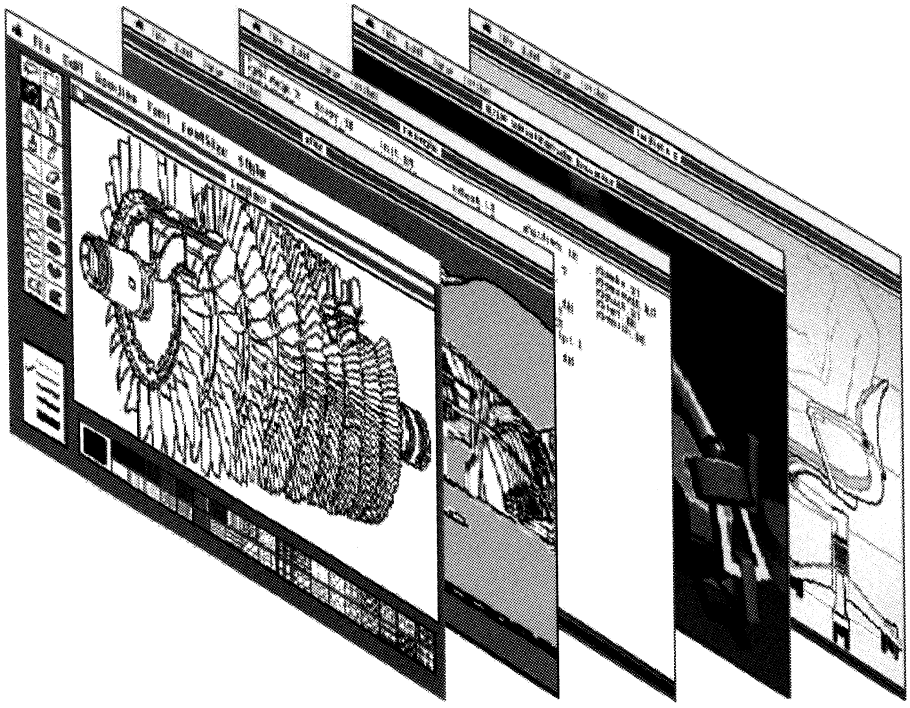
Apple® A/UX™ Communications User's Guide



Copyright

Apple Computer, Inc.
20525 Mariani Ave.
Cupertino, California 95014
(408) 996-1010

Limited Warranty on Media and Replacement

If you discover physical defects in the manuals distributed with an Apple product or in the media on which a software product is distributed, Apple will replace the media or manuals at no charge to you, provided you return the item to be replaced with proof of purchase to Apple or an authorized Apple dealer during the 90-day period after you purchased the software. In addition, Apple will replace damaged software media and manuals for as long as the software product is included in Apple's Media Exchange Program. While not an upgrade or update method, this program offers additional protection for up to two years or more from the date of your original purchase. See your authorized Apple dealer for program coverage and details. In some countries the replacement period may be different; check with your authorized Apple dealer.

ALL IMPLIED WARRANTIES ON THE MEDIA AND MANUALS, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has tested the software and reviewed the documentation, **APPLE AND ITS SOFTWARE SUPPLIER MAKE**

NO WARRANTIES OR REPRESENTATIONS, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO SOFTWARE, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS SOFTWARE IS SOLD AS IS, AND YOU THE PURCHASER ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE.

IN NO EVENT WILL APPLE OR ITS SOFTWARE SUPPLIER BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE SOFTWARE OR ITS DOCUMENTATION, even if advised of the possibility of such damages. In particular, Apple and its software supplier shall have no liability for any programs or data stored in or used with Apple products, including the costs of recovering such programs or data.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

A/UX Communications User's Guide

Contents

Preface

Conventions Used in This Manual

Throughout the A/UX manuals, words that must be typed exactly as shown or that would actually appear on the screen are in `Courier` type. Words that you must replace with actual values appear in *italics* (for example, *user-name* might have an actual value of `joe`). Key names appear in CAPS (for example, RETURN). Special terms are in **bold** type when they are introduced; many of these terms are also defined in the glossary in the *A/UX System Overview*.

Syntax notation

All A/UX manuals use the following conventions to represent command syntax. A typical A/UX command has the form

`command` [*flag-option*] [*argument*]...

where:

`command`	Command name (the name of an executable file).
flag-option	One or more flag options. Historically, flag options have the form -[*opt*...] where *opt* is a letter representing an option. The form of flag options varies from program to program. Note that with respect to flag options, the notation `[-a][-b][-c]` means you can select one or more letters from the list enclosed in brackets. If you select more than one letter you use only one hyphen, for example, `-ab`.
argument	Represents an argument to the command, in this context usually a filename or symbols representing one or more filenames.

[]	Surround an optional item.
...	Follows an argument that may be repeated any number of times.
`Courier type`	anywhere in the syntax diagram indicates that characters must be typed literally as shown.
italics	for an argument name indicates that a value must be supplied for that argument.

Other conventions used in this manual are:

<CR>	indicates that the RETURN key must be pressed.
^*x*	An abbreviation for CONTROL-*x*, where *x* may be any key.
cmd(*sect*)	A cross-reference to an A/UX reference manual. *cmd* is the name of a command, program, or other facility, and *sect* is the section number where the entry resides. For example, `cat`(1).

Chapter 1

Introduction

The A/UX operating system includes a number of programs that allow you to communicate with other users, with other A/UX systems, and with machines running different operating systems. *A/UX Communications User's Guide* can help you use the A/UX facilities to communicate with the outside world.

This guide assumes that you're reasonably comfortable using A/UX but that you've perhaps done little or no programming. If you're just starting out with A/UX or don't have a lot of computer experience, you'd be wise to go through *Getting Started With A/UX* before experimenting with these facilities.

A/UX Communications User's Guide is divided into five chapters:

Chapter 1 summarizes the contents of the manual.

Chapter 2 discusses the `mail` program, the facility you use to send and receive electronic mail. (Note that in the A/UX standard distribution, `mail` is set to `mailx`, which includes greater functionality. See `mailx`(1) in *A/UX Command Reference*.) A variety of commands allow messages to be saved, edited, included in other messages, or "carbon copied" (openly or blindly) to other users. Also, there is a facility for saving messages according to subject in **folders**, which can be manipulated from within `mail`.

Chapter 3 covers B-NET, BSD-derived network software that communicates between machines on a local area network. You can use B-NET to transfer files between machines (`rcp`), to execute commands on different machines on the network (`remsh`), and to send and receive mail across the network on different machines (network `mail`). In addition, you can use it to log in remotely to another machine (`rlogin`) and compute remotely as if you were on a local machine. Using `rlogin`, you can toggle between the remote and the local machine. All of these require that the remote system be running an operating system like A/UX which supports the BSD networking

package. Chapter 3 also describes `telnet` (which is similar to `rlogin` but does not require that the remote system run an operating system similar to A/UX); `ftp` (which is similar to `rcp`, but does not require that the remote system run an operating system similar to A/UX); and remote `talk` (a facility allowing two-way, real-time conversations between two users in separate on-screen windows.) While describing these commands, special attention is paid to the treatment of shell metacharacters, pipelines, output redirection, and quoting, relative to both local and remote machines. General issues, such as understanding the effect of network permissions, setting up a `.rhosts` file, and using short command forms and system nicknames, are also discussed in this chapter. B-NET performs its own error checking and error correction.

Chapter 4 discusses UUCP, another network system that links machines using modems and other UUCP systems. You can use UUCP to transfer files, to send and receive mail, and to execute commands on different machines on the network. UUCP is a local facility set up by system administrators. You must have access permission from other systems on the UUCP network to use it. Access to remote machines is usually limited to a directory designated as the public area. UUCP performs its own error checking and error correction.

Chapter 5 describes `cu`, a program that allows you to communicate with another computer, usually over serial lines. The remote system need not run A/UX. Once established, the remote connection functions exactly like a local one. `cu` does no error checking.

Chapter 2
Using `mail`

Contents

Figures

Chapter 2

Using `mail`

1. Introduction

This chapter describes how to use the `mail` facility to send and receive mail. It assumes you're familiar with the A/UX shell, a text editor, and some of the common A/UX commands. (Note that in the A/UX standard distribution, `mail` is set to `mailx`; see `mailx`(1) in *A/UX Command Reference*.)

`mail` sends and receives messages on your local machine and across such networks as ARPANET, UUCP, and B-NET. It divides the mail you receive into messages and allows you to deal with them in any order. It provides `ed`-like commands for manipulating messages and sending mail.

The mail system accepts incoming messages for you and collects them in a file called the **system mailbox**. When you log in, the system notifies you of any messages waiting in your system mailbox.

When you read your mail, `mail` opens your system mailbox file, separates that file into individual messages, and marks each message with its author's login name and the date the message was sent. You may then read, reply to, delete, or save these messages.

The following sections describe all of the `mail` commands and facilities. Section 9 summarizes the `mail` commands for your reference.

2. Sending mail

To send a message to someone, type

`mail` *address*

address may be a login name (see below) or it may be a network address (see Section 7, "`mail` and Networks"). For example, to send

mail to a user whose login name is `fred`, type the following:

```
mail fred
```

The system responds:

```
Subject:
```

Fill in the subject, if you wish, and press RETURN. If you don't want to include a subject, just press RETURN.

Type your message. At the end of your message, type an *eof* at the beginning of a line. The A/UX standard distribution defines the *eof* sequence as CONTROL-d. See `stty`(1) in *A/UX Command Reference* for information about the *eof* sequence, or type `stty -a` for the current setting for *eof* and other definable sequences on your system. `mail` prints `EOT` (end-of-transmission) and returns you to the command prompt. When `fred` next logs in, he receives the message

```
You have mail.
```

The mail `fred` receives consists of the message you typed, preceded by a line telling who sent the message (your login name) and the date and time you sent it.

2.1 Canceling a message

If, while composing a message, you decide not to send it, you can cancel the message by typing an *interrupt*. The A/UX standard distribution defines the *interrupt* sequence as CONTROL-c. See `stty`(1) in *A/UX Command Reference* or type `stty -a` for information about the *interrupt* sequence. `mail` prints the message

```
(Interrupt -- one more to kill letter)
```

Type an *interrupt* a second time to cancel the letter. Your partial letter is saved in the file `dead.letter` in your home directory. Once you have sent mail to someone, there is no easy way to retrieve it, so be careful.

2.2 Sending mail to several people

If you want to send the same message to several people, list their login names on the command line. For example,

```
$ mail sam bob john
Subject: Tuition

Tuition fees are due next Friday.
Don't forget!!
```

followed by an *eof* sends the reminder to `sam`, `bob`, and `john`.

3. Receiving mail

If there is mail in your system mailbox, you'll get the following message when you log in:

```
You have mail.
```

You may then view the mail by typing

```
mail
```

`mail` displays its version number and date, lists the messages you have waiting, and waits for your command. For example, `mail` might print the following list of messages:

```
>   N 1 root      Wed Sep 21 09:21  "Tuition"
    N 2 sam       Tue Sep 20 22:55
```

The > indicates the current message. The first field is the **message status**. This field is blank if you've read the message. `N` means the message is new (that is, sent since you last read your mail). `U` means the message is unread (previously listed but unread).

The second field is the **message number**. The messages are numbered sequentially, starting with 1. The third field tells you who sent the message. The fourth field is a timestamp indicating when the message was sent. The fifth field is the **message subject**. This is the heading the sender gave the message.

3.1 Displaying a message

You may use the `type` command to display a message. Examine the first message by giving the command

```
type 1
```

The message might look like this:

```
Message  1:
From root  Wed Sep 21 09:21:45 1978
Subj: Tuition

Status: R

Tuition fees are due next Wednesday.
Don't forget!!
```

You can abbreviate `type` to `t` or leave the command out altogether and simply specify the message number; `mail` will assume you want the `type` command. If you don't specify the message number, `mail` assumes you are referring to the current message. Pressing RETURN at the `mail` prompt (`?` by default) displays the next message. You will learn how to change this prompt later in this chapter.

3.2 Saving messages

Messages that you have read but not deleted are automatically saved in the file `mbox` in your login directory. The `save` command saves messages in the file specified instead of saving them in `mbox`. For example,

```
save 1 mailbox
```

saves message `1` in the file `mailbox`.

```
save 1
```

saves message `1` in a filename derived from the name of the sender. `s` is a synonym for `save`.

3.3 Deleting mail

Messages that have been read but not saved or deleted are saved in the file `mbox` in your login directory when you leave `mail`. If you don't want to save a message, use the `delete` command (it may be abbreviated to `d`) to remove it. For example, the command

```
delete 1
```

deletes message `1`. `mail` won't display deleted messages. The deleted message disappears altogether, along with its number. Use the `undelete` command to retrieve a deleted message.

3.4 Message lists

Many `mail` commands accept a **message list** as an argument. A message list is a list of message numbers, ranges, and names, separated by spaces or tabs. A message list allows you to address a command to many messages at once. Commands that use message lists ignore deleted messages, except for the `undelete` command, which only applies to deleted messages.

A **message number** may be either the number of the message (listed in the second field) or one of the following special characters:

^ (first message)

. (current message)

$ (last message)

When you enter `mail`, the current message is the first message.

A **message range** is two message numbers separated by a hyphen (–). For example, to display the first four messages, type

```
type 1-4
```

To display all the messages from the current to the last, type

```
type .-$
```

A **message name** is the login name of a user who sent a message. If you specify a message list with only user names, `mail` selects every message sent by one of those users. If you specify both a range and a message name, `mail` selects every message sent by that user *within that message range.*

For example, to print every message sent to you by `root`, type

```
type root
```

The asterisk character (*) specifies every message. For example,

```
type *
```

prints all undeleted messages,

```
delete *
```

deletes all messages, and

```
undelete *
```

undeletes all deleted messages.

The slash character (/) searches for a word in the subject line. For example, if you want to print the headers of all messages containing the word PASCAL, type

```
from /pascal
```

Note that subject searching ignores uppercase/lowercase differences.

3.5 Replying from within `mail`

The `reply` command allows you to reply to a message. You may abbreviate this command to `r`. When you use `reply`, `mail` begins a message addressed to the sender. When you've finished your message, type an *eof* at the beginning of a line to end it. (The A/UX standard distribution defines the *eof* sequence as CONTROL-d; see `stty`(1) in *A/UX Command Reference* for more information.) `mail` prints `EOT` and the `mail` prompt. It's then ready to accept another command.

If you had just read the sample message, you could reply by typing:

```
reply
```

`mail` responds with the following:

```
To: root
Subject: Re: Tuition fees
```

`mail` then waits for you to enter your reply. Note that it copies the subject header from the original message so that correspondence about a particular matter retains the same subject heading.

If the original message was mailed to several people (listed in the `To:` header), `mail` sends your reply to the same people. Similarly, if the original message contains a `Cc:` (carbon copies to) field (described under "Manipulating `mail` Messages"), `mail` sends your reply to those users, too. `mail` will not send the message to you, even if you appear in the `To:` or `Cc:` field, unless you ask to be included by setting the `metoo` option.

The `reply` command is useful for sustaining extended conversations over the message system, with other "listening" users receiving copies of the conversation.

Sometimes you receive a message sent to several people and want to reply only to the person who sent it. The `Reply` command (with a capital `R`) sends your reply only to the message sender.

3.6 Mailing from within `mail`

You can send mail from within the `mail` facility using the `mail` command. The form of this command is

```
mail username
```

For example, to send a message to `frank`, type

```
mail frank
```

Then, after entering a subject (if you wish), type your message:

```
This is to confirm next Friday's 4 pm meeting.
```

Start a new line and type an *eof* to send the message (see `stty`(1) for information about the *eof* sequence).

You can abbreviate the `mail` command to `m`.

3.7 Accessing the shell from within `mail`

To execute a shell command without leaving `mail`, type the command preceded by an exclamation point (just as in the text editor). For example,

```
!date
```

prints the current date without leaving `mail`.

3.8 Printing a summary of commands

The `help` command prints a summary of the `mail` commands. The `help` command can be abbreviated to `hel`.

3.9 Leaving `mail`

Leave `mail` with the `quit` command (abbreviated to `q`). `quit` saves the messages you have read (but have not deleted or saved) in the file `mbox` in your home directory. `quit` discards deleted messages, and leaves unread messages in your system mailbox.

Type `x` (short for `exit`) to leave `mail` quickly without altering either your system mailbox or `mbox`. This is useful if you delete a message and then want it back.

3.10 Reading mail from a different file

To read mail from a file other than your system mailbox, use the `-f` option when you start `mail`. For example, to read the mail in your `mbox` file in your home directory, type the following:

```
mail -f mbox
```

By default, `mail -f` by itself reads your `mbox` file.

4. Setting up your `mail` environment

This section describes how to set up your `mail` environment.

4.1 The `set` command

You may tailor many features of `mail` with the `set` command. You can find a list of all the `mail` variables you may set or unset in the "`mail` Variables" section at the end of this chapter.

Toggle variables are either on or off. The `set` command turns a toggle variable on. The `unset` command turns it back off. For example, the `askcc` variable tells `mail` to prompt you for a `Cc:` (carbon copy) for your messages. To set the `askcc` variable, type:

```
set askcc
```

Another toggle variable is `hold`. The `hold` variable keeps messages in your system mailbox after you read them instead of moving them to the `mbox` file in your home directory.

Some `mail` variables, however, are on by default. To turn them off, use the command form

```
set novariable
```

For example, the `save` variable, which saves canceled messages in the `dead.letter` file in your home directory, is on by default. Typing

```
set nosave
```

would turn it off.

Not all `mail` variables can be toggled on or off. These variables must be given a value. For example, the `SHELL` variable tells `mail` which shell you want to use. To tell `mail` to use the C shell, type

```
set SHELL=/bin/csh
```

No spaces are allowed in `SHELL=/bin/csh`. For more information about shells, please refer to *A/UX User Interface*.

The command `set` by itself gives a list of which `mail` variables are currently set.

One helpful `mail` variable is `crt`. The `crt` variable paginates messages that are too long to fit on your screen. To set the `crt` variable, tell `mail` how many lines your terminal has. For example, usually you would type

```
set crt=24
```

This sends messages longer than 24 lines through the `pg` program. `pg` prints a screenful of information, then displays the prompt

```
:
```

Press RETURN to see the next screenful. See `pg`(1) in *A/UX Command Reference* for more information.

4.2 Aliases

An **alias** is a name that stands for one or more user names. For example, if you were working on a project with several other people, you could define an alias for all the members of the project and then send mail to the alias. To tell `mail` that the alias `project` stands for four members of your project: `sam`, `sally`, `steve`, and `susan`, you'd type

```
alias project sam sally steve susan
```

To send mail to these four people, you'd type

```
mail project
```

This command also provides a convenient alias for someone whose user name is hard to type. For example, if a user named Throckmorton Vandersleeve had the login name `throckmorton`, you might use

```
alias tv throckmorton
```

This allows you to send mail using the shorter name, `tv`.

4.3 Setting up a permanent `mail` environment

The `set` and `alias` commands let you make changes in how you use `mail`. Each time you leave `mail`, however, these changes are lost. To make an `alias` or `set` command part of your permanent `mail` environment, use any A/UX text editor to include it in the `.mailrc` file in your home directory.

Your `.mailrc` file might look like this:

```
set asksub nosave SHELL=/bin/csh
alias project sam sally steve susan
```

In the example, `nosave` turns off the `save` variable, which is on by default. The example also demonstrates that you can set many `mail` variables in the same `set` command.

You can continue long lists in the `alias` command onto the next line by starting the next line with a space or tab.

You can also put comments in `.mailrc` by starting them with a # character.

Your system administrator can also set `mail` variables and aliases for everyone on your system. The systemwide `mail` environment instructions are in the file `/usr/lib/mailx/mailx.rc`.

5. Maintaining folders

`mail` can group messages together in folders. Each folder is a single file and all your folders are in one directory. To tell `mail` where your folder directory is, put a line with the following form into your `.mailrc` file:

```
set folder=letters
```

Because the directory name (`letters`) doesn't begin with a slash (/), `mail` assumes that it's a subdirectory of your home directory. For example, if your home directory is `/usr/person`, the above example tells `mail` that your folder directory is in `/usr/person/letters`.

You can use a folder name anywhere you would use a filename. To indicate that you're referring to a folder, however, you must precede the folder name with a +. For example, to `save` a message in a folder named `classwork`, type

```
save +classwork
```

If the `classwork` folder doesn't exist, `mail` creates it.

To copy a message into a folder without removing it from your system mailbox, use the `copy` command. For example, to copy the current message into the `classwork` folder and leave a copy in your system mailbox, type

```
copy +classwork
```

You can use all of the `mail` commands to operate on folders.

To read a folder, use the `folder` command. For example, to read the `classwork` folder, type

```
folder +classwork
```

To find out which folder you're currently editing, type

```
folder
```

To list your current set of folders, type

```
folders
```

To specify a folder when you start `mail`, use the `-f` flag option described earlier. For example, if you type

```
mail -f +classwork
```

`mail` reads your `classwork` folder instead of your system mailbox.

6. Tilde escapes

Tilde escapes let you enter `mail` commands while you're typing a `mail` message.

A tilde escape consists of a tilde (~) at the beginning of a line, followed by a single-character command (single-character abbreviations are listed with full command names in the "Summary of Commands" section of this chapter). Type an *interrupt* to stop any tilde escape (be careful not to type an *interrupt* twice: this will kill your letter). (The A/UX standard distribution defines the *interrup* sequence as CONTROL-c; see `stty`(1) in *A/UX Command Reference* for more information.)

For example, to print the message you are currently entering, type

```
~p
```

This displays a line of dashes, the recipients of your message, and the text of the message.

6.1 Editing a `mail` message

To edit your message, call up the line editor with

```
~e
```

After modifying the message, write it out and quit the editor. `mail` prints

```
(continue)
```

Continue typing your message, or type an *eof* to end the message. (The A/UX standard distribution defines the *eof* sequence as CONTROL-d; see `stty`(1) in *A/UX Command Reference* for more information.) To use an editor other than the default `ed` editor, set the `EDITOR` variable. For example, to use the `ex` editor, include the following line in your `.mailrc` file:

```
set EDITOR=/usr/bin/ex
```

If you want to use `vi` on the message you are currently entering, use this escape:

```
~v
```

`~v` works like `~e`. To use a visual editor other than `vi`, set the `VISUAL` variable in your `.mailrc` file.

6.2 Appending a file to a `mail` message

To append a file to the message you are currently entering, use the following escape:

`~r` *filename*

The filename may contain shell metacharacters like `*` and `?`. `mail` returns a message if the file doesn't exist or can't be read. When `mail` appends the file, it prints the number of lines and characters appended. After this, you may continue adding text to your message.

The escape

```
~d
```

reads in the file `dead.letter` in your home directory. (This file is created when you cancel a message using *interrupt*. The A/UX standard distribution defines the *interrupt* sequence as CONTROL-c; see `stty`(1) in *A/UX Command Reference* for more information.)

6.3 Manipulating `mail` messages

Save the message you are currently entering in a file with the following escape:

```
~w filename
```

You may use shell metacharacters in the filename. As in `~r`, `mail` prints the number of lines and characters written to the file, after which you may continue appending text to your message.

To append the current message to the message you are currently entering, type

```
~m
```

You may read a message into the message you are currently entering with

```
~m message-number
```

You may name any undeleted message, or list of messages. For example,

```
~m 4
```

reads message 4 into the message you are currently entering, shifted to the right by one tab stop. If you don't want the message shifted to the right by a tab stop, use the `~f` escape. This is the usual way to forward a message.

If you want to add recipients to the message you are currently entering, use the following command:

```
~t name1 name2 ...
```

The original recipients still receive the message; you cannot remove someone from the recipient list with `~t`.

If you want to add a subject line to your message, use the following command:

~s *arbitrary-string-of-text*

This creates a new subject with *arbitrary-string-of-text* (which can include spaces). You can see what the message will look like by using ~p.

If you want to list certain people as carbon copy recipients rather than as direct recipients, use the command

~c *name1 name2 ...*

Those named will be added to the `Cc:` list.

6.4 Editing `mail` message fields

If you want to edit the three fields `To:`, `Subj:`, and `Cc:` (as opposed to only adding to them), use the command

~h

This prints `To:` and the current list of recipients and leaves the cursor at the end of the line. Characters you type are appended to the end of the current list of recipients. You may also use your *erase* character to erase back into the list of recipients, or use your *kill* character to erase them altogether. For example, if your *erase* and *kill* characters are the # and @ symbols, type

```
~h
To: root kurt####bill
```

This changes `root kurt` to `root bill`. (See `stty`(1) for information about the *erase* and *kill* sequences, or type `stty -a` for the current settings for your system.) When you finish editing the `To:` field, press RETURN to advance to the `Subj:` field, which you can edit in the same way. Pressing RETURN again allows you to edit the `Cc:` field. Press RETURN again to continue appending text to the end of your message.

6.5 Escaping to the shell

To escape temporarily to the shell, use the ~! escape:

~! *command*

This executes *command* and returns you to `mail` without changing your message. To filter your message through a shell command, use the ~| escape:

~| *command*

This pipes your message through the command and uses the output as the new text of your message. If the command produces no output (for example, due to an error) `mail` retains the old version of your message. A frequently used filter is the command `fmt`, which formats outgoing mail.

6.6 Miscellaneous tilde escapes

To escape temporarily to the `mail` command mode, use the following:

~: *mail-command*

For example, the following command is useful for displaying the message to which you're replying:

```
~:t
```

The ~: escape is also useful for setting `mail` variables and modifying aliases.

If you're typing a message containing a line beginning with a tilde, you must double it. For example,

```
~~This line begins with a tilde.
```

This sends the line

```
~This line begins with a tilde.
```

Finally, the escape

```
~?
```

prints a summary of tilde escapes.

On some terminals (particularly those with no lowercase characters) tildes are difficult to type. `mail` allows you to change the escape character with the `escape` option. For example, you may type

```
set escape=]
```

to use a right bracket instead of a tilde. If you need to send a line

beginning with right bracket, you double it, just as you would for ~. Changing the escape character means that ~ no longer has any special meaning to `mail`.

7. `mail` and networks

You can use `mail` to send messages to people on other machines. When you address mail to a login name, the mail is sent to a person on your machine. If you want to send mail to someone on another machine, you need to include some more information in the address.

There is additional information about networks in the last three chapters of this guide.

7.1 UUCP

(UNIX-to-UNIX copy) is an A/UX utility that communicates over serial lines and telephone lines.

To send mail to someone via UUCP, you need to know the route your message will take. It is possible that your machine and the destination machine are directly connected; on the other hand, your machine and the destination machine may route messages through one or more intermediary machines. For the sake of example, let's say you want to send mail to *username* on the machine `rhost6`, but your machine has no connection to `rhost6`. Your machine does, however, talk to a machine named `rhost1`, and it turns out that `rhost1` connects to `rhost2`, `rhost2` connects to `rhost3`, `rhost3` connects to `rhost4`, and `rhost4` connects to `rhost5`, which in turn is connected to `rhost6`. In this example, the route would be expressed as

`rhost1!rhost2!rhost3!rhost4!rhost5!rhost6!`*username*

which you would then use as the mail address. Figure 2-1 illustrates this sample UUCP route. Note that with UUCP your message does not necessarily take a geographically direct route.

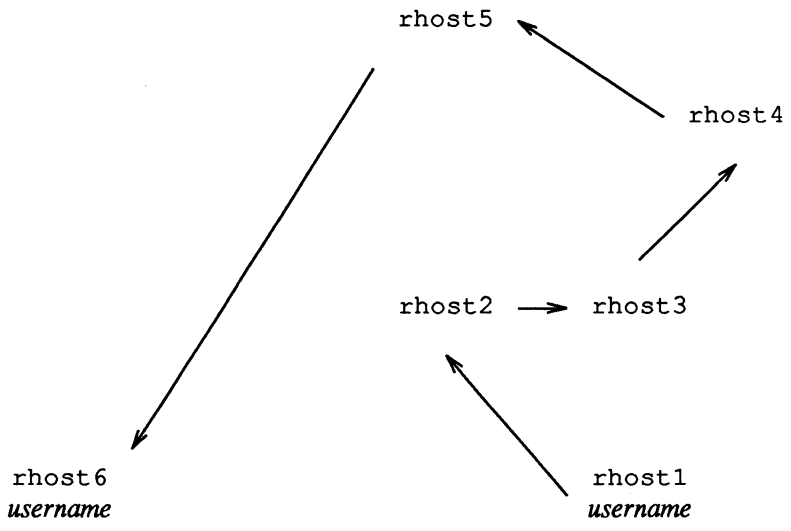
The `hostname` or `uuname -l` command tells you a local machine's name. Have someone on the remote machine type

```
hostname
```

or

```
uuname -l
```

Figure 2-1. Sample indirect route with UUCP



The machine prints its name. For example, it may print

```
sys10
```

To determine whether or not your machine has a direct link to the desired destination, type the command

```
uuname
```

`uuname` returns one or more hostnames. If `uuname` returns the name of the destination machine, you can address your message as follows:

rhost!*username*

If `uuname` does not list the name of the destination machine, it is possible to route your message through one of the machines named, provided it maintains a connection (direct or indirect) to the final destination. However, indirect routing is difficult to do unless you

know the name of every machine the message has to pass through. Your system administrator can help you determine correct UUCP routes.

7.2 B-NET

If you want to send a message to a recipient on your local area network running B-NET, use this syntax:

```
mail username@rhost
```

rhost is the machine name and *username* is the user's login name. (You can also use the syntax *rhost*:*username*.) Unlike UUCP, you do not need to know the names of the intermediate machines. For example, you could send `kirk` his mail with

```
mail kirk@doc
```

and you could send `scotty` his mail with

```
mail scotty@bashful
```

even though your machine is connected directly to `kirk`'s but not to `scotty`'s.

7.3 DDN

If your machine is connected to the wide area Defense Data Network (DDN, which consists of the ARPANET and MILNET), you can send messages to people on the DDN using a name with the form

username`@`*rhost*

where *username* is the login name of the person you're trying to reach, and *rhost* is the name of the remote machine where they log in on the DDN. For example,

```
mail kirk@doc
```

sends mail to `kirk` on the machine `doc`.

7.4 Forwarding mail

If you have accounts on several machines and want to receive your mail on only one machine, you can forward your mail by creating a file called `.forward` in your home directory on each of the forwarding machines. This file contains the address of the machine where you want to receive your mail. For example, if `fred` has accounts on both

marilyn and doc (connected to each other with B-NET) and wants to receive all his mail on doc, he would include the following line in his .forward file on marilyn:

```
doc!fred
```

8. Sending mail to files and programs

You can send messages directly to files or to programs. The next sections provide details.

8.1 Sending mail to files

You might want to send mail to a file to maintain a journal or keep a record of mail sent to a group of users. A recipient name with / in it or beginning with a + is assumed to be the pathname of a file. If the file already exists, the message is appended to the end of the file. To refer to a file in your current directory, precede the name with ./. For example, to send mail to the file memo in the current directory, type

```
mail ./memo
```

A name beginning with a + is assumed to be a folder name in your folder directory. Using the previous alias example, you might give the command

```
alias project sam sally steve susan +project
```

Then, all mail sent to project would be saved in the folder project in your folder directory and be sent to the members of the project. You may read this file by typing this command:

```
mail -f +project
```

8.2 Sending mail to programs

Sometimes you may want to send mail directly to a program. For example, you might write a project billboard program and want to access it using mail. To send messages to the program, use the following format:

```
mail "|program"
```

Be sure to surround the | and the program name in double quotes and don't leave any spaces between | and the program name when you type it on the command line or include it in the alias command.

8.3 Sending large files

This next command requires a UUCP network for remote hosts. For more information see Chapter 4, "Using UUCP."

The `uuto` command sends large files to people on both local and remote machines. The format of a `uuto` command is

`uuto` *filename rhost*`!`*username*

where *filename* is the name of the file you're sending, *rhost* is the machine you're sending it to, and *username* is the login name of the person you're sending it to. If you're using the C shell, you must escape the `!` with a backslash or quotes. If you're sending a file to someone on your machine, the format is

`uuto` *filename username*

When your file arrives at its destination, `uuto` informs your recipient.

If you send the file using the format

`uuto -m` *filename rhost*`!`*username*

`uuto` also informs you that your file is delivered.

Before you send a file using `uuto`, make sure the file has read permission for others and that the directory containing the file has both read and execute permission for others (see `chmod`(1) in *A/UX Command Reference* for more information).

Type `uustat` to check the status of a file sent with `uuto`. This will tell you if the file has reached its destination.

The `uupick` command retrieves files sent with `uuto`. `uuto` sends mail informing the recipient that a file is waiting. Typing `uupick` searches the public directory for any file sent to you. If a file is found, you're first prompted with `?`, the hostname, and the filename. `uupick` then waits for your response. Several possible responses are

`m`	Save the file in the current directory.
`m` *directory*	Save the file in the specified directory.
`d`	Delete the file.

`q` Exit from `uupick`. Any unmoved or undeleted files will wait in the public directory until the next time you use the `uupick` command.

9. Summary of commands

9.1 Flag options

Specify flag options when you start `mail` using this command form:

`mail` *flag-option*

`mail` recognizes the following flag options:

`-d` Turn on debugging output.

`-e` Check if there is mail. `mail` prints nothing and exits with a successful return code if there is mail to read.

`-f`[*filename*] Read messages from *filename* instead of the system mailbox. If you don't specify a *filename*, `mail` reads the `mbox` file in your home directory.

`-F` Record the message in a file named after the first recipient. Overrides the `record` variable, if set (see "`mail` Variables" later in this chapter).

`-H` Print header summary only.

`-i` Ignore *interrupts* if configured as RUBOUT or DELETE. Useful on noisy phone lines, which sometimes generate interrupts. It's usually more effective to leave your interrupt sequence configured to CONTROL-c, as defined by the A/UX standard distribution. See `stty`(1) in *A/UX Command Reference* for more information about redefining the interrupt character. See also `ignore` in "`mail` Variables" later in this chapter.

`-n` Do not read the system default `mailx.rc` file.

`-N` Do not print initial header summary.

`-s` *subject* Set the Subject header field to *subject*. If the subject contains blanks, enclose it in quotes.

`-u` *user*
: Read *user*'s system mailbox. This works only if *user*'s mailbox is not read protected.

`-U`
: Convert `uucp`-style addresses to internet standards. Overrides the `conv mail` variable.

`-h` *number*
: The number of network **hops** made so far. This is provided for network software to avoid infinite delivery loops.

`-r` *address*
: Pass *address* to network delivery software. The *address* and `-r` are sent along to the mail delivery system. All tilde commands are disabled.

9.2 Message lists

Each message is assigned a sequential number, and there is a "current" message, marked by a > in the header summary. If you don't specify a message number, `mail` assumes you're referring to the current message. Many commands operate on an optional list of messages (*msglist*). A *msglist* is a list of message specifications separated by spaces, which may include the following:

n
: Message number *n*

`.`
: The current message

`^`
: The first undeleted message

`$`
: The last message

`*`
: All messages

n–*m*
: An inclusive range of message numbers

user
: All messages from *user*

/string
: All messages with *string* in the subject line (case ignored)

`:`*c*
: All messages of type *c*, where *c* is one of the following:

  `d` deleted messages

  `n` new messages

o old messages

r read messages

u unread messages

9.3 Commands

The following is a complete list of `mail` commands:

`!`*shell-command*

Execute the following shell command (see `shell` in "Environment Variables" below).

`#` *comment*

Begin comment. This may be useful in `.mailrc` files.

`=` Print the current message number.

`?` Print a summary of commands.

`a[lias]` *alias name ...*
`g[roup]` *alias name ...*

Define a name to stand for a set of other names. This is useful when you send messages regularly to a group of people and want to avoid retyping their names.

`alt[ernates]` *name ...*

If you have accounts on several machines, you may want to use `/usr/lib/aliases` on all of them except one to direct your mail to a single account. The `alt (alternates)` command informs `mail` that each of these other addresses is really you. When you `reply` to messages sent to one of these alternate names, `mail` does not send a copy of the message to this other address (which would simply be directed back to you by the alias mechanism). Typing `alt` with no argument lists the current set of alternate names. `alt` is usually used in the `.mailrc` file.

`cd` [*directory*]
`ch[dir]`[*directory*]

Change current directory. If you don't specify a *directory*, `cd` changes to your home directory.

`c[opy]`[*msglist*][*filename*]

Copy messages to the file without removing it from your system mailbox. Otherwise, it is equivalent to the `s` (`save`) command.

`C[opy]`[*msglist*]

Save the specified messages in a file (whose name is the login name of the author of the saved messages) without removing them from your system mailbox. Otherwise, it is equivalent to the `S` (`Save`) command.

`d[elete]`[*msglist*]

Delete a list of messages. You can reclaim deleted messages with the `undelete` command. If you set `autoprint`, `mail` automatically displays the next message (see "`mail` Variables").

`di[scard]`[*header-field* ...]
`ig[nore]`[*header-field* ...]

Add the specified *header-field*s to the list of fields not to print when you print a message. (A *header-field* may be any string of printable ASCII characters, except blanks, tabs, and `:`. A *header-field* is terminated by a colon (`:`). Some standard header fields are `Date:`, `From:`, `To:`, `Cc:`, `Bcc:`, `Sender:`, and `Message-ID:`. See RFC822 for more information.) For example, you could ask `mail` not to print the `status` field of your messages. The ignored fields are included when you save, print, or type these messages. Typing `discard` with no arguments lists the current set of ignored fields.

dp [*msglist*]
dt [*msglist*]

Delete the current message and print the next message. This is useful for quickly reading and disposing of mail.

ec[ho] *string* ...

Echo the given strings.

e[dit][*msglist*]

Edit the given messages. The EDITOR variable specifies the name of the editor (see "mail Variables"). The default editor is ed.

[e]x[it]

Exit from mail without changing the system mailbox or the file you were reading. This is useful if you accidentally delete several messages (see also quit).

fi[le][*filename*]
fold[er][*filename*]

Write out the changes to the current file and switch to a new mail file or folder. Some special conventions are recognized for the following names:

%	The current system mailbox
% *user*	The system mailbox for *user*
#	The previous file
&	The current mbox

The default file is the current system mailbox.

Typing folder without an argument tells you which file you are currently reading.

folders List the names of the folders in your folder directory (see "mail Variables").

fo[llowup][*message*]

Respond to a message, and record the response in a file

whose name is the login name of the author of the message. Overrides the `record` variable, if set (see also the `F`, `S`, and `C` commands in this section and `outfolder` in "`mail` Variables").

`F[ollowup]`[*msglist*]
Respond to the first message in the *msglist*, and send the response to the author. The subject line is taken from the first message and the response is recorded in a file whose name is the login name of the author. See also the `fo`, `S`, and `C` commands in this section and `outfolder` in "`mail` Variables."

`f[rom]`[*msglist*]
Print the header summary for the specified messages.

`g[roup]` *alias name* ...
`a[lias]` *alias name* ...
Define a name to stand for a set of other names. This is used when you send messages regularly to a certain group of people and want to avoid retyping their names.

`h[eaders]`[*message*]
Print the current list of message headers surrounding the specified message. The `screen` variable sets the number of headers per page (see "`mail` Variables"). See also the `z` command.

`hel[p]` Print a summary of commands.

`ho[ld]`[*msglist*]
`pre[serve]`[*msglist*]
Hold the specified messages in the system mailbox when you `quit`.

`i[f]` *action*
mail-commands
`el[se]`

mail-commands
en[dif]

Specify where *action* can be either s or r. If you specify s (send), mail executes the following *mail-commands*, up to an else or endif, when you're sending mail. If you specify r (receive), mail executes the following *mail-commands*, up to an else or endif, when you're receiving mail. Useful in the .mailrc file.

ig[nore][*header-field* ...]
di[scard][*header-field* ...]

Add the specified *header-fields* to the list of fields not to print when you print a message. (A *header-field* may be any string of printable ASCII characters, except blanks, tabs, and :. A *header-field* is terminated by a colon (:). Some standard header fields are Date:, From:, To:, Cc:, Bcc:, Sender:, and Message-ID:. See RFC822 for more information.) For example, you could ask mail not to print the status field of your messages. The ignored fields are included when you save, print, or type these messages. Typing ignore with no arguments lists the current set of ignored fields.

l[ist] Print the names of all mail commands.

m[ail] *loginname* ...

Mail a message to one or more people.

mb[ox][*msglist*]

Save the specified messages in mbox in your home directory when you quit. See mbox (in "mail Variables") for a description of this file. See also the exit and quit commands.

n[ext][*message*]

Display the next message. If you specify a message list, mail prints the next such message.

`pi[pe]`[*msglist*] [*shell-command*]
`|` [*msglist*][*shell-command*]
Pipe the message through the given *shell-command*. If you don't specify any arguments, `mail` pipes the current message through the command specified by the `cmd` variable. If you set the `page` variable, `mail` inserts a form feed character after each message (see "`mail` Variables").

`pre[serve]`[*msglist*]
`ho[ld]`[*msglist*]
Hold the specified messages in the system mailbox when you `quit`.

`P[rint]`[*msglist*]
`T[ype]`[*msglist*]
Display the specified messages, including all header fields. Overrides the `ignore` command.

`p[rint]`[*msglist*]
`t[ype]`[*msglist*]
Display the specified messages. Messages longer than the number of lines specified by the `crt` variable are paged through the command specified by the `PAGER` variable. The default command is `pg` (see "`mail` Variables").

`q[uit]` Leave `mail` and update the file, folder, or system mailbox you were reading. Messages you have read are stored in `mbox` and unread messages are stored in the system mailbox. Messages you saved in a file are deleted from the system mailbox.

`R[eply]`[*msglist*]
`R[espond]`[*msglist*]
Reply to one or more messages. The reply is sent *only* to the authors of the messages. The subject line is taken from the first message. If you set `record` to a filename, `mail` saves the response at the end of that file (see "`mail` Variables").

r[eply][*message*]
r[espond][*message*]

Send a response to the author and all of the recipients of a message. The subject line is taken from the message. If you set `record` to a filename, `mail` saves the response at the end of that file (see "`mail` Variables").

S[ave][*msglist*]

Save the specified messages in a file whose name is the login name of the author of the first message. See also the `Copy`, `followup`, and `Followup` commands in this section and `outfolder` (in "`mail` Variables").

s[ave][*msglist*][*filename*]

Append related messages to the end of a file. If the file you specify doesn't exist, it is created. If a filename is not specified, the related messages are appended to `mbox`. If a message list is not specified, the current message is used. The related messages are deleted from the system mailbox when you leave `mail` unless you set `keepsave` (see "`mail` Variables" and the `exit` and `quit` commands).

se[t]
se[t] *variable*
se[t] *variable=string*
se[t] *variable=number*

Set a variable to customize `mail`. Variables are either toggled on or off, or they are given a string or number value. To set a toggle variable, type

`set` *variable*

To set a string or number variable, type

`set` *variable=string*

or

`set` *variable=number*

You may specify several variables with one `set` command. Typing `set` prints all defined variables and

their values. See "`mail` Variables" for detailed descriptions of the `mail` variables.

`sh[ell]` The `shell` command invokes an interactive shell (see also the `SHELL` variable in "`mail` Variables").

`si[ze]`[*msglist*]
Print the number of characters in the specified messages.

`so[urce]` *filename*
Read `mail` commands from the given file (usually `.mailrc`) and return to command mode. Useful when you want to fix your `.mailrc` file and need to reread it.

`to[p]`[*msglist*]
Print the top few lines of the specified messages. If you want to change the number of lines printed, set the `toplines` variable (see "`mail` Variables"). The default is 5.

`tou[ch]`[*msglist*]
Touch the specified messages. That is, give the current time and date as the last modified time and date for the specified messages. If any message in `msglist` is not specifically saved in a file, it is placed in the `mbox` when you leave `mail`. (See `exit` and `quit`.)

`T[ype]`[*msglist*]
`P[rint]`[*msglist*]
Display the specified messages, including all header fields. Overrides the `ignore` command.

`t[ype]`[*msglist*]
`p[rint]`[*msglist*]
Display the specified messages. Messages longer than the number of lines specified by the `crt` variable are paged through the command specified by the `PAGER` variable. The default command is `pg` (see "`mail` Variables").

u[ndelete][*msglist*]
: Restore the specified deleted messages. This command restores only those messages deleted in the current mail session. If you set `autoprint`, `mail` displays the last restored message (see "`mail` Variables").

uns[et] *name ...*
: Unset a previously set option. A shell variable cannot be unset.

ve[rsion]
: Print the current version and release date of the `mail` program.

v[isual][*msglist*]
: Edit the given messages with a screen editor. Use the `VISUAL` variable to change the default screen editor (see "`mail` Variables").

w[rite][*msglist*] *filename*
: Write the text of the given messages to the specified file, without saving the header and trailing blank line. Otherwise equivalent to the `save` command.

  Exit from `mail` without changing the system mailbox or the file you were reading. This is useful if you accidentally delete several messages (see also `quit`).

z+
z-
: Scroll the header display forward (`z+`) or backward (`z-`) one screen. The number of headers displayed is set by the `screen` variable (see "`mail` Variables").

9.4 Tilde escapes

Use the following commands only when you're typing a message. Type these tilde escapes at the beginning of a line. See `escape` (in "`mail` Variables") to change the tilde character.

~!*shell-command*
: Execute the shell command and return to `mail`.

~|*shell-command*
: Pipe the body of the message through the given *shell-command*. If successful, the output of the command replaces the message.

~.
: End the message.

~:*mail-command*
~_*mail-command*
: Perform a `mail` command. Valid only when sending a message from within `mail`.

~?
: Print a summary of tilde escapes.

~A
: Insert the autograph string `Sign` into the message (see "`mail` Variables").

~a
: Insert the autograph string `sign` into the message (see "`mail` Variables").

~b *name...*
: Add the *names* to the blind carbon copy (`Bcc:`) list.

~c *name...*
: Add the *names* to the carbon copy (`Cc:`) list.

~d
: Read in the `dead.letter` file. This is a file that you aborted from `mail`. See `DEAD` (in "`mail` Variables").

~e
: Edit the current message with the line editor. See also `EDITOR` (in "`mail` Variables").

~f [*msglist*]
: Append the specified messages to your letter. The messages are inserted unchanged into the current message.

~h
: Edit and append to the header fields `Subject:`, `To:`, `Cc:`, and `Bcc:`.

~i *string*
: Insert the value of the named variable into the text of the message. For example, `~i Sign` is equivalent to `~A`.

~m [*msglist*]
: Append the specified messages to the current message, shifting the new text to the right one tab stop. Valid only when sending a message from within `mail`.

~p
: Display the current message.

~q
: Cancel the current message. Any text you have entered is saved in `dead.letter`. See `DEAD` (in "`mail` Variables").

~r *filename*
~< *filename*
~<! *shell-command*
: Read in the specified file. The exclamation point (`!`) executes the shell command and inserts its output into the message.

~s *string...*
: Set the subject line to *string*.

~t *name...*
: Add the given *names* to the `To:` list.

~v
: Edit the current message with the screen editor. See also `VISUAL` (in "`mail` Variables").

~w *filename*
: Write the current message into the given file, without the header.

~x
: Cancel the current message without saving it in `dead.letter`.

9.5 Environment variables

The following are environment variables, which are not alterable from within `mail`:

`HOME=`*directory*
: The user's home directory.

`MAILRC=`*filename*
: The name of the start-up file. Default is `$HOME/.mailrc`.

9.6 `mail` variables

The following variables are internal `mail` variables. They may be set from the `.mailrc` file or with the `set` command. The `unset` command unsets these variables:

`allnet`
: Default: `noallnet`. Treat all network names with the same login name identically. This causes the `msglist` message specifications to behave similarly. See also the `alternates` command and the `metoo` variable.

`append`
: Default: `noappend`. Append messages to the end of the `mbox` file (instead of adding them to the front) when you leave `mail`.

`askcc`
: Default: `noaskcc`. Prompt for the `Cc:` list at the end of the message.

`asksub`
: Default: `asksub`. Prompt for subject if you don't specify it on the command line with the `-s` flag option.

`autoprint`
: Default: `noautoprint`. Automatically display the next message after `delete` and `undelete`.

`bang`
: Default: `nobang`. Enable the special-casing of exclamation points (`!`) in shell escape command lines, as in `vi`.

`cmd=`*shell-command*
: No default. Set the default for the `pipe` command.

`conv=`*conversion*
: Default: disabled. Convert UUCP addresses to the specified address style. The only valid conversion now is *internet*, which requires a mail delivery program conforming to the RFC822 standard for electronic mail addressing. See also `sendmail` and the `-U` flag option.

`crt=`*number* Default: disabled. Pipe messages having more than *number* lines through the specified `PAGER` variable (`pg` by default).

`DEAD=`*filename*
: Default: `$HOME/dead.letter`. The name of the file in which to save canceled messages.

`debug` Default: `nodebug`. Display debugging information. Messages are not delivered.

`dot` Default: `nodot`. Use a period alone on a line to terminate a message.

`EDITOR=`*shell-command*
: Default: `ed`. The line editor specified by the `edit` or `~e` commands.

`escape=`*c* Default: `~`. Substitute *c* for the `~` escape character.

`folder=`*directory*
: No default. The name of the directory in which to store folder files. If *directory* does not start with a slash (/), it is assumed to be under your home directory. See also `outfolder` below.

`header` Default: `header`. Print header summary when entering `mail`.

`hold` Default: `nohold`. Hold the messages you have read in the system mailbox instead of moving them to the standard `mbox` save file.

`ignore` Default: `noignore`. Ignore RUBOUT and DELETE interrupts when entering messages. Handy for noisy dial-up lines.

`ignoreeof` Default: `noignoreeof`. Ignore *eof* (the A/UX standard distribution defines this as CONTROL-d) during message input. You must terminate with a period (`.`) on a line by itself or by the `~.` command. See also `dot` above.

`keep` Default: disabled. Truncate the system mailbox instead of deleting it when it is empty. Useful if you protect your mailbox.

`keepsave` Default: `nokeepsave`. Keep saved messages in the system mailbox instead of deleting them.

`MBOX=`*filename*
Default: `$HOME/mbox`. The name of the file where messages you have read are saved. The `exit` command overrides this function, as does saving the message explicitly in another file.

`metoo` Default: `nometoo`. If you are included in an alias, `mail` sends you a copy of all messages sent to that alias.

`LISTER=`*shell-command*
Default: `ls`. The command (and flag options) used when listing the contents of the folder directory.

`onehop` Default: disabled. Doesn't change the recipient's address relative to the originating author's machine when responding to a message. This improves efficiency in a network where all machines can send directly to all other machines (i.e., one hop away).

`outfolder` Default: `nooutfolder`. Locates the files recording outgoing messages in the directory specified by the folder variable, unless / is included in the filename. See `folder` above and the `Save`, `Copy`, `followup`, and `Followup` commands.

`page` Default: `nopage`. Used with the `pipe` command to insert a form feed after each message sent through the pipe.

`PAGER=`*shell-command*
: Default: `pg`. Sets the command and flag options to use when paginating output.

`prompt=`*string*
: Default: `?`. Set the `mail` prompt to `string`.

`quiet`
: Default: `noquiet`. Don't print the opening message and version when entering `mail`.

`record=`*filename*
: Default: disabled. Record all outgoing mail in *filename*.

`save`
: Default: `save`. Save canceled messages in `dead.letter`. See `DEAD`.

`screen=`*number*
: No default. Sets the number of lines in a screen of headers for the `headers` command.

`sendmail=`*shell-command*
: Default: `mail`. Alternate command for delivering messages.

`sendwait`
: Default: `nosendwait`. Wait for background mailer to finish before returning.

`SHELL=`*shell-command*
: Default: `sh`. The name of the shell you want to use.

`showto`
: Default: disabled. When displaying the header summary of a message from you, print the recipient's name instead of the author's name.

`sign=`*string*
: No default. The variable inserted into the message when you type `~a`. (See also `~i` in "Tilde Escapes.")

`Sign=`*string*
: No default. The variable inserted into the message when you type `~A`. (See also `~i` in "Tilde Escapes.")

`toplines=`*number*
: Default: 5. The number of lines of header to print with the `top` command.

`VISUAL=`*shell-command*

Default: `vi`. The name of the screen editor to use.

Chapter 3
Using B-NET

Contents

Figures

Chapter 3

Using B-NET

1. Introduction to B-NET

B-NET connects computer systems (via a hardware connection such as an Ethernet) to form a network. A typical network is illustrated in Figure 3-1.

Figure 3-1. A typical network

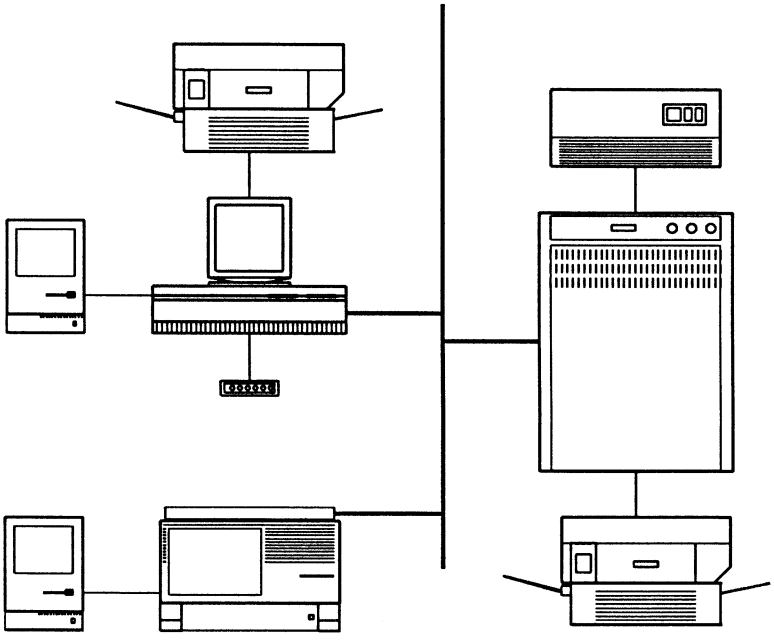


Figure 3-1 shows a network where the resources of five machines and their peripheral devices are available to all users logged into a machine on the network. For example, a user logged in through the

Macintosh™ computer (used as a terminal) in the lower left corner of Figure 3-1 can send a file to be printed on the remote LaserWriter® printer attached to the Macintosh II at the upper left of the figure, or a user on the Macintosh II in the upper left of Figure 3-1 can back up files on the device connected to the VAX at the right of the figure.

In general, **local** refers to the computer on which you initially log in using the standard A/UX login. **Remote** refers to other computers on the network that you access using the B-NET commands. The only exception to this rule is that you can access your local computer "remotely" after you've connected to a remote computer.

The B-NET commands explained in this chapter allow you to log in (using `rlogin` and `telnet`) to different systems, to copy files between different systems (with `rcp` and `ftp`), to use remote devices, and to execute commands (using `remsh`) on different machines. You can also send electronic mail and talk interactively to users who are logged in on different remote systems. Remote machines may or may not be in the same building.

`telnet` and `ftp` allow login and file transfer regardless of the hosts' operating systems and work on wide-area as well as local-area networks. The other B-NET commands, such as `rcp`, `rlogin`, and `remsh`, work only between systems such as A/UX that support BSD-compatible networking software. For the purposes of this chapter, a **wide-area network** is a network that is geographically distributed and may include different types of machines and operating systems; a **local-area network** is geographically contained and includes machines that support the same (or compatible) operating systems. Typically, you may reach a machine on a wide-area network through the phone lines, while machines on a local-area net are connected via Ethernet or another type of cable.

Refer to *A/UX Network System Administration* to learn how to install and maintain B-NET and to get an overview of its internal structure. If you are interested in developing network applications, refer to *A/UX Network Applications Programming*, which explains interprocess communication primitives and network implementation issues at the system interface level. *A/UX Network System Administration* and *A/UX Network Applications Programming* are distributed via APDA.

1.1 B-NET network command overview

Before you can use B-NET, you need a login account (or access to another person's account) on at least two computers on the network. Your system administrator usually sets this up for you on the remote computer.

The machine that you log in to using the standard A/UX `login` procedure is your local system, and any computers that you access via the network commands are remote systems. You'll use the following commands most often when you're working with B-NET. The commands are listed, along with the section in which they're explained.

`rlogin` Connect your terminal on the current system to a remote system. You can then use applications on the remote system as if it were your local machine. (See Section 2.)

`remsh` Execute on the remote machine the command(s) following `remsh`. `remsh` connects to the machine specified, executes the command specified, and terminates when the command terminates. (See Section 3.)

You often must use quoting with this command to control which machine interprets all or part of the command. (See Section 9.)

`rcp` Copy files between machines without remotely logging in. (See Section 4.)

You must sometimes use quoting with this command to control which machine interprets all or part of the command. (See Section 9.)

`telnet` Connect your terminal on the current system to a remote machine. This command is analogous to the `rlogin` command, but implements the `telnet` protocol. (See Section 5.) `telnet` doesn't assume the remote machine uses BSD-compatible software. After connecting, you must enter commands that use whatever syntax the remote system uses.

`ftp` Connect to a specified remote host for copying files between local and remote systems. This command is analogous to the `rcp` command but uses the `ftp` protocol. (See Section 6.)

`ftp` has its own syntax, and it doesn't assume the remote machine uses BSD-compatible software.

`mail` Deliver electronic messages to users across the network. (Note that the A/UX standard distribution aliases the `mailx` utility to `mail`; see `mailx`(1) in the *A/UX Command Reference*.) Although `mail` works very similarly on the network or on a single system, you must use certain addressing conventions to send mail across the network. (See Section 7.)

`talk` Communicate interactively with another user across the local network. Whatever you type at your terminal appears on theirs and vice versa, allowing you to "converse." (See Section 8.)

Most commands that aren't interactive are shown with an ampersand (`&`) as the last character on the command line to indicate that these commands run in the background. Many network commands can be run in the background if you redirect the standard output away from the terminal screen. All commands are terminated with a RETURN unless stated otherwise.

1.2 Checking machine status on B-NET

Two commands check the status of remote machines on the local network:

`ruptime` Give a status line for each machine on the local network. `ruptime` stands for "remote uptime."

`rwho` Give the login names of users who are logged in at each machine on the local network.

The command

```
ruptime
```

(typed at the shell prompt) returns a status line for each machine connected to the network. The output looks something like:

Figure 3-2. Network machine status lines

```
bashful  up       0:39,  1 user
doc      up    4+22:39,  2 users,  load 0.37,  0.27,  0.19
dopey    up    9+22:16,  3 users,  load 0.00,  0.00,  0.13
sleepy   up    6+15:06,  0 users,  load 0.01,  0.02,  0.02
grumpy   up       0:14,  0 users,  load 0.01,  0.06,  0.09
happy    down     0:57
sneezy   up    7+04:41,  3 users,  load 0.48,  0.94,  0.73
```

The first column is the name of each system on the local network. You use this name with B-NET commands to access the machine across the network.

The second column tells you whether or not the system is up. The third column indicates how long the system has been up or down in days, hours, and minutes, respectively. (For example, `4+22:39` means 4 days, 22 hours, and 30 minutes.) The fourth column represents how many users are currently logged in to the system, and the fifth, sixth, and seventh columns show the load average (the average number of processes on that system) over the past 1, 5, and 15 minutes, respectively.

You may specify the following flag options on the command line when you invoke `ruptime`:

`-a` Count all logged-in users, no matter how long they've been idle. Often, users who have been idle for an hour or more are not listed as logged in on a particular machine unless you specify this flag.

`-l` Sort the listing by load average. Without this flag, the listing is sorted by host name.

`-t` Sort the listing by which systems have been up the longest. Without this flag, the listing is sorted by host name.

`-u` Sort the listing by the number of users on each remote system. Without this flag, the listing is sorted by host name.

You may also use the `ruptime` command with `grep` to obtain a listing for a single machine, or for all machines that are up, and so on. For example, to see the listing for a machine named `doc`, use the command

```
ruptime | grep doc
```

This returns the status line

```
doc  up  4+22:39, 2 users, load 0.37, 0.27, 0.19
```

The command

```
rwho
```

lists all users logged in on any machine on the local network and the name of the machine they're logged in to. If a user hasn't used the system for a minute or more, `rwho` reports this as idle time. If a user hasn't used the system for an hour or more, the user is omitted from the output of `rwho`, unless you've specified the `-a` flag.

You may also use the `rwho` command with `grep` to see if a particular user is logged in or who is using a particular remote system. For example, the command

```
rwho | grep doc
```

returns the login names of every user on the remote system `doc`.

2. Logging in to a remote machine

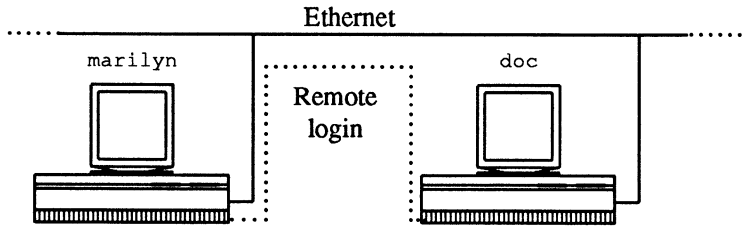
You use the `rlogin` (remote login) command to log in to a remote machine. The `rlogin` command creates a virtual terminal on the remote system you specify on the command line. This functions like a direct connection. The `rlogin` command has the format

```
rlogin rhost
```

where *rhost* (for "remote host") is the name of the remote computer. Figure 3-3 shows a remote login by a user on the machine named `marilyn` to the machine named `doc`. Assuming that the user has the same login name on both machines, he or she would use the command

```
rlogin doc
```

Figure 3-3. Logging in to a remote machine



If you use `rlogin` to gain access to the remote system `doc`, you'll see

```
$ rlogin doc
$
```

where the first line is the command you've given on the local computer and the second is the shell prompt on the remote computer.

The remote system reads your remote login file before displaying a shell prompt. When the shell prompt appears, you are connected to the remote computer and are working in its shell.

If you don't get the remote shell prompt, it's possible you don't have an account on the remote machine. In this case, you should see your system administrator about getting the appropriate account.

The command

```
hostname
```

returns the name of your current system, in this example, `doc`.

On some systems you may omit the `rlogin` name from the command line. In this case, the command

rhost

logs you in to the remote system.

Note: See the section "Abbreviated Command Formats" in this chapter for more information.

2.1 `rlogin` flag options

You may specify the following options on the `rlogin` command line:

`-e` *character* Change the default escape character. This option is described in the section "Changing the Escape Character" later in this chapter.

`-l` *username* Specify a different login name on the remote machine.

If your login name is not the same on the local and remote systems, or if you have access to another user's account, use the `-l` flag option to the `rlogin` command

`rlogin` *rhost* `-l` *username*

where *username* is your login name on the remote system. You should also create a `.rhosts` file in your remote login directory to establish equivalence between your two login names. See the section "Network Permissions" in this chapter for more information. On some systems, the remote machine can discover that the two accounts are equivalent even if the user names differ.

2.2 Using the remote system

When you're connected to a remote A/UX system via `rlogin`, everything you do is executed on the remote machine. If you need to access a file on the local system, you can suspend the remote login temporarily. See "Suspending `rlogin`."

For example, to create a directory `Test` on the remote machine and move into it, type

```
mkdir Test
cd Test
```

Then to create a file `jelly` and put some text into it, type

```
vi jelly
...
add some text
...
:wq
```

After you write and quit the file, to print it on your screen, type

```
cat jelly
```

To direct the output of the `cat` command into a new file, type

```
cat jelly > jam
```

The output is directed into `jam` on `doc`, the *remote* computer. To make a copy of `jelly` on your *local* machine, `marilyn`, use the `rcp` network command. (See "Remote File Copy: `rcp`.")

2.3 Suspending `rlogin`

If you are using the C shell or the Korn shell on the system from which you used the `rlogin` command (the "local" system), you can type the escape character followed by a *susp* character (*susp* is set to CONTROL-z in the A/UX standard distribution; see `stty`(1) in *A/UX Command Reference* and *A/UX User Interface* for more information).

In this case, you would type

~CONTROL-z

followed by RETURN to suspend the `rlogin` session and temporarily return to the local system. You can then resume working on the remote system by typing

```
fg
```

See "Job Control" in the chapters on the C shell or Korn shell in *A/UX User Interface*.

If you are using the Bourne shell, you may want to invoke `rlogin` from a shell layer. In this case, you can use the same command as above (~CONTROL-z) to temporarily return to the `shl` prompt on the local system. You can then resume working on the remote system by "resuming" the shell layer. See "Shell Layering" in *A/UX User Interface*.

2.4 Exiting: terminating `rlogin`

When you want to disconnect from the remote system, first press RETURN. Then type

```
~.
```

(tilde period) followed by another RETURN.

The sequence

escape-character.

(where the *escape-character* is the first character on a new line) disconnects you from the remote system and returns you to your local account. This sequence also cancels the `rlogin` attempt and returns you to your local account if you press it before it has logged you in to the remote account. You may type the tilde-period sequence at any time on the remote system (even if you are running an application there), and it will return you to the local computer.

On most systems, the commands `exit` and `logout` also terminate `rlogin`.

> *Note:* When you're remotely logged in to a system, the shell doesn't continue processing your remote background jobs when you terminate `rlogin`. If you want your background jobs to continue processing when you disconnect from the system, use `nohup` when you start the jobs.

For example, use

`nohup` *command* `&`

rather than

command `&`

2.5 Changing the escape character

When you're working on a remote machine, your local system processes your input by directing it across the network to the remote system. The tilde character (~) at the beginning of a line introduces an **escape sequence,** which tells B-NET that the character or control sequence that follows should be interpreted on the local machine, not the remote one. Normally, you'd use escape sequences only when you're suspending `rlogin`, using shell layering, or disconnecting from the remote system.

The tilde character is called the **escape character.** You may change the escape character from a tilde to any other printable character using the `-e`

flag option on the `rlogin` command line.

For example, to log in to the remote system `doc` and change the escape character to `]`, you'd type

```
rlogin doc -e]
```

In this case, the escape sequence to terminate `rlogin` would be

```
].
```

The sequence to suspend `rlogin` in this case would be

```
]CONTROL-z
```

The new escape character lasts only for the current `rlogin`; the next time you remotely log in, the escape character will be tilde again.

2.6 Troubleshooting

If you're using the `rlogin` command for the first time and you see

```
Login incorrect
```

check with your system administrator about your login name and password on the remote account. If you have different login names on the local and remote systems, use

```
rlogin -l login-name
```

(where *login-name* is your name on the remote system). You must have previously configured a `$HOME/.rhosts` file on one or both systems; see "Network Permissions."

If you get the message

```
Connection timed out
```

the remote machine is either down or very busy. If the remote machine is down, you'll have to wait until it's up again, or try another machine. The `ruptime` command doesn't report that a machine is down unless there has been no response for five minutes.

If you get the message

```
rhost: unknown host
```

the local machine doesn't know the remote system's address or you entered the remote hostname incorrectly on the command line, so `rlogin`

cannot connect to it. Your system administrator needs to modify the local system files to correct this, or you need to make sure you entered the correct *rhost* name.

The message

```
rhost: host name for your address unknown
```

means the remote machine does not know your local system's name and address. Your system administrator needs to correct this by modifying system files on the remote machine.

When you've logged in to the remote computer via `rlogin`, the shell environment of your remote account may be different from your local environment. This can cause some unexpected responses, including `vi`'s treating your terminal as a dumb terminal.

If `/etc/termcap` and other required files are not resident on the remote system, the system administrator has to correct this situation.

If you think the problem is with your personal files rather than the file system, terminate `rlogin`. You can then `rcp` your local login file. For example, if you are using the Bourne or Korn shells, copy your `.profile` file to the remote system using the command

```
rcp .profile 'rhost:$HOME' &
```

The single quotes enclosing the second argument are essential. After you have done this, try `rlogin` again.

3. Remote shell: `remsh`

You can use the `remsh` command to execute a command on a remote UNIX machine while you stay on your local machine.

The `remsh` command has the format

```
remsh rhost command
```

where *rhost* is the name of the remote computer, and *command* is a single A/UX command.

In Figure 3-4, a local `cat` process pipes an input file to the `remsh` command, which in turn conveys the file to `doc` and executes the remote `lp` process. The command to achieve this is:

```
cat a | remsh doc lp
```

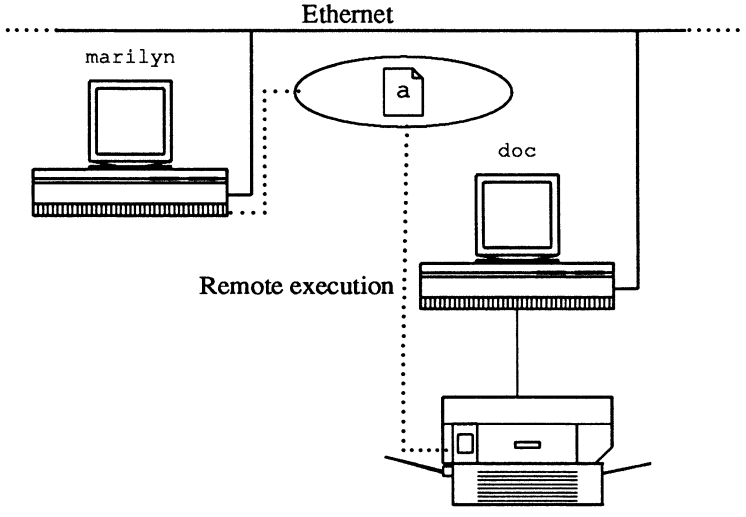
Note that the `lp` command is executed on the remote CPU `doc`. It is important to understand the two separate execution environments; for example, the command

```
remsh doc cat a | lp
```

pipes the remote file `a` from `doc` to the local printer on `marilyn`.

Note: If you are familiar with the A/UX shell quoting mechanisms, you can simply follow the rule that the local shell interprets all metacharacters that are not quoted. You can control the interpretation of shell variables using double quotes (interpretation on the local system) or single quotes (interpretation on the remote system). If you are not familiar with the shell quoting mechanisms, see the section "Local and Remote Shell Topics" in this chapter and *A/UX User Interface*.

Figure 3-4. Printing a local file on a remote printer



You must have a login account (or access to another user's account) on the remote system (see the section on "Network Permissions" in this chapter).

You cannot use the `remsh` command successfully to run an interactive program like `vi`, which sets special tty modes. For that, you must use `rlogin` or `telnet`.

3.1 `remsh` flag options

You can specify the following flag options on the `remsh` command line:

`-l` *username* Specify a different login name on the remote system.

`-n` Redirect standard input from `/dev/null`.

If your login names differ across systems, or if you're using another user's remote account, the `-l` flag option is required:

```
remsh rhost -l username command
```

Note: If the login names on your local and remote accounts are different, you must supply the remote login name on the `remsh` command line using the `-l` flag option. You must also configure `$HOME/.rhosts` on the remote computer (see "Network Permissions"). This section assumes that the login names are identical across the systems.

If you're using the `remsh` command in the background, the `-n` flag option redirects the standard input of the remote command from `/dev/null` (instead of your terminal) as follows:

```
remsh rhost -n command &
```

In most cases, the only noticeable effect of the `-n` flag option is that the network is less fussy about what it sees. If a remote command tries to read from (remote) standard input while it's executing, it could "hang" indefinitely without terminating. If this happens to a remote background command, you can try using the `-n` flag option on the `remsh` command line the next time you use that command.

3.2 Running remote processes

When you use the `remsh` command with shell metacharacters such as the pipe (`|`) and redirection characters (>, >>, <), you can be flexible in

choosing where a command runs, where the output is directed, and which peripheral devices are used. In the simplest case, you can run a remote process, such as `date`, on the remote machine `doc` with the command

```
remsh doc date
```

This gives a response such as

```
Fri Aug 9 13:25:19 PDT 1985
```

You can run a process on a remote file. For example, to send a remote file (`jelly`) to `doc`'s line printer, you'd use the command

```
remsh doc lp jelly &
```

If you want to use a file on your local machine in a process on the remote machine, use the pipe character (`|`) with a local `cat` command to convey the file across the network as input to the remote process. For example, to run an `nroff` process on the remote machine `doc` using a local file, `feet`, and display the results on your terminal screen, you'd use the command

```
cat feet | remsh doc nroff
```

Note that this process cannot run in the background unless you redirect the output.

3.3 Using remote devices

Many networks have printers attached to a few systems, which may be accessed by all users across the network. The following examples show how to send a file to a printer attached to a remote system named `doc`. Note that pipes (`|`) are interpreted by the local machine unless they're quoted.

A local `cat` command

```
cat feet | remsh doc lp &
```

conveys a file across the network to the remote line printer.

If you need to `nroff` the file before printing it, you can format it on the local or remote system. The command

```
nroff feet | remsh doc lp &
```

formats your file locally and prints it remotely, while the command

```
cat feet | remsh doc nroff \| lp &
```

both formats and prints your (local) file on the remote system. Because the second pipe character in the command is "quoted" with a backslash (\), that character (and the command that follows it) is interpreted on the remote machine. Thus, the remote line printer is used.

If the second pipe character is not quoted

```
cat feet | remsh doc nroff | lp &
```

it's interpreted on the local machine. In that case, the `nroff` process executes on the remote machine, but the output is sent to a local spooler.

The `remsh` command also allows you to run pipelined processes on a number of remote machines, distributing the CPU load. For example,

```
cat feet | remsh doc nroff | remsh bashful lp &
```

In a pipeline, you can quote the pipes between commands or not, with no noticeably different effect. In the command line above, the second pipe is not quoted, so it is interpreted on the local system. Thus, the output of the remote `nroff` command is returned to the local system where it is again sent across the network to another system. A similar command

```
cat feet | remsh doc nroff \| remsh bashful lp &
```

uses a backslash to quote the second pipe character, causing it to be interpreted on the remote system. Thus, the output of the `nroff` command is sent to another remote machine without first returning across the network to the local system. See "Quoting Mechanisms" for more information.

3.4 Terminating `remsh`

The `remsh` command normally terminates when the remote command does. If you need to terminate `remsh` abnormally, you may send an *interrupt* (set to CONTROL-c in the A/UX standard distribution) or a *quit* signal (CONTROL-\ in the A/UX standard distribution). These will propagate across the network to terminate the remote command.

You can usually terminate a remote background command using the `kill` command with the process ID on the local machine. If the `kill` command works only on the local `remsh` command (allowing the remote

process to continue running), try using the `-9` flag option to the `kill` command

```
kill -9 process-ID
```

or using the `-n` flag option on the `remsh` command line.

3.5 Troubleshooting

The remote system must be able to determine that your local and remote accounts are equivalent, or it won't allow you to use `remsh`. For further information, see the section on "Network Permissions" in this chapter.

The most common mistake on the `remsh` command line is incorrect quoting of command arguments. See "Quoting Mechanisms" and "`remsh` Command Line Considerations" later in this chapter and the appropriate chapter in *A/UX User Interface* for details.

If you get the message

```
No match
```

the local shell may be interpreting shell metacharacters. Try enclosing the entire command argument in single quotes.

You may see the message

```
Command not found
```

if you use a specialized command (such as a system shell script) that exists on one system and not others. This can also occur if the file system is set up differently on the remote system, and the path to the command is interpreted locally. In this case, try quoting the argument.

If the path to your login directory is different on the remote system and you reference your HOME variable (in the Bourne shell or the Korn shell) or use the tilde shorthand (in the C shell) without quoting

```
remsh rhost ls -l ~
```

you may get the message

```
No such file or directory.
```

In this case, try enclosing the whole command in single quotes

```
remsh rhost 'ls -l ~'
```

or

```
remsh rhost 'ls -l ~username'
```

Other errors can occur when the permissions set by the system administrator are not as they should be. See the section on "Network Permissions" in this chapter for more information.

If you omit the command from `remsh`, it simply connects you to the remote system using the `rlogin` process. The command

```
remsh doc
```

or

```
remsh doc -l username
```

logs you in on the remote host, `doc`.

4. Remote file copy: `rcp`

The `rcp` command uses the format

```
rcp source-file dest-file
```

where *source-file* is the file being copied and *dest-file* is the destination file (into which a copy of the source file is placed). The format is similar to that of the A/UX `cp` command.

The first argument to `rcp` is always the source file and the second argument is always the destination. Either or both of these files may be remote.

The `rcp` command assumes a file is remote if it's specified in the form

rhost:*path*

where *path* is the path to the file (or directory) on the remote machine. The *path* is interpreted relative to your home directory on the remote machine, unless you enter the absolute pathname (that is, the path relative to the root directory).

If both arguments are in the remote format

```
rcp rhost:path rhost:path
```

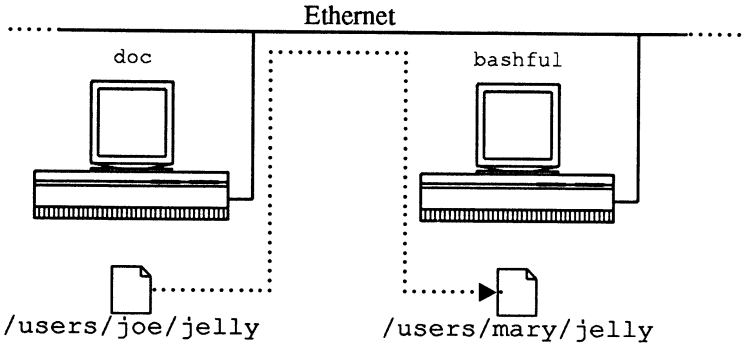
the `rcp` command copies a file from one remote machine to another.

For example, as shown in Figure 3-5,

```
rcp doc:/users/joe/jelly bashful:/users/mary/jelly
```

copies the `jelly` file in `joe`'s directory on `doc` to `mary`'s directory on `bashful`.

Figure 3-5. Copying a file across the network



4.1 `rcp` flag options

The name of the remote system (*rhost*) may also take the form

rhost.*username*

where *username* is the name of another user (or another login name of yours) on *rhost*. (See "Network Permissions" for more information about permissions on the network.)

In this form, the remote argument(s) to `rcp` is in the format

rhost.*username*:*path*

The `-r` flag option to `rcp` copies whole directory trees. The command

`rcp -r` *rhost*`:`*path path*

copies all the files and subdirectories contained in *rhost*:*path* into a new directory beneath *path*. See "Copying Directories" in this chapter for more information.

4.2 Copying files

The command

```
rcp rhost:path dest-file &
```

(where *path* terminates in a filename) copies and renames a file from your remote account to your current local directory.

For example, to copy the remote file `gateau` from the system `doc` to your current directory and rename it `cake`, you'd type the command

```
rcp doc:gateau cake &
```

`gateau` is copied from your login directory on `doc`, and renamed `cake` in your current directory on the local system.

The command

```
rcp gateau doc:cake &
```

copies `gateau` from your current directory and renames it `cake` in your remote login directory on `doc`.

4.3 Shorthand notations

When *rhost*:*path* is the second (*destination*) argument, it may end in an existing directory name. In that case, the local file is copied into that directory, and retains its local name.

The *destination* argument may also be replaced with a period (`.`) if you don't need to rename the file. When used as the *destination* argument, the period character means "current directory." The command

```
rcp doc:cake . &
```

is the equivalent of

```
rcp doc:cake cake &
```

If you are using the C shell or the Korn shell, you may also use the tilde (~) notation to indicate your login directory:

```
rcp cake doc:~ &
```

When you use this notation, the file is copied from your current directory to your login directory on the remote machine.

4.4 Copying directories

The `-r` flag option to `rcp` copies a whole directory (including any subdirectories) across the network. Both the source and destination arguments must end in a directory name or shorthand notation (see the preceding section for a description of shorthand notations). The command

```
rcp -r rhost:path path &
```

creates a new directory below the local *path* and copies the contents of a remote directory into it. The name of the new local directory is the same as its remote name. For example,

```
rcp -r doc:Test . &
```

creates a `Test` directory below your current (local) directory. The local `Test` directory contains copies of every file and subdirectory contained in the remote `Test` directory on `doc`.

If you specify a name following the *destination* directory

```
rcp -r rhost:path path/newname &
```

`rcp` creates a new directory by that name, and copies the remote directory into it.

> *Note:* A new directory is created by `rcp -r`; you should not create an empty directory expecting to copy the remote directory's contents.

The `rcp -r` command also works between two remote computers or from a local source to a remote machine.

4.5 Terminating `rcp`

The `rcp` command normally terminates when the file copy is complete. As with the `remsh` command, you may send an *interrupt* (set to CONTROL-c in the A/UX standard distribution) or a *quit* signal (CONTROL-\ in the A/UX standard distribution) across the network to terminate `rcp`. In this case, you'll usually have an incomplete copy at your destination directory.

You may also use the `kill` command with the process ID on the local machine to stop an `rcp` process.

4.6 Troubleshooting

The `rcp` command does not prompt for passwords, so the remote system must be able to determine that your local and remote accounts are equivalent. Otherwise, it won't allow you to use `rcp`. See the section on "Network Permissions" in this chapter.

In some cases, more than one user on a machine may share the same user ID. If that's the case on either system, the *path* argument may be interpreted relative to a directory that's not your home directory. If the command

```
rcp rhost:source-file dest-file
```

prints a message such as

```
rcp: source.dir: No such file or directory
```

try the same command again, specifying your remote login name with the format

```
rcp rhost:~username/source-file dest-file
```

where *username* is your login name on the remote machine. If this command solves the problem, you should specify your login directory in every `rcp` command.

If the absolute path to your login directory differs across machines, you should quote the `$HOME` variable reference or the tilde shorthand for a relative path on a remote machine (using a backslash, double quotes, or single quotes) to ensure that the tilde is interpreted on the remote machine.

You may find out the absolute path to your home directory on *rhost* using the command

```
remsh rhost grep username /etc/passwd
```

If the absolute path to your home directory is identical on both machines, the tilde does not need to be quoted.

If you have repeated problems, check with your system administrator.

When you're using the `-r` flag option to `rcp`, remember that it creates a directory *below* your current directory. It's a common mistake to create a destination directory and move into it before using the `rcp -r` command.

5. Virtual terminal: `telnet`

The `telnet` utility is a user interface to the Transmission Control Protocol/Internet Protocol (TCP/IP) standard TELNET Protocol. (Most computers implementing the TCP/IP protocols recognize FTP and TELNET). The `telnet` utility is used to establish a connection with another host. It sets up a remote login connection similar to `rlogin`, simulating a terminal with a physical connection to the remote host.

The `telnet` utility is slower than `rlogin` but is useful for connecting to machines that run an operating system different from A/UX, or are not located on the same local area network as your local A/UX host.

There are two `telnet` modes: command mode and input mode. In command mode, you use your normal editing conventions to enter `telnet` commands. In input mode, the text you type is forwarded to the remote host.

5.1 Command mode

You use all `telnet` commands when you're in command mode.

Once you establish a connection with the remote system, `telnet` enters input mode, and the text you type is sent to the remote host. When you have logged in to the remote system, you see the regular prompt for that machine and you can carry out a normal login session.

To issue local `telnet` commands while you're logged in to the remote system, enter `telnet` command mode by typing the escape character (CONTROL-`]`), followed by RETURN.

5.1.1 Logging in

The command to invoke `telnet` is

```
telnet rhost
```

(where *rhost* is the network name of the computer you want to access).

You may also invoke `telnet` without specifying *rhost* on the command line, in which case `telnet` enters command mode and displays the `telnet` prompt.

You may then request access to a particular computer with the `open` command:

`telnet`	invoke `telnet` in command mode
`telnet>`	(the `telnet` prompt follows)
`open` *rhost*	connect to remote system

5.1.2 The escape character

The default `telnet` escape character is CONTROL-`]` (hold down the CONTROL key and press the right bracket key). You may change the escape character with the `escape` command, using a caret (`^`) to represent a CONTROL sequence.

For example, to change the escape character to CONTROL-x, you'd type

```
CONTROL-]
```

then, to the `telnet>` prompt, you'd type

```
escape ^X
```

After pressing RETURN, you'd see

```
Escape character is ^X
```

5.1.3 Current status

After you have logged in to a remote host with `telnet`, you can learn the current `telnet` status by pressing CONTROL-`]` to enter command mode, and then by typing

```
status
```

This should return the message

```
Connected to rhost
Escape character is ^]
```

telling you that you're connected and what the escape character is.

5.1.4 Logging out

To close your current `telnet` connection but remain in `telnet` command mode, first press CONTROL-`]` to enter command mode, then use the command

```
close
```

You may then open a connection to a different host computer using the `open` command. Alternatively, you may close the current session and quit the `telnet` program with the command

```
quit
```

(to the `telnet>` prompt, after you've entered command mode). Then you return to the local system prompt.

5.1.5 Troubleshooting

The error messages described for the `rlogin` command also apply to `telnet`. Your local system administrator can probably correct the situation if you see the message

```
rhost:unknown host
```

(The local machine does not know the remote machine's address.)

In the following cases, you may need to contact the administrator of the remote system.

```
Connection refused.
```

This means that the remote system detects a permissions violation or that the remote system is up but the remote server is not running.

```
Connection timed out.
```

The remote system is either down or very busy.

```
rhost: host name for your address unknown.
```

The remote machine does not know the local machine's address.

5.2 Commands

The following `telnet` commands are available to you if you precede them with the `telnet` escape character (by default, CONTROL-]). You may abbreviate each command, as long as you enter enough of the command to identify it uniquely:

`close`	Close this session and return to command mode.
`quit`	Close any session that is currently open and exit `telnet`.
`escape [`*char*`]`	Set the `telnet` escape character to the specified character. Control characters are specified as ^

	followed by a single letter (e.g., CONTROL-x is `^X`).
`status`	Show the current status of `telnet`.
`options`	Toggle viewing of `telnet` options. When this mode is enabled, all `telnet` options are displayed. Options sent by `telnet` are displayed as sent, and options received from the `telnet` server are displayed as received.
`crmod`	Toggle carriage return mode. When this mode is enabled, any RETURN characters received from the remote host are mapped into a RETURN and a line feed. This mode does not affect those characters you type, only those received. This mode is required for some hosts that like to ask the user to do local echoing.
`?` [*command*]	Get help. When you don't include any arguments, `telnet` prints a help summary.

6. File transfer protocol: `ftp`

The `ftp` utility is a user interface to the Transmission Control Protocol/Internet Protocol (TCP/IP) standard File Transfer Protocol (FTP). (Most computers implementing the TCP/IP protocols recognize FTP and TELNET.) The `ftp` utility was developed to allow the reliable transfer of files to and from a remote network site, regardless of the hosts' operating systems (in contrast to `rcp`, which requires the hosts to run A/UX or some other BSD-compatible networking software).

Unlike the `rcp` command, `ftp` is interactive, it keeps track of your current directory on both local and remote computers, and it interprets pathnames relative to your current directory on either machine. It does not interpret pathnames relative to your login directory, unless that is your current directory.

6.1 Using ftp

6.1.1 Logging in

The section "ftp Commands" contains a list of ftp commands for connecting to and disconnecting from remote systems.

Invoke ftp with the command

```
ftp rhost
```

The ftp program immediately attempts to establish a connection to an ftp server on *rhost*, and prompts for your login name and password on the remote machine. This prints something like:

```
Connected to rhost.
220 rhost FTP server ready.
Name (rhost:carl):
331 Password required for carl.
Password (rhost:carl):
230 User carl logged in.
ftp>
```

(where carl is your login name on the local host).

If you have a different login name on the remote system, enter the remote login name in response to the prompt

```
Name (rhost:carl):
```

Otherwise, just press the RETURN key. If the remote account has a password, you will be asked to supply it.

You may also invoke ftp without specifying the remote host on the command line. In this case, ftp enters command mode and displays the ftp prompt:

```
ftp>
```

You may request access to a particular computer by using the open command:

```
open rhost
```

ftp responds similarly to the sequence displayed when you type the remote machine's name on the command line.

Once you have successfully logged in, you see the prompt

```
ftp>
```

and can begin entering the `ftp` commands to list directories, remove files, copy files, and so on.

6.1.2 Transferring files

Files may be transferred across the network using the `get` and `put` commands. The `get` command copies a file from the remote system to your local system, while `put` copies in the reverse direction. Both commands take filename arguments that may be either complete pathnames, or pathnames relative to your *current* directory.

The format of the `get` and `put` commands is

`get` *remote-source local-dest*

`put` *local-source remote-dest*

If you specify the filename as – (the minus sign), `ftp` uses the standard input for reading or standard output for writing.

See the sections "`ftp` Commands" and "File Transfer Parameters" for a list of commands to use in transferring files.

Before you use these commands, however, it is often desirable to set certain environmental conditions in `ftp`.

6.1.3 Flag options

You may specify the following flag options on the command line when you invoke `ftp`.

`-v` Force `ftp` to show all responses from the remote server, as well as report on data transfer statistics.

`-n` Restrain `ftp` from autologin so that you are connected to a host without being prompted to log in (see `user` in "`ftp` Commands" below).

`-i` Turn off interactive prompting during multiple file transfers.

`-d` Enable socket level debugging. Helpful only to programmers.

`-g` Disable filename globbing; that is, treat all filenames and pathnames literally, without interpreting metacharacters as such.

6.1.4 Setting conditions

There are several `ftp` toggle commands that turn a condition on or off, depending on the previous state of the condition. See the section "Toggle Commands" for a complete list. These commands can only be entered from the `ftp>` prompt.

For example, the command

```
bell
```

causes a bell to sound or not to sound after each file transfer, depending on the previous state.

The `verbose` command is on by default. The command

```
verbose
```

turns explanatory messages off.

To find the current state of all toggle conditions and other status information, use the command

```
status
```

6.1.5 Using directories

To learn what files are in the current directory on the remote computer, use the `dir` command. `ftp` responds with a listing identical to that given by the A/UX command, `ls -al`:

```
drwxr-xr-x 5   vz   16655   Jul   54   10:11   forms
-rw-rw-r-- 1   vz    7389   Aug   23   13:34   program
-rw-rw-r-- 1   vz      57   Sep   27   12:03   new.01
```

The commands to change your current directory on the local and remote computers are `lcd` and `cd`, respectively.

In the example directory listing above, the command

```
cd forms
```

changes the current remote directory to the `forms` directory.

See the section "`ftp` Commands" for a list of `ftp` commands for using directories.

6.1.6 Logging out

If you want to close a session with a particular computer, but remain in `ftp`, use the command

```
close
```

To close the current connection and exit `ftp` altogether, use

```
quit
```

6.2 `ftp` commands

The following lists of commands are recognized by `ftp` and are entered at the prompt

```
ftp>
```

Command arguments that have embedded spaces may be quoted with double quotes (`""`).

`!` [*command* [*args*]]
: Invoke an interactive shell on the local machine. If there are arguments, the first is taken to be a command to execute directly, with the rest of the arguments as the command's arguments.

`$` *macro-name* [*args*]
: Execute the macro *macro-name* that was defined with the `macdef` command. Arguments are passed to the macro without metacharacter expansion.

`account` [*passwd*]
: Supply a supplemental password required by a remote system for access to resources once a login has been completed successfully. If no argument is included, the user will be prompted for an account password in a non-echoing input mode.

`append` *local-file* [*remote-file*]
: Append a local file to a file on the remote machine. If *remote-file* is left unspecified, the local filename is used in naming the remote file, after being altered by any `ntrans` or `nmap` setting. File transfer uses the current settings for `type`, `form` (format), `mode`, and `struct` (structure).

`bye`
: Terminate the FTP session with the remote server and exit `ftp`. An

eof (CONTROL-d in the A/UX standard distribution) will also terminate the session and exit.

`cd` *remote-directory*
Change the working directory on the remote machine to *remote-directory*.

`cdup`
Change the remote machine working directory to the parent of the current remote machine working directory.

`close`
Terminate the FTP session with the remote server, and return to the command interpreter. Any defined macros are erased.

`delete` *remote-file*
Delete the file *remote-file* on the remote machine.

`dir` [*remote-directory*] [*local-file*]
Print a listing of the directory contents in the directory, *remote-directory*, and, optionally, place the output in *local-file*. If no directory is specified, the current working directory on the remote machine is used. If no local file is specified, or *local-file* is -, output comes to the terminal.

`disconnect`
A synonym for `close`.

`get` *remote-file* [*local-file*]
Retrieve the *remote-file* and store it on the local machine. If the local filename is not specified, it is given the same name it has on the remote machine, subject to alteration by the current `case`, `ntrans`, and `nmap` settings. The current settings for `type`, `form`, `mode`, and `struct` (structure) are used while transferring the file.

`help` [*command*]
Print an informative message about the meaning of *command*. If no argument is given, `ftp` prints a list of the known commands.

`lcd` [*directory*]
Change the working directory on the local machine. If no *directory* is specified, the user's home directory is used.

`ls` [*remote-directory*] [*local-file*]
Print an abbreviated listing of the contents of a directory on the remote machine. If *remote-directory* is left unspecified, the current working directory is used. If no local file is specified, or if *local-file* is `-`, the output is sent to the terminal.

`macdef` *macro-name*
Define a macro. Subsequent lines are stored as the macro *macro-name*; a null line (consecutive newline characters in a file or carriage returns from the terminal) terminates macro input mode. There is a limit of 16 macros and 4096 total characters in all defined macros. Macros remain defined until a `close` command is executed. The macro processor interprets `$` and `\` as special characters. A `$` followed by a number (or numbers) is replaced by the corresponding argument on the macro invocation command line. A `$` followed by an `i` signals to the macro processor that the executing macro is to be looped. On the first pass, `$i` is replaced by the first argument on the macro invocation command line, on the second pass, it is replaced by the second argument, and so on. A `\` followed by any character is replaced by that character. Use the `\` to prevent special treatment of the `$`.

`mdelete` [*remote-files*]
Delete the *remote-files* on the remote machine.

`mdir` *remote-files local-file*
Like `dir`, except multiple remote files may be specified. If interactive prompting is on, `ftp` will prompt the user to verify that the last argument is indeed the target local file for receiving `mdir` output.

`mget` *remote-files*
Expand the *remote-files* on the remote machine and do a `get` for each filename thus produced. See `glob` for details on the filename expansion. Resulting filenames will then be processed according to `case`, `ntrans`, and `nmap` settings. Files are transferred into the local working directory, which can be changed with

`lcd` *directory*

New local directories can be created with

`! mkdir` *directory*

`mkdir` *directory-name*
Make a directory on the remote machine.

`mls` *remote-files local-file*
Like `ls`, except multiple remote files may be specified. If interactive prompting is on, `ftp` will prompt the user to verify that the last argument is indeed the target local file for receiving `mls` output.

`mput` *local-files*
Interpret metacharacters in the filenames in the list of local files given as arguments and do a `put` for each file in the resulting list. See `glob` for details of filename expansion. Resulting filenames will then be processed according to `ntrans` and `nmap` settings.

`nmap` [*inpattern outpattern*]
Set or unset the filename mapping mechanism. If no arguments are specified, the filename mapping mechanism is unset. If arguments are specified, remote filenames are mapped during `mput` commands and `put` commands issued without a specified remote target filename. If arguments are specified, local filenames are mapped during `mget` commands and `get` commands issued without a specified local target filename.

This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. The mapping follows the pattern set by *inpattern* and *outpattern*. *inpattern* is a template for incoming filenames (which may have already been processed according to the `ntrans` and `case` settings). Variable templating is accomplished by including the sequences

```
$1, $2, ..., $9
```

in *inpattern*. Use the \ character to prevent special treatment of the `$`, `[`, `]`, and `,` characters. All other characters are treated literally, and are used to determine the `nmap` *inpattern* variable values. For example, if the *inpattern* is

```
$1.$2
```

and the remote filename is `mydata.data`, `$1` would have the value `mydata`, and `$2` would have the value `data`. The *outpattern* determines the resulting mapped filename. The sequences

```
$1, $2,..., $9
```

are replaced by any value resulting from the *inpattern* template. The sequence `$0` is replaced by the original filename. Additionally, the sequence `[`*seq1*`,`*seq2*`]` is replaced by *seq1* if *seq1* is not a null string; otherwise, it is replaced by *seq2*.

For example, for input filenames `myfile.data` and `myfile.data.old`, the command

```
nmap $1.$2.$3 \[$1\,$2\].\[$2\,file\]
```

would yield the output filename `myfile.data`. For the input filename `myfile`, the same command would yield the output filename `myfile.file`, and for the input filename `.myfile`, the same command would yield `myfile.myfile`.

Spaces may be included in *outpattern*, as in the example:

```
nmap $1 |sed "s/  *$//" > $1
```

`ntrans` [*inchars* [*outchars*]]

Set or unset the filename character translation mechanism. If no arguments are specified, the filename character translation mechanism is unset. If arguments are specified, characters in remote filenames are translated during `mput` commands and `put` commands issued without a specified remote target filename. If arguments are specified, characters in local filenames are translated during `mget` commands and `get` commands issued without a specified local target filename.

This command is useful when connecting to a non-UNIX remote computer with different file naming conventions or practices. Characters in a filename matching a character in *inchars* are replaced with the corresponding character in *outchars*. If the character's position in *inchars* is longer than the length of *outchars*, the character is deleted from the filename.

`open` *host* [*port*]

Establish a connection to the specified *host* FTP server. An optional port number may be supplied, in which case, `ftp` will attempt to contact an FTP server at that port. If autologin is enabled (default), `ftp` will also attempt to log the user in to the FTP server (see "`$HOME/.netrc`").

`proxy` *ftp-command*

Execute an `ftp` command on a secondary control connection. This command allows simultaneous connection to two remote FTP servers for transferring files between the two servers. The first `proxy` command should be an `open`, to establish the secondary control connection. Enter the command

```
proxy ?
```

to see other `ftp` commands executable on the secondary connection.

The following commands behave differently when prefaced by `proxy`:

`open` Does not define new macros during the autologin process.

`close` Does not erase existing macro definitions.

`get` and `mget`
: Transfer files from the host on the primary control connection to the host on the secondary control connection.

`put`, `mput`, and `append`
: Transfer files from the host on the secondary control connection to the host on the primary control connection. Third party file transfers depend upon support of the FTP protocol PASV command by the server on the secondary control connection.

`put` *local-file* [*remote-file*]

Store a local file on the remote machine. If *remote-file* is left unspecified, the local filename is used, after processing according to any `ntrans` or `nmap` settings in naming the remote file. File transfer uses the current settings for `type`, `form` (format), `mode`, and `struct` (structure).

`pwd`
Print the name of the current working directory on the remote machine.

`quit`
A synonym for `bye`.

`quote` *arg1 arg2 ...*
The arguments specified are sent, verbatim, to the remote FTP server.

`recv` *remote-file* [*local-file*]
A synonym for `get`.

`remotehelp` [*command-name*]
Request help from the remote FTP server. If a *command-name* is specified, it is supplied to the server as well.

`rename` [*from*] [*to*]
Rename the file *from* on the remote machine, to the file *to*.

`reset`
Clear reply queue. This command resynchronizes command/reply sequencing with the remote FTP server. Resynchronization may be necessary following a violation of the FTP protocol by the remote server.

`rmdir` *directory-name*
Delete a directory on the remote machine.

`send` *local-file* [*remote-file*]
A synonym for `put`.

`status`
Show the current status of `ftp`.

`user` *user-name* [*password*] [*account*]
Identify yourself to the remote FTP server. If the password is not specified, and the server requires it, `ftp` prompts you for it (after disabling local echo). If an account field is not specified, and the FTP server requires it, you will be prompted for it. If an account field is specified, an account command will be relayed to the remote server after the login sequence is completed if the remote server did not require it for logging in. Unless `ftp` is invoked with autologin

disabled, this process is done automatically on initial connection to the FTP server.

`?` [*command*]
A synonym for `help`.

6.2.1 Toggle commands

The following `ftp` commands toggle environment and file transfer conditions (that is, turn a condition on or off, depending on the previous setting). Arguments enclosed in brackets are optional; the brackets aren't part of the command syntax. By default, `bell` is turned off, `case` is turned off, `cr` is turned on, `prompt` is turned on, `rununiqe` is turned off, and `sendport` is turned on.

`bell`
Arrange that a bell be sounded after each file transfer command is completed. The default is off.

`case`
Toggle remote computer filename case mapping during `mget` commands. When `case` is on (default is off), remote computer filenames with all letters in uppercase are written in the local directory with the letters mapped to lowercase.

`cr`
Toggle carriage return stripping during ASCII-type file retrieval. Records are denoted by a carriage return-line feed sequence during ASCII-type file transfer. When `cr` is on (the default), carriage returns are stripped from this sequence to conform with the UNIX convention (a single line feed as the record delimiter). Records on non-UNIX remote systems may contain single line feeds; when an ASCII-type transfer is made, these line feeds may be distinguished from a record delimiter only when `cr` is off.

`debug` [*debug-value*]
Toggle debugging mode. If an optional *debug-value* is specified, it is used to set the debugging level. When debugging is on, `ftp` prints each command sent to the remote machine, preceded by an arrow (the string `-->`).

`glob`
Toggle filename expansion for `mdelete`, `mget`, and `mput`. If

globbing is turned off with `glob`, the filename arguments are taken literally and not expanded. Globbing for `mput` is done as in the C shell (see `csh`(1) in *A/UX Command Reference*). For `mdelete` and `mget`, each remote filename is expanded separately on the remote machine and the lists are not merged. Expansion of a directory name is likely to be different from expansion of the name of an ordinary file: the exact result depends on the foreign operating system and FTP server, and can be previewed by issuing

`mls` *remote-files* `-`

Note: `mget` and `mput` are not meant to transfer entire directory subtrees of files. That can be done by transferring a `tar` archive of the subtree (in binary mode). (See `tar`(1) in *A/UX Command Reference* for more information.)

`hash`
Toggle number-sign (#) printing for each data block transferred. The size of a data block is 1024 bytes.

`prompt`
Toggle interactive prompting. Interactive prompting occurs during multiple file transfers to allow the user selectively to retrieve or store files. If prompting is turned off (default is on), any `mget` or `mput` will transfer all files, and any `mdelete` will delete all files.

`runique`
Toggle storing of files on the local system with unique filenames. If a file already exists with a name equal to the target local filename for a `get` or `mget` command, a `.1` is appended to the name. If the resulting name matches another existing file, a `.2` is appended to the original name. If this process continues up to `.99`, an error message is printed, and the transfer does not take place. The generated unique filename will be reported. Note that `runique` will not affect local files generated from a shell command. The default value is off.

`sendport`
Toggle the use of PORT commands. By default, `ftp` will attempt to use a PORT command when establishing a connection for each data transfer. The use of PORT commands can prevent delays when

performing multiple file transfers. If the PORT command fails, `ftp` will use the default data port. When the use of PORT commands is disabled, no attempt will be made to use PORT commands for each data transfer. This is useful for certain FTP implementations which do ignore PORT commands but, incorrectly, indicate they've been accepted.

`sunique`
: Toggle storing of files on remote machine under unique filenames. Remote FTP server must support FTP protocol STOU command for successful completion. The remote server will report unique name. Default value is off.

`trace`
: Toggle packet tracing.

`verbose`
: Toggle verbose mode. In verbose mode, all responses from the FTP server are displayed. In addition, if verbose is on and a file transfer completes, statistics regarding the efficiency of the transfer are reported. By default, verbose is on.

6.2.2 File transfer parameters

The following `ftp` commands set file transfer parameters. By default, `ftp` uses `ascii` type, `file` format, `stream` mode, and `stream` structure.

`ascii`
: Set the file transfer `type` to network ASCII. This is the default.

`binary`
: Set the file transfer `type` to support binary image transfer.

`form`*format*
: Set the file transfer `form` to *format*. The default format is `file`.

`mode` [*mode-name*]
: Set the file transfer `mode` to *mode-name*. The default mode is `stream` mode.

`struct` [*struct-name*]
: Set the file transfer structure to *struct-name*. By default, `stream` structure is used.

`tenex`
Set the file transfer `type` to that needed to talk to TENEX machines.

`type` [*type-name*]
Set the file transfer `type` to *type-name*. If no type is specified, the current type is printed. The default type is network ASCII.

7. Remote `talk`

Like the `mail` program, the A/UX `talk` program also functions across the network if B-NET addressing conventions are used in the address. The `talk` command establishes an interactive connection between two users across the network. See `talk`(1) in *A/UX Command Reference* for more information.

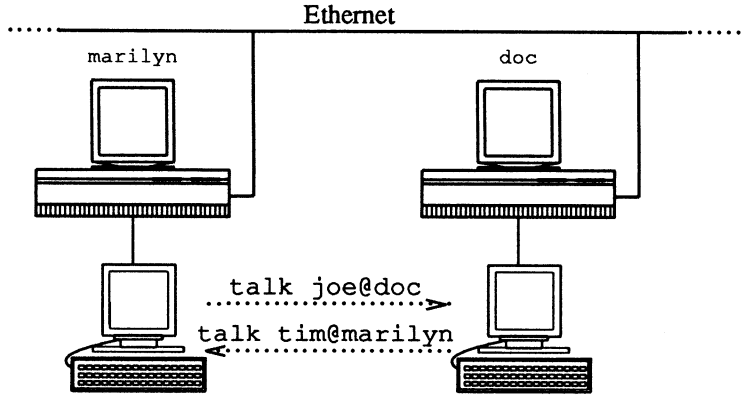
The address format is

username`@`*rhost*

For example, to `talk` to `joe` on the remote host `doc`, as in Figure 3-6, you'd use the command

```
talk joe@doc
```

Figure 3-6. Talking to a user on a remote machine



At this point, your screen clears and a line appears dividing your screen into two windows. At the top of the screen you see the message

```
[No connection yet]
```

The message at the top of your screen may change to read

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

If your login name is `tim` on host `marilyn` as shown in Figure 3-6, the following message appears on Joe's screen:

```
Message from TalkDaemon@doc...
talk: connection requested by tim@marilyn.
talk: respond with: talk tim@marilyn
```

If Joe responds with

```
talk tim@marilyn
```

his screen clears and is divided into two windows. This establishes communication. Once communication is established, both of you may type simultaneously; the output is directed to the separate windows.

CONTROL-l clears the screen (if you run out of space). Your standard erase, kill, and word kill characters work in `talk` just as in your shell environment.

7.1 Terminating `talk`

The *interrupt* signal terminates the `talk` program. The A/UX standard distribution sets the *interrupt* signal to CONTROL-c; you should check `stty`(1) in *A/UX Command Reference* for more information about defining this sequence, or type `stty -a` to see the current setting for the *interrupt* signal on your system.

When you send an *interrupt* signal, the connection closes and you exit the `talk` program. This is also useful if you get tired of waiting for someone to respond to your initial attempt to `talk`.

7.2 Troubleshooting

You may see the message

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

on the top of your screen when you are trying to contact someone. The person you're trying to talk to may have used the `mesg` command to deny

talk permission (if they don't want to be interrupted).

The `mesg` command without any argument returns its current value:

```
mesg
is y
```

If you want to turn off talk permission, use the command

```
mesg n
```

Also, certain commands such as `nroff` and `pr` refuse to be interrupted by `talk` messages.

8. Local and remote shell topics

This section describes how various A/UX shell capabilities are used across the network. This is particularly important to the `remsh` command when you're specifying which machine executes a command, which devices are used, where output files are created, and so on.

8.1 Quoting mechanisms

There are three ways to quote special characters across the network:

- Precede each metacharacter with a backslash (\). A backslash preceding a metacharacter prevents the local shell from interpreting the metacharacter.
- Enclose the metacharacters in single quotes (' '). Single quotes prevent the local shell from interpreting any metacharacters in the enclosed string.
- Enclose the metacharacters in double quotes (" "). Within double quotes, variable substitution and command substitution occur in the local shell, but filename expansion does not.

The difference between using single and double quotes in a network command applies to variable substitution (for example, `$j`) and command substitution (`` `command` ``). When you use double quotes, both of these substitutions occur locally. (See "Quoting" in *A/UX User Interface* for a description of these shell capabilities.)

A backslash before each metacharacter causes the local shell to strip off the backslash and pass the character to the remote shell. For example, in the command

```
remsh doc cat jelly \> jam &
```

the redirection symbol is interpreted by the remote shell, and the file `jam` is created on the remote machine `doc`.

In the command

```
remsh doc cat jelly > bread &
```

the redirection symbol is interpreted by the local shell, and the file `bread` is created on the local machine. This has the same effect as an `rcp` command.

Single quotes can be placed around the metacharacters, command arguments, or parts of the command arguments. All information within the single quotes is passed intact to the remote shell, and interpreted there. All of the following commands have the same effect:

```
remsh doc cat jelly '>' jam &
remsh doc cat 'jelly > jam' &
remsh doc 'cat jelly > jam' &
```

The remote shell interprets the filenames and redirection symbol, creating a new file on the remote host.

Double quotes around the command arguments (or any part of them) are usually used when you want to interpret a shell variable on the local system, or when you want command substitution to occur on the local system. For example, if you have a shell variable set in your local shell

```
j=22
```

and a variable set in the remote shell

```
j=15
```

you can print the remote value using single quotes as follows:

```
remsh doc 'echo $j'
15
```

To print the local value on the remote system, use double quotes:

```
remsh doc "echo $j"
22
```

In the latter case, the remote system prints the value of the variable, which has already been interpreted by the local shell. See "Shell Variables" below and *A/UX User Interface* for more information.

8.2 `remsh` command line considerations

8.2.1 Multiple commands

You can use the `remsh` command to execute multiple commands by quoting the semicolon (`;`) that separates the different commands on the same command line. To run the `date` command followed by the `hostname` command (which returns the name of the system) on a remote host named `doc`, type

```
remsh doc date \; hostname
```

You get a response such as

```
Mon Aug 12 10:55:37 PDT 1985
doc
```

If you don't quote the semicolon by preceding it with a backslash, the `remsh` command will simply terminate after executing a single command. The semicolon character (and the command that follows it) will then be interpreted and executed on the local machine.

If the name of your current system is `lcl`, the command

```
remsh doc date ; hostname
```

gives a response such as

```
Mon Aug 12 10:57:06 PDT 1985
marilyn
```

8.2.2 Pipelines

You may use the `remsh` command to execute pipelined commands on a remote system by quoting the pipe character (|) that separates the multiple processes on the same command line. If you use the backslash to quote the pipe character, you may run different parts of the command line on the local or remote machine, as you see fit.

Suppose you have a remote file `jelly` that needs to be formatted using `neqn` and `nroff` before it is printed. The command

```
remsh doc neqn jelly | nroff | lp &
```

causes the `neqn` process to execute on the remote machine. The output of that process is passed through the pipe. Because the pipe is not quoted, it's interpreted by the *local* shell, so `nroff` runs on the local machine, and the output is passed to the local line printer. If the command were in either of the following formats,

```
remsh doc neqn jelly \| nroff \| lp &
```

or

```
remsh doc 'neqn jelly | nroff | lp' &
```

all processes would execute on the remote machine, and the remote line printer would be used. Quoting the pipes individually allows a range of flexibility in using different resources, but in this case the same effect is obtained when you quote the whole command argument or the individual metacharacters.

For example, if you need to format a local file `bread` on the remote machine and direct the output to a file `frmt.bread` on the local machine, use the command

```
cat bread | remsh doc troff > frmt.bread &
```

Neither of the metacharacters needs to be interpreted on the remote shell in this format.

The `remsh` command may also be used in pipelines. For example,

```
cat bread | remsh doc nroff | remsh bashful lp &
```

In this case, the pipes may be quoted or not quoted, with the same effect.

8.2.3 Filename expansion

The shell metacharacters for filename expansion are interpreted by your local shell unless they are quoted. For example, the command

```
remsh doc ls -l 1*
```

causes the local shell to search your current directory for any filename beginning with `1`. If it finds one or more files beginning with `1` in your local directory, it expands the filenames and sends these across the network. If the local shell doesn't find a filename beginning with `1`, it

prints the message

```
No match.
```

If the remote shell received an expanded name beginning with `1` and doesn't have a file by that name, it prints the same message.

Unless you have two identical files (one local and one remote) that begin with the character `1`, the above command will not produce the effect you want. To run `ls -l` on *remote* files beginning with `1`, you must quote the filename expansion metacharacter with a backslash, single quotes, or double quotes. For example,

```
remsh doc ls -l '1*'
```

or

```
remsh doc ls -l "1*"
```

or

```
remsh doc ls -l 1\*
```

causes the filename expansion to take place in the login directory of your remote account. Assuming that you have one or more filenames beginning with `1` there, the output of the remote process appears on your screen. For example,

```
-rw-------  1 joe    239 Aug 23 11:33 10.check
-rw-r-----  1 joe    555 Sep  3 12:21 15.ftp
-rw-r-----  1 joe  36467 Nov 15 12:36 1234
```

8.2.4 Output redirection

The shell metacharacters for redirecting standard output are interpreted by your local shell unless they are quoted. The command

```
remsh doc cat '1*' >> 1.all &
```

concatenates the files beginning with `1` in your login directory on the remote computer into a file named `1.all` on the *local* computer.

If the output redirection symbols are quoted, for example,

```
remsh doc cat '1* >> 1.all'&
```

all files beginning with `1` in your login directory on the remote computer are concatenated into a file named `1.all` on the *remote* host.

8.2.5 History substitution

History substitution in the C shell does not work across the network. The `remsh` command does not use the shell in which you have a history of your commands, but invokes a new shell to execute its command. The command

```
remsh doc '\!c'
```

does not perform history substitution on either machine. The system prints the message

```
Event not found.
```

8.2.6 Aliases

Aliases that you have set on either system may be used with the `remsh` command if they are set using the format

`alias` *alias-name command(s)*

If, however, you have aliases set as part of an `if` statement, for example,

```
if ($?prompt) then
        alias ls ls -f
        .
        .
        .
endif
```

they cannot be used in a `remsh` command. The remote shell is not interactive, so it doesn't set the alias. Also, aliases that are set in your `.login` file (if you are using the C shell) or your `.profile` file (if you are using the Korn shell) will not be set, because the shell that's invoked by the `remsh` command is not a login shell.

8.2.7 `grep`

When you use `grep` with the `remsh` command to search files on the remote system for a regular expression, the regular expression has to be quoted *twice*. The argument to `grep` is always quoted once to prevent the local shell from interpreting the metacharacters contained in the regular expression; the second quotes are required to prevent their interpretation by the remote shell. You may use single quotes and a backslash:

`remsh` *rhost* `grep 'abc\[0-9\]\*'` *remote-file*

or single and double quotes:

```
remsh rhost grep '"abc[0-9]*"' remote-file
```

(Note that the latter method does not work reliably for embedded dollar signs.) Either command causes the `grep` command to receive the argument and search *remote-file* for the regular expression you intended.

8.2.8 Shell variables

If you create a shell variable `i` and set its value on the local system to the pathname

```
set i =/v2/sys/dist/sys
```

and on the remote computer you have a variable `i` set to another value,

```
set i =/v/test/sys
```

the following command uses the local value:

```
remsh rhost cat "$i | remote-program" &
```

The double quotes cause the local shell to substitute `/v/sys/dist/sys` for the shell variable `i`. That value is then passed with the rest of the command arguments to the remote shell.

If you want the value substituted on the remote system instead, use single quotes instead of double quotes:

```
remsh rhost '$i | remote-program' &
```

This passes the string `$i` to the remote shell, which substitutes the value it has recorded for this variable. Note, however, that the value for `i` must have been set in your `.profile` file on the remote system. If you set the value at the shell prompt during your current login session, the shell invoked by the `remsh` command will not know about it.

Of course, the pathname that is substituted for `$i` has to be accessible on the remote computer for either of these `remsh` commands to have any effect.

8.2.9 Shell scripts

You can also use the `remsh` command to execute multiple commands by enclosing them in a shell script.

If the shell script on the remote machine expects to receive arguments, these arguments may be passed across the network via `remsh`. The command

```
remsh rhost script A B C &
```

provides arguments `A`, `B`, and `C` as values to the variables `$1`, `$2`, and `$3` (or `$argv[1]`, `$argv[2]`, and `$argv[3]`) to the shell script `script` on the remote machine.

There are three things to consider when you pass arguments to a remote shell script in this way:

- You must quote any metacharacters in the arguments.
- You may define variables within the arguments locally, and in that case they may not be available, or may differ, on the remote machine.
- Environmental variables (which could affect the way the shell script runs) may differ on the remote machine.

9. Network issues

This section explains permissions and other considerations that affect how B-NET appears to you, the user. Network administration tasks are covered in *A/UX Network System Administration.*

9.1 Network permissions

Each system on the network has an `/etc/hosts.equiv` file set up by the system administrator. This file contains lines that specify the names of systems whose users are allowed to access this system. The network processes look in this file to determine whether specific hosts on the network are allowed to access this system. If a user is remotely accessing this system from a "friendly" host who is listed in this file, and if the user has an entry in `/etc/passwd` on this system, the user is allowed to access the system using `rlogin` and `remsh` commands.

9.1.1 $HOME/.rhosts

You may set up a `.rhosts` file in your login directory to allow explicit access to your login account. This file is most often used when your user name is different on the local and remote systems (as in the example below where a user named Bill has two login names, `bill` and `wrn`). You could also include login names of other users, which would allow them

unrestricted access to your account if they logged in using the `-l` flag option to the `remsh` or `rlogin` commands.

> *Note:* When you include a line in your `.rhosts` file that allows a certain user (or all users from a certain system) access to your account, in effect you're giving them your password. Normally you should restrict this file to your own login names.

The `.rhosts` file uses the format

hostname [*username*]

Where the parts are interpreted as follows:

hostname — The name of a remote system followed by any number of spaces or tab characters.

username — The name of a user on *hostname*. This argument is optional. If the *username* parameter is not stated, anyone from *hostname* is allowed access to your account.

Each line of this format indicates that *username* from *hostname* has permission to access this particular account.

This is most commonly used by users with account names that differ across systems. If Bill has the login name `bill` on `sneezy` and the login name `wrn` on `doc`, his `.rhosts` file on `sneezy` should include the line

```
doc wrn
```

His `.rhosts` file on `doc` should include the line

```
sneezy bill
```

From his `sneezy` account, he can then use the command format

```
rlogin doc -l wrn
```

or

```
remsh doc -l wrn command
```

If he is logged in to `doc`, he can use the format

```
rlogin sneezy -l bill
```

or

```
remsh sneezy -l bill command
```

Network users can also share accounts. For example, a user has the login name `bill` on the local system `sneezy`. Another user has the login name `ramona` on remote system `doc`. `ramona` is logged in to her account on `doc`. She wants to run the `date` command remotely on `sneezy`, but she doesn't have an account on that machine.

If `bill` has a `.rhosts` file containing the line

```
doc ramona
```

Ramona can use the command

```
remsh sneezy -l bill date
```

9.1.2 $HOME/.netrc

You may set up a `.netrc` file in your login directory to provide information to the `ftp` autologin process. This file is used to contain login and initialization information for the remote FTP server.

> *Note:* This file is often not used for security reasons, since a superuser on the local system is able to obtain your password for the remote FTP site using this file. Note that, if you supply your remote password in this file, `ftp` aborts the autologin process unless the `.netrc` file is unreadable to anyone but yourself. Thus, after you create a `.netrc` file you should change the mode to read-write permissions for yourself only.

The `.netrc` file uses the following tokens; they may be separated by spaces, tabs, or newlines:

`machine` *rhost*
: (where *rhost* is the remote machine name). The autologin process searches the `.netrc` file for a `machine` token that matches the remote machine specified on the `ftp` command line or as an `open` command argument. If it finds the right machine name, it processes the rest of the tokens in your `.netrc` file, stopping when the end-of-file condition is reached or another `machine` token is encountered.

`login` *name*
(where *name* is your login name on the remote machine). If this token is present, the autologin process initiates a login using the specified name.

`password` *string*
(where *string* is your password on the remote machine). If this token is present, the autologin process supplies the specified string if the remote server requires a password as part of the login process. Note that if this token is present in the `.netrc` file, `ftp` aborts the autologin process if the `.netrc` is readable by anyone other than the user.

`account` *string*
(where *string* is an additional account password). If this token is present, the autologin process supplies the specified string if the remote server requires an additional account password, or the autologin process initiates an ACCT command, if it does not.

`macdef` *name*
(where *name* is a macro definition). This token works like the `ftp macdef` command. A macro is defined with the specified name; its contents begin with the next `.netrc` line and continue until a null line (consecutive newline characters) is encountered. If you define a macro named `init`, it is automatically executed as the last step in the autologin process.

9.2 Abbreviated command formats

Your system administrator can set up a `/usr/hosts` file on each system. If this file exists, include the path `/usr/hosts:` in the `PATH` variable in your local `.login` file. (Actually, this file can have any pathname as long as you include it in your `PATH` variable.)

When you have done this, you may omit the `rlogin` and `remsh` names from the command line. In this case, the `rlogin` command format is

rhost

or

rhost `-l` *username*

The `remsh` command format is

rhost command

or

rhost `-l` *username command*

When used in this format, *rhost* must be the official name of the system (that is, the name shown in `ruptime` output).

9.3 System nicknames

Each system on your network should have at least one alias or "nickname" consisting of one or two characters. These are set up by the system administrator in the `/etc/hosts` file.

To find out if a remote system named `bashful` has a nickname available, use the command

```
grep bashful /etc/hosts
```

This should print a line such as

```
192.111.0.8 bashful ba b
```

The names following the hostname are the nicknames you may use instead of the hostname in most network commands. In this case, you could use the format

```
rlogin ba
```

or

```
rlogin b
```

Chapter 4
Using UUCP

Contents

Chapter 4

Using UUCP

1. Introduction to UUCP

The UUCP network links A/UX systems over telephone lines or direct serial lines. You can access A/UX systems that are part of the UUCP network to transfer files, execute commands on remote machines, and send and receive mail.

UUCP is generally used by people who want to link the different A/UX machines they have. It is also commonly used to link a business's different offices or to link different businesses.

The UUCP network is not a commercial network that you have to "join" nor an open system that just anyone can access. The system administrator sets permissions that control who is allowed to use any of the UUCP machines. Before you can use UUCP you must receive access permission from the host machine and your machine must be set up correctly. For information about setting up your machine to connect to other systems, please see *A/UX System Administrator's Reference* or consult your system administrator.

1.1 UUCP access

When you use UUCP to gain access to a remote machine, the directory you enter on the remote machine is the directory all UUCP users enter. That directory (usually `/usr/spool/uucppublic`) is designated as the **public area**.

The system administrator decides where else on the remote machine you can go and sets permissions accordingly. Many systems restrict UUCP access to the public directory. If you have questions about what you can access on the remote machine, please consult your system administrator.

1.2 Conventions in this chapter

The terms **computer**, **machine**, **system**, and **host** are used interchangeably in this manual to mean a computer on a network. In

general, **local** refers to the machine on which you initially log in using the standard A/UX login. **Remote** refers to other computers on the network that you can access using the commands in this manual.

All commands are terminated with a RETURN unless stated otherwise.

2. Using UUCP network commands

2.1 Network status

You get the status of the last transfer to each system on the network by using the `uustat` command. For example,

```
uustat -mall
```

reports the status of the last transfer to all of the systems known to the local system. The output might appear as

```
mhb5c   08/10-X:35      CONVERSATION SUCCEEDED
resear  08/20-17:01     CONVERSATION SUCCEEDED
minimo  07/22-16:31     DIAL FAILED
austra  08/20-18:36     WRONG TIME TO CALL
ucbvax  08/20-20:37     LOGIN FAILED
```

where the status indicates the time and state of the last transfer to each system.

When you're sending files to a system that has not been contacted recently, it's a good idea to use `uustat` to see when the last access occurred, because the remote system may be down or out of service.

2.2 Network names

You can find the names of the systems on the network via the `uuname` command. Only the names of the systems in the network are printed.

3. Transferring files

3.1 Filename syntax in UUCP

When you refer to a file on a remote system, you must use a specific syntax that identifies the file uniquely. There are also some restrictions on pathnames to prevent security violations. For example, pathnames may not include "`..`" as a component because it is difficult to determine whether the reference is to a restricted area.

The basic syntax is

system-name ! *pathname*

where *system-name* is the name of a system that UUCP recognizes. Note that `csh` users must escape the "!", for example, by preceding the exclamation point with a backslash (\). The *pathname* may contain any of the following:

- A file's full pathname, such as

  ```
  zeus!/usr/sid/stuff
  ```

  where `zeus` is the system and `/usr/sid/stuff` is the file's full pathname (the file `stuff` in the `/usr/sid` directory).

- A directory name, as in

  ```
  zeus!/usr/sid
  ```

- The login directory on a remote system, indicated with the ~ character. The combination `~fred` refers to `fred`'s login directory on the remote system. For example,

  ```
  zeus!~fred/names
  ```

  expands to

  ```
  zeus!/usr/sys/fred/names
  ```

  if the login directory for user `fred` on the remote system is `/usr/sys/fred`.

- The public area, referenced by a similar use of the prefix *~user* preceding the pathname. For example,

  ```
  zeus!~/sally/audit
  ```

  would expand to

  ```
  zeus!/usr/spool/uucp/sally/audit
  ```

  if `/usr/spool/uucp` is used as the public directory.

- The current directory (or the login directory on the remote system) used by default if the pathnames you enter don't use any of these combinations or prefixes. For example,

```
zeus!stuff
```

would expand to

```
zeus!/usr/you/stuff
```

You may use the naming convention in reference either to the source filename (where the information is coming from) or to the destination filename (where the information is going).

3.2 File transfer syntax

The following command transfers the file `prospects` from your machine to `fred`, who is on the directly connected machine `mars`.

```
uucp prospects mars!~/fred/prospects
```

You can use UUCP to pass files between systems that aren't directly connected in the network. To do this, you send the file via intermediate machines.

For example, if you want to send a file from system A to system C, but they're not directly connected, you can route it through system B, if both A and C are connected to B. To do this, specify the file's pathway to go from system A to system B and on to system C. The command uses a variation of the **bang** (`!`) syntax to describe the path to be taken.

For example, if you were on system `zeus` and you wanted to transmit the file `prospects` to `fred`'s home directory on system `thor`, but had to route it through `mars` and `venus`, you would specify the transfer with

```
uucp prospects mars!venus!thor!~/fred/prospects
```

Note that the pathname is the path the file must take to reach `thor`. Fetching a file from another system via intermediate **nodes** is done similarly. For example,

```
uucp mars!venus!thor!~/fred/prospects x
```

fetches `prospects` from `fred`'s home directory on system `thor` and renames it `x` on the local system. The forwarding prefix is the path *from* the local system to the remote system, *not* the path from the remote to the local system.

The following sections discuss different combinations of transfers.

3.3 Using metacharacters in file transfers

You may use `uucp` with great flexibility to transfer any number of files to a remote system. For example, the syntax supports the `*`, `?` and `[...]` metacharacters (for further information about these metacharacters, consult *A/UX User Interface*). For example,

```
uucp *.[ch] zeus!dir
```

transfers *all* files whose names end in `c` *or* `h` to the directory `dir` in your login directory on `zeus`.

You may fetch files from a remote system in a similar manner. For example,

```
uucp zeus!\*.[ch] dir
```

fetches all files ending in `c` *or* `h` from the login directory on `zeus` and places the copies in the subdirectory `dir` on the local system.

If you specify that the files be placed in a directory that doesn't exist, the directory will be created.

3.4 Switching

You may arrange file transfer in such a way that the local system effectively acts as a switch. For example,

```
uucp mars!dog zeus!mouse
```

fetches `dog` from your login directory on `mars`, renames it `mouse`, and places it in the login directory on `zeus`.

3.5 Broadcasting

Broadcast capability (that is, copying a file to many systems) is *not* supported by UUCP. You may, however, simulate it via a shell script, as in

```
for i in zeus mars venus
do
        uucp stuff $i!broad
done
```

(For more information about shell scripts and how to use them, please refer to *A/UX User Interface*.)

Unfortunately, because one `uucp` command is spawned for each transmission, you cannot track the transfer as a single unit.

3.6 Spooling

To continue modifying a file while a copy is being transmitted across the network, use the `-C` flag option. This forces a copy of the file to be queued. By default, `uucp` doesn't queue copies of the files because it wastes storage space and CPU time.

For example, the following command forces the file `work` to be copied into the spool directory before it's transmitted:

```
uucp -C work zeus!~/you/work
```

It can be risky to copy a file as you're modifying it, so this command is generally not run in the background while the source file is open.

3.7 Notification of transfer

The success or failure of a transmission is reported to users asynchronously via the `mail` command. `uucp` notifies you in a file of your choice. The choices for notification are:

- Notification returned to the requester's system via the `-m` flag option. This is useful when you're distributing files to other machines. Instead of your logging in to the remote machine to read mail, mail is sent to you when the copy is finished.
- A variation of the `-m` flag option is to send notification to a file (`-m`*file*, where *file* is a filename). For example,

  ```
  uucp -mxfer /etc/passwd mars!/usr/fred
  ```

  sends the file `/etc/passwd` to system `mars` and places the file in the file `/usr/fred`. The transfer's status is reported in the file `xfer` as

  ```
  uucp job 0306 (8/20-23:08:09) (0:31:23)
  /etc/passwd copy succeeded
  ```

- `uux`, which tells `uucp` to execute a command remotely, always reports the exit status of the remote execution unless you suppress notification (via the `-n` flag option). See `uux`(1) in *A/UX Command Reference* for more information.

4. Executing commands on a remote machine: uux

You may execute commands on a remote machine using uux, the remote execution facility. For example,

```
uux "!diff zeus!/etc/passwd \
thor!/etc/passwd > !pass.diff"
```

executes the command diff on the password files on zeus and thor and places the result in pass.diff. You need to use double quotes with uux when your command includes special shell characters such as <, >, ;, or |.

4.1 Tracking and status

4.1.1 The job number

You may associate a single job number (also known as the process ID) with each command execution so that you can terminate the job or obtain its status.

By default, the uucp and uux commands do *not* print the job number for each job. If you make the environment variable JOBNO part of your environment and exported,

```
JOBNO=ON
export JOBNO
```

in the Bourne shell or Korn shell, or

```
setenv JOBNO
```

in the C shell, uucp and uux print the job number. Similarly, if you wish to turn the job numbers off, set the environment variable as follows:

```
JOBNO=OFF
```

If you wish to force printing of job numbers without using the environment mechanism, use the -j flag option. For example,

```
uucp -j /etc/passwd mars!/dev/null
```

forces the system to print the job number (282):

```
uucp job 282
```

If you don't use the `-j` flag option, you may find your jobs' numbers using the `uustat` command:

```
uustat
```

Then the system prints the job numbers for the current jobs:

```
0282 tom mhtsb 08/20-21:47 08/20-21:47 JOB IS QUEUED
0272 tom mhtsb 08/20-21:46 08/20-21:46 JOB IS QUEUED
```

In this example, the two jobs (282 and 272) are queued.

4.1.2 Job status: `uustat`

The `uustat` command lets you check on one or all jobs that have been queued. You may use the job number as a key to query status of the particular job. For example,

```
uustat -j0711
```

asks about the status of job 0711. The system responds:

```
0711 tom mhtsb 07/30-02:18 07/30-02:18 JOB IS QUEUED
```

There are several possible status messages for a given job; the most frequent ones are `JOB IS QUEUED` and `JOB COMPLETED`. For more information, please consult `uustat`(1) in *A/UX Command Reference*.

4.2 Job control

Using the unique job number generated for each `uucp` or `uux` command, you can control jobs in several ways.

4.2.1 Terminating jobs

You can terminate a job that consists of transferring many files from several different systems using the `-k` flag option of `uustat`:

```
uustat -k0711
```

If any part of the job has left the system, only the *remaining* parts of the job on the local system are terminated.

4.2.2 Requeuing a job

`uucp` clears out its working area of jobs on a regular basis (usually every 72 hours) to prevent the buildup of jobs that cannot be delivered. You may use the `-r` flag option to force a job's date to become the current date, thereby lengthening the time that `uucp` attempts to

transmit the job. Please note that the `-r` flag option does not make the job *immortal*, but only postpones its deletion until the next cleanup. You may use the `-r` flag option over again, however, extending the job's lifetime even further.

5. `mail`

The `mail` command uses `uux` to forward mail to other systems. (The A/UX standard distribution makes `mail` the alias for `mailx`.) The address format for mailing across the UUCP network is

`mail` *system*`!`*user*

The `mail` command invokes `uux` to execute `rmail` on the remote system (`rmail` is a link to the `mail` command).

For example, if you were sending a message to `groucho` on `mars`, you would type

```
mail mars!groucho
```

The `mail` facility starts up and you may enter your message as you normally would in `mail`. When you're finished, type an *eof* to indicate the end of message, and the message will be sent to `groucho` on `mars`. (The A/UX standard distribution defines the *eof* sequence as CONTROL-d. See `stty`(1) in *A/UX Command Reference* for information about the *eof* sequence, or type `stty -a` for the current setting for *eof* on your system.)

Forwarding mail through several systems, for example,

```
mail zeus!mars!groucho
```

does not use the `uucp` forwarding feature, but is simulated by the `mail` command itself. Also see "UUCP" in Chapter 2.

6. Other utilities that use UUCP

There are several utilities that rely on `uucp` or `uux` to transfer files to other systems.

6.1 `uuto`

The `uuto` command uses the `uucp` facility to send files while allowing the local system to control the file access.

Suppose your login is `zorro` and you're on system `thor`. You have a friend on system `mars` with a login name of `cisco`. You could send the file `robber` using `uuto` by entering the following:

```
uuto robbers mars!cisco
```

The file is sent to a public directory defined by UUCP. In this example, `cisco` receives the following mail:

```
>From nuucp Tue Jan 25 11:09:55 1983
/usr/spool/uucppublic/receive/cisco/mars\
//filename from thor!zorro arrived
```

See `uuto`(1) in *A/UX Command Reference* for more details.

6.2 USENET news

A large number of UNIX sites worldwide participate in USENET (or `netnews`), a distributed electronic bulletin board with a wide range of topics. To participate, your system administrator must get the USENET software from the University of California at Berkeley (`netnews` is not available through Apple Computer). Check with your system administrator to see whether your site in on the USENET.

6.3 Other applications

The Office Automation System (OAS) uses `uux` to transmit electronic mail between systems in a manner similar to the standard `mail` command. Some sites have replaced utilities, such as `lp`, with shell scripts that invoke `uux` or `uucp`. Other sites use the UUCP network as a backup for higher speed networks.

Chapter 5
Using cu

Contents

Figures

Chapter 5

Using cu

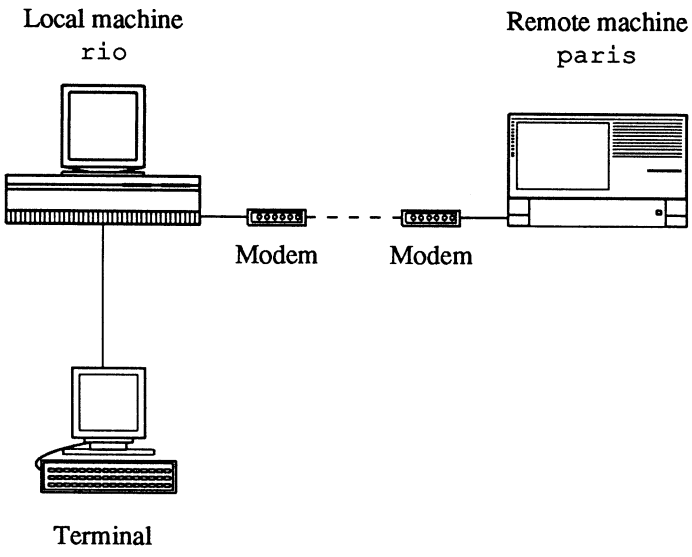
1. What is cu?

cu is a program that manages interactive communications between your machine (the **local** machine) and another machine (the **remote** machine). The remote machine may or may not be another A/UX system; in either case, connection is usually made through a serial line (either a direct connection or with a modem). Once you've connected the machines using cu, you can work with the remote machine as if you were directly connected to it.

cu also provides additional features: You can transfer files back and forth between machines, send the output of a local command to a remote machine, capture the output of a remote command on the local machine, and even escape to an interactive shell on the local machine. In short, cu is an interactive serial communications program. There is, however, no guarantee that one machine receives exactly what the other machine sends: cu does no error checking on data transfers. This is not likely to be a problem, except during large file transfers over (possibly noisy) phone lines. In a case where error checking and error correction are necessary, some other communications package (such as UUCP or the public-domain `kermit`) should be used. If you are transferring small files, you could also run an error check on them and then resend them if you find any errors.

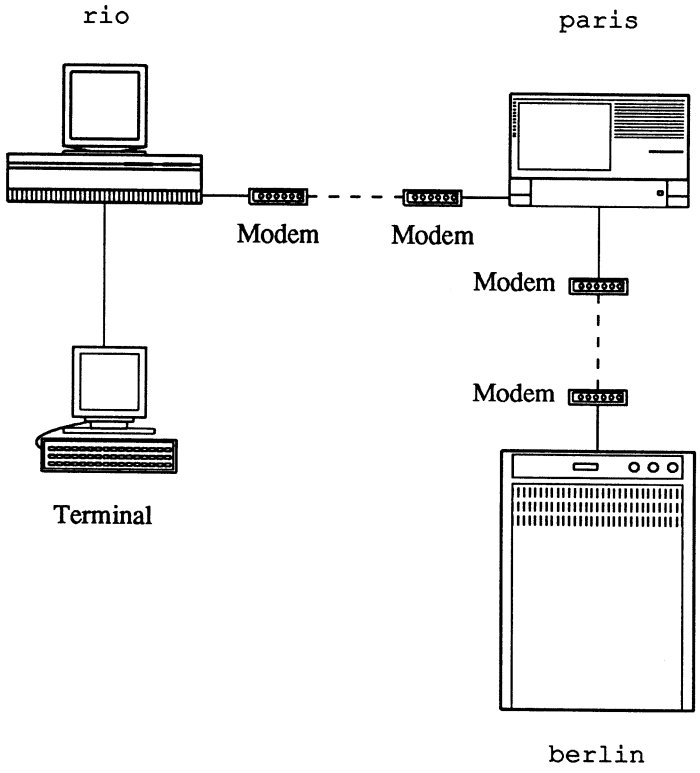
The following figures illustrate several possible ways of using cu. In Figure 5-1, a user sitting at a terminal is logged directly into a machine named `rio`. On `rio`, the user is running cu to connect through a modem to a remote machine, `paris`. Once the user has logged into `paris`, he may interact with `paris` exactly as if his terminal were connected directly to it. In addition, he may instruct the machines to transfer files in either direction. This is a typical use of cu.

Figure 5-1. A simple `cu` configuration



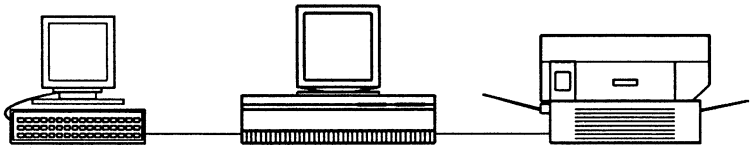
In Figure 5-2, the same user has invoked `cu` once again, this time on the remote machine, `paris`. He may now connect to a remote mainframe, `berlin`, and engage in an interactive login session on it. Once again, the user can transfer files between `paris` and `berlin`; he can also transfer files between `rio` and `berlin`, or (as before) between `rio` and `paris`.

Figure 5-2. Using `cu` remotely



As a final example of using `cu`, consider the situation illustrated in Figure 5-3. Here, a user is logged in over a normal serial line to the local machine, `rio`, using the standard A/UX login procedures. Then, the user has invoked `cu` to manage an interactive session over another serial line which is connected to a laser printer. If the printer has enough intelligence to manage its end of the conversation, the user can interact directly with the printer in its own command language. Note in particular that `cu` does not assume that the remote machine is an A/UX system.

Figure 5-3. Using `cu` with other machines



Terminal Local machine Laser printer

There is another interactive communications program, `tip`, that is very similar to `cu`. It provides essentially the same features as `cu`, plus some additional ones. Which of these two programs you use will depend on your particular needs and on the configuration of your system. This manual describes `cu`, since it is more widely available than `tip`. For information on `tip`, see `tip`(1) in *A/UX Command Reference*.

2. Invoking `cu`

The `cu` command syntax is

```
cu [-d] [-e] [-h] [-lline] [-m]
[-n] [-o] [-sspeed] [telno] [| systemname] [| dir]
```

The two flag options you are most likely to use are `-l` and `-s`, which specify the line and the speed for the connection to the remote system. For instance,

```
cu -l /dev/tty0 -s1200
```

if successful, establishes a connection at 1200 baud through the serial port `/dev/tty0`. If there is a system expecting a login connected to the other end of this line, you see a typical login message, for example:

```
UNIX System V.2 (paris)

login:
```

At this point, you can log in to the remote system and access its resources (such as files and peripherals) exactly as if you had originally

logged in to the machine `paris`. Additionally, `cu` allows ASCII file transfer between machines.

You should note that you may not have any choice in the serial line used by `cu`, since this is configured by your system administrator. If the line you choose is not available, `cu` responds with an error message. For instance, if the line `/dev/tty0` (as requested above) is not configured to allow `cu` access, you get the following message:

```
Connect failed: Requested device not known
```

To guard against multiple users connecting to remote machines through the same line, `cu` opens modems and terminal lines with exclusive access and honors UUCP's locking protocol (see Chapter 9, "The UUCP System" in *A/UX Local System Administration* and `uucp`(1) in *A/UX Command Reference*).

You may also specify a telephone number as a command line argument. If the telephone number is the *only* argument, `cu` will attempt to connect to an available line and dial the specified number. Or, you can specify a remote system name (as defined in the system file `/usr/lib/uucp/L.sys`) on the command line. (See Chapter 9, "The UUCP System" in *A/UX Local System Administration* for more information.) For example, if your system is appropriately configured, the following may be acceptable `cu` command lines:

```
cu 8452583
cu paris
```

Other available flag options are as follows:

`-h`	Emulate local echo, thereby supporting calls to systems which expect terminals to be set to half-duplex mode.
`-t`	Set appropriate mapping of RETURN to RETURN-line feed pairs (used when dialing an ASCII terminal which has been set to auto-answer).
`-d`	Cause diagnostic traces to be printed.
`-e`	Set even parity generation for data sent to remote.
`-o`	Set odd parity generation for data sent to remote.

`-m`	Set for use of a direct line which has modem control.
`-n`	Request that phone number be dialed by the user rather than taken from command line.
`dir`	Ensure that `cu` use the line specified with the `-l` flag option.

3. Using `cu`

Once you have invoked `cu` and established a connection with a remote host, `cu` manages two processes, a transmit process and a receive process. The transmit process sends everything typed by the user to the remote machine, except lines beginning with the `cu` escape character, the tilde (~). Similarly, the receive process takes all output produced by the remote machine and sends it to the user's local terminal screen (with an exception noted below). With these two differences aside, the connection established by `cu` operates exactly like any other serial connection between devices.

At some point during your `cu` session, you may want to run a command on the local machine. There are two ways to do this: by escaping to an interactive shell, or by instructing the local machine to run a single command. You can escape `cu` to an interactive shell on the local machine by typing the following command at the beginning of a line:

```
~!
```

When you exit that shell, you return to `cu`.

To run a single command on the local system, use the sequence

~!*cmd*

For example, if you want to sort a file on the local machine prior to transferring it to the remote machine, you might type the following:

```
~!sort staff.86 > newstaff.86
```

The command specified is run in a subshell via `sh -c`. Accordingly, the command should conform to the syntactic conventions of the Bourne shell. When the command completes, you return to `cu`.

`cu` provides a further command to allow you to run a command on the local system and send any output to the remote system. For example, the command

```
~$cat test.sh
```

sends the contents of the local file `test.sh` to the remote system. Note that those contents are *not* put into a file, but simply sent through the serial line to the other machine. Presumably, unless you intend havoc to ensue, the contents of the file `test.sh` are either commands or data that are meaningful to the remote system.

There are two commands recognized by `cu`'s receive process. If the remote machine sends a line beginning with the sequence

~>:*filename*

all the subsequent output, up to the next line beginning with the sequence ~>, is diverted into a file having the name *filename*. Similarly, output is appended to *filename* if the sequence

~>>:*filename*

is encountered.

4. File transfer with `cu`

You may send a file from the local machine to the remote machine by using the `~%put` command. For example, the command

```
~%put chap.5
```

sends the file `chap.5` from the current directory on the local machine to the current directory on the remote machine. If a file with that name already exists on the remote machine, it is overwritten by the version from the local machine. If there is no file `chap.5` in the current directory on the local machine, `cu` gives an error message. If you encounter this message, you may want to move to a different directory before retrying. For example,

```
~%cd /users/doc/rachel/book
```

To give the file transferred a different name on the remote system, you may provide a second argument to the command:

```
~%put chap.5 sec.5
```

This transfers the file, as before, giving the remote file the name `sec.5`.

To transfer a remote file to the local machine, use the `~%take` command. For example,

```
~%take appendix.3
```

This works exactly like the `~%put` sequence, but in reverse.

5. Using `cu` with non-A/UX devices

As noted in the first section of this chapter, it is possible to use `cu` to communicate through a serial line with devices that do not run the A/UX operating system. For instance, it may be useful to connect to a laser printer in order to test the communications settings of the printer or to modify certain parameters. This can be done easily with `cu`; all that `cu` expects is that the remote device accept whatever input you give to it and (perhaps) respond with some output of its own.

In cases where the remote machine is not running A/UX (or any similar derivative of the UNIX operating system), however, certain `cu` commands will not work properly. In particular, the commands `~%take` and `~%put` probably will not function correctly, since they require that the remote machine understand at least the three commands `stty`, `echo`, and `cat`. In such cases, however, it may be possible to achieve almost the same effect. For example, if you want merely to send to the remote device a string of characters stored in the file `test.ps` on the local machine, you may invoke the following command:

```
~$cat test.ps
```

This runs the A/UX command `cat test.ps` and sends the output to the remote device. Presumably, `test.ps` contains commands or data that are meaningful to that device.

Generally, there is no easy way to get the remote device to simulate the effect of the `~%take` command, primarily because the remote device may not have anything even remotely resembling a file system within it (so there is no sense in asking it to send you a file). The closest you can come to such an operation may be to have the remote device echo an appropriate command (`~>:`*filename*) to cause the local machine to

capture the subsequent output and save it in an A/UX file. This may or may not work for your purposes.

6. Quitting cu

To terminate the remote interactive connection established by `cu`, type the command

```
~.
```

This drops the existing connection and exits `cu`, returning you to a shell on the local machine.

You may or may not want to log out of the remote machine before terminating the `cu` session. If, for example, you have established the connection for the purpose of running the shell programs `take` or `put`, then you should *not* log out prior to dropping the connection. See `take`(1) or `put`(1) for information on using those programs with `cu`.

7. Summary of cu commands

When you type characters, they're normally transmitted directly to the remote machine. If you type a tilde (~) at the start of a line, the line is interpreted as a command to `cu`. `cu` recognizes the following commands:

`~.`	Drop the `cu` connection and exit.
`~%cd` *name*	Move to the specified directory. If you don't enter a directory name, you move to your home directory.
`~!`	Escape to an interactive shell on the local system. When you exit the shell, you return to `cu`.
`~!`*cmd*	Run *cmd* on the local system (via `sh -c`).
`~$`*cmd*	Run *cmd* on the local system and send its output to the remote system.
`~%put` *file* [*tofile*]	Copy *file* from the local machine to the remote machine. If the parameter *tofile* is omitted, the file is given the same name on the remote system.

`~%take` *file* [*tofile*]	Copy *file* from the remote machine to the local machine. If the parameter *tofile* is omitted, the file will be given the remote filename on the local machine.
`~%break`	Send a BREAK to the remote machine.
`~%nostop`	Toggle between DC1/DC3 (CONTROL-q and CONTROL-s) input control protocol and no input control. This is useful if the remote system does not respond properly to the DC1 and DC3 characters.
`~~...`	Send the line ~... to the remote machine.

The receive process recognizes two commands:

~> :*filename*	Put all subsequent output, up to the first line beginning with the sequence ~>, into *filename*.
~>> :*filename*	Append all subsequent output, up to the first line beginning with the sequence ~>, to *filename*.