REVIEW EXERCISES

1. What is the command for terminal-to-terminal communication? What is the key used to signal the end of the communication?
2. What command is usually used by the system administrator to inform users about everyday events?
3. What command broadcasts a message to everyone on the system?
4. How do you make your terminal immune to unwanted messages?
5. What is the command used to read mail?
6. How do you know you have mail?
7. What is the purpose of the `.mailrc` file?
8. What is the command in **mailx** command mode to
   a. display the list of the message headers?
   b. read mail from a specific filename?
   c. change the subject field?
   d. display a list of tilde escape commands?
   e. read a specific file and add it into your message?
   f. invoke the default editor?
9. What is the purpose of the following shell variables?
   a. *DEAD*
   d. *record*
   c. *PAGER*
   e. *MBOX*
   f. *LISTER*
   g. *header*
10. What is the command to communicate with another logged-on user?

Terminal Session

In this terminal session, you practice sending messages to other users. Send the messages to yourself to practice the commands. Then, when you feel comfortable using the commands, select another user as your partner and practice sending mail.

The following exercises are recommended. In order to master the many commands of the UNIX communication utilities, you must spend time at your terminal and try all combinations of the commands.

1. Create a directory `EMAIL` in your `HOME` directory.
2. Create a `.mailrc` file in your `HOME` directory. If one already exists, modify it.
3. Set up **mailx** as follows:
   a. Use the **alias** command to assign names to a group of users.
   b. Set up a file in the `EMAIL` directory to save your outgoing mail automatically.
   c. Set your `mbox` in the `EMAIL` directory.
4. Send the following mail to yourself:

```
So little time and so much to do.
Could it be reversed?
So much time and so little to do.
```

5. While **mailx** is in the input mode, use the vi editor to compose your mail.
6. Read the current date and time and append it to the end of your message.
7. Save the message in a file before transmitting it.
8. Compose a few more messages in the same manner, using vi and other commands, and send them to yourself. This should give you enough messages to practice reading your mail.
9. Read your mail.
10. Use all the commands in the **mailx** command mode, including delete, undelete, save, and so on.
11. Read your mail and exit **mailx** using the **x** command. Then read your mail and exit **mailx** using the **q** command. Observe the UNIX messages.
12. Use **mailx** and look at your `mbox`.
13. Now find another partner and have a chat using the **write** command.
14. Set the **mesg** to **n**, then to **y**, and observe the effect with your partner.
15. Sending mail to others may be one of the less boring parts of learning UNIX. Practice and have fun.