The UUCP System

Editor's Note: Cross-references in the text refer to chapters in the companion book, *UNIX: The Complete Reference, Second Edition,* by Rosen, Host, Klee, Farber, and Rosinski. The material in this chapter is intended primarily for historical purposes in describing the UUCP system, which has been replaced largely by the TCP/IP environment. It should be noted, though, that a number of these UUCP utilities are still available on UNIX systems.

The UNIX System was designed to allow different computers to communicate easily. It is noted for its wide range of communications and networking capabilities, which include facilities for electronic mail, file transfer, logging in to remote machines, remote execution of commands, and file sharing. The original utilities designed for these tasks are included in a part of the UNIX System called the UUCP System. Although the UUCP System has been supplanted for many uses by TCP/IP utilities (described in Chapter 9), it is still used extensively and you may still find its capabilities useful.

This chapter will describe the Basic Networking Utilities (BNU), included with versions of UNIX based on UNIX System V Release 4, which include commands you can use to call another system, and the UUCP System. (Most other versions of UNIX should have these commands as well.) You can use the UUCP System to transfer files between computers or to execute a command on a remote machine. In this chapter, you will learn how to use the Basic Networking Utilities to call up and log in to remote systems. You will learn how to transfer files between computers using several different UUCP System utilities and how to handle files sent to you via the UUCP System. Also, you will learn how to execute commands on remote systems using the **uux** command. Moreover, in this chapter you will learn how to administer the UUCP System.

The Basic Networking Utilities

The Basic Networking Utilities include the UUCP System and the **cu** and **ct** commands used to call other machines known to your system. The UUCP System is named after the **uucp** command (*U*NIX-to-*U*NIX System *copy*), used to transfer files. Besides the **uucp** command, the UUCP System includes a wide range of other commands. The most important user commands in the Basic Network Utilities are displayed in Table 1.

Command	Action
cu	Call another system and manage a dialog, including ASCII file transfer.
ct	Dial a remote terminal and generate a login process.
uucp	Copy files from your system to another, allowing the remote system to control file access (by default, files are copied into a public directory, */var/spool/uucppublic*).
uupick	Search */var/spool/uucppublic* for files sent to you, and prompt for their disposition.
uux	Execute a command on a remote system.
uuname	Print the names of all systems known to **uucp**.
uustat	Display the status of current **uucp** jobs; cancel previous jobs; provide system performance information.

TABLE 1 Important BNU Commands for Users

The Development of the UUCP System

The UUCP System dates back to 1976, when it was first conceived and developed by Mike Lesk at Bell Laboratories. A rewritten, enhanced, and improved version of the UUCP System, known as Version 2 UUCP, was included in early releases of the UNIX System and was the standard version until 1983. However, the extensive use of the UUCP System across a wide range of communications facilities made it necessary to enhance its capabilities and performance. A new version of the UUCP System was developed by Peter Honeyman, David Nowitz, and Brian Redman at Bell Laboratories in 1983, supporting a wider range of networking, providing administrative facilities, and correcting deficiencies in Version 2 UUCP. This version, known as *HoneyDanBer* (from the logins of its developers, *honey, dan,* and *ber*), was incorporated in UNIX System V Release 2 and is the basis of the Basic Networking Utilities in Release 4. The Basic Networking Utilities are (almost completely) compatible with Version 2 UUCP Systems, so that computers running older versions of the UUCP System can communicate with machines running newer versions of UUCP.

Other versions of the UUCP System, including *Taylor UUCP,* have been developed as part of the GNU Distribution. The Taylor UUCP System is included with Linux. How commands work in Taylor UUCP is similar to how they work in Version 2 UUCP.

The UUCP Network

Traditionally, one of the networks used to transfer electronic mail between UNIX computers was the *UUCP Network,* a collection of machines known to one another. Machines on the UUCP networks are connected in a variety of ways: They are linked via dedicated private-line communications; they are connected on a local area network; or they are connected via modems through the telephone network. (Today, instead of the UUCP Network, electronic mail is generally transferred over the Internet, using TCP/IP.)

If your system is part of the UUCP Network, it may be connected to only a few other systems, or it may be connected to hundreds in a network of interconnected systems. Let's consider a small hypothetical network. In this example, there are seven machines. They can be diverse machines, made by different manufacturers, with different hardware, and even running different releases or versions of the UNIX System. They can even be running other operating systems for which the UUCP System has been developed.

In this network, the machine *jersey* knows about *alpha, bravo, chucky, hotdog*, and *zephyr.* Meanwhile, hotdog and zephyr also know about each other, but *ferdie* is known only to zephyr. When we say that jersey knows about alpha, it means that jersey can connect (via dial-up telephone line, LAN, or dedicated private line) to alpha, and that jersey is allowed to log in to alpha to exchange information.

Normal network connections are bidirectional. If jersey can connect, log in, and send information to alpha, then alpha can connect, log in, and send information to jersey. This is not always necessary, however, and the UNIX System provides the flexibility to have a system call out but not receive calls, or vice versa. For example, often a large computer will *poll* many smaller computers in a network; in this case, the smaller computers receive calls but do not call out.

A user on jersey can exchange information with any user on alpha, bravo, chucky, hotdog, or zephyr: Communication is particularly flexible, since any user on any of these systems can communicate with any other user on these systems. A user on alpha can go through jersey to get to a user on bravo, or through jersey and zephyr to get to a user on ferdie.

This network of computers is easily expanded. As soon as the system administrators agreed to connect zephyr and ferdie (by exchanging and installing one line of system administration information), all users on ferdie had access to the entire network of machines. If, in the future, ten other machines connect with ferdie, then all of the users on those machines can also be connected to jersey, alpha, bravo, chucky, hotdog, and zephyr.

Regardless of the nature of the connection, the systems and the details of how to connect with them are specified in *uucp configuration files.* The UUCP System uses the information contained in these configuration files, maintained by your system administrator, to determine how to connect with a remote system. Configuration files, and the way that **uucp** uses them, will be discussed later in this chapter. As a user, you need only know a path to your destination machine. You do not need to know the nature of connections between the computers in this path.

The Structure of the UUCP Network

The UUCP Network is a network of machines that has no central administration. This means that no computer manages the entire network. You join the network by finding another machine already on the UUCP Network that agrees to be your neighbor, in the sense that this computer agrees to add configuration information for setting up a connection with your machine. Once you have found a neighbor, you can then connect through this machine to all other machines that can connect to it. To communicate with a remote computer not known to your computer, you need a path to this computer. The problem of finding such a path is partly mitigated by the existence of the *UUCP Map,* which specifies connections between various computers. The UUCP Map is kept by the UUCP Network Project, which posts this map each month in the *comp.mail.maps* newsgroup. Each host on the UUCP Network pays the costs of the links connecting to other machines.

Using uuname

The ease of connecting to and extending the UUCP Network is one reason why it has grown to include thousands of machines. When your machine is part of the UUCP Network, it will know about and be known by one or more other machines. To find out which systems your system knows about, use the **uuname** command, as in the following example:

```
$ uuname
alpha
bravo
chucky
hotdog
zephyr
```

The **uuname** command prints the list of all systems that your system can directly connect with using **uucp**. On some systems, the number of machines that can be contacted using **uucp** may run into the hundreds. Each system is listed separately on a line, so you can determine the number of computers that your system can connect to directly by counting the lines returned by **uuname**. For instance,

```
$ uuname | wc -l
    2840
```

shows that the system on which this command line has been run (a computer at AT&T Laboratories) is directly connected to 2,840 other systems. Smaller computers are usually connected to fewer machines or sometimes only one.

To see if a specific system is known to yours, pipe the output of **uuname** to **grep** to search for that system name. This will print out the name of the system if it is in the **uuname** list, but will return nothing if it is not. For example,

```
$ uuname | grep chicago
$ uuname | grep chucky
chucky
```

shows that the current system knows about the system chucky but not the system *chicago.* Using **uuname** with the **–l** (*l*ocal) option prints out the name of your local system. For example,

```
$ uuname -l
jersey
```

means that the name of the system you are on is jersey, and more important, that it is known to other systems on the UUCP Network by that name.

Using the cu Command to Call a Remote System

The **cu** (*C*all *U*NIX) command allows you to log in and use a remote system from your home system. When you use **cu**, your system accesses the remote system using a dedicated communications line, a local area network, or a modem, depending on the configuration of certain files (described later in this chapter).

Making the Connection

When you use **cu**, you first call the remote system and make a connection with it. Remote systems known to your system can be called by name. You can also connect to a remote system by instructing your system how to make the connection.

You can call a system known to yours by name (listed in the **uuname** command output) using the **cu** command with the name of the system as an argument. You do not know how the systems are connected, since the file, */etc/uucp/Systems,* maintains this information. (Details on this file and how it is used are provided later in this chapter.) If there are many connections between systems, **cu** has the ability to choose among several media to establish the connection. For example, you can use **cu** to connect to the system alpha, known to your computer, using the following command:

```
$ cu alpha

Connected
login:
```

Using **cu** to contact the system alpha results in the *Systems* file being checked and a connection being set up according to the *Systems* file configuration. Once your system has successfully connected with alpha, you see the "Connected" message, followed by the normal "login:" prompt and login procedure on alpha. If you try to contact a system that is not included in the *Systems* file, you will get a message like this (depending on the particular version of UNIX you run):

```
$ cu beta
Connection failed:  SYSTEM NOT IN Systems FILE.
```

In new versions of UNIX (including those based on Release 4), the **cu** command has been enhanced to recognize eight-bit and multibyte characters, such as kanji characters.

Using cu with Telephone Numbers

If you specify a telephone number as an argument to **cu**, an *automatic calling unit (ACU)* listed in the file */etc/uucp/Devices* will be selected, and the system will dial the telephone number to be called.

A valid telephone number for **cu** is a string consisting of digits 0 through 9, the symbols * and # (from the telephone keypad), and the symbols = and –. The = symbol instructs **cu** to wait for a secondary dial tone before dialing the rest of the string; the – symbol introduces a pause (four seconds long) before dialing further.

Suppose you have to dial 9, wait for a dial tone for an outside line, and then dial the number 1 (201) 555-1234 to reach alpha. You would call alpha using the command

```
$  cu 9=12015551234
```

If you need to dial a *9 before reaching an outside line, use the following command:

```
$ cu "*9=12015551234"
```

The quotation marks are required so that the * is not interpreted by the shell.

If you dial a system but no connection can be made, you will get an error message. For instance,

```
$ cu 9=12015551234
Connect failed:  CALLER SCRIPT FAILED
```

If there is a problem with the ACU, the **cu** command may appear to hang.

Commonly Used cu Options for Connections The **cu** command supports several options for calling out to another system. Among these are the following:

–s *speed*	Specify the transmission speed in bits per second to be used for the connection. The default is "Any," in which the value will depend on the speed in the */etc/uucp/Devices* file (described later).
–c *type*	Specify the local area network to be used. The value of type is taken from the first field of the */etc/uucp/Devices* file. (In versions of UNIX based on Release 4, the *–c* option can also be used to specify a class of lines.)
–l *line*	Specify the device to be used as a communications line. The option overrides selection of devices from the *Devices* file.
–e	Set even parity.
–o	Set odd parity.
–h	Set half-duplex.
–b *n*	Set *n*-bit (7- or 8-bit) characters.
–n	Prompt for a telephone number. This provides additional user security, since the telephone number is not part of the command line and therefore is not displayed in response to a **ps –f** command.
–t	Call out to a terminal with an auto-answer modem (see also the **ct** command).

To call alpha using 28800 baud, dial 9 to get an outside line, and then 1 (201) 555-1234, using the following command:

```
$ cu -s28800 9=12015551234
```

To call out over a modem attached to */dev/term/04,* use the following command line:

```
$ cu -lterm/04 9=12015551234
```

Using Your cu Connection

After making a connection with the remote system, **cu** runs as two separate processes, a transmit process and a receive process.

The *transmit process* reads from your standard input (normally your keyboard) and passes this input to the remote system, except for lines that start with a tilde (~). The *receive process* accepts input from the remote system and, except for lines that start with a tilde, passes this input to your standard output. The special meaning of lines beginning with ~ will be described later.

Running Commands During a cu Session

Once you are logged in to the remote system, you can do anything normally possible on that system. For instance, suppose you log in from *jersey* to *nevada* by typing the following command on *jersey*:

```
$ cu nevada
Connected
login:
```

After successfully logging in by supplying your username and password, you can run commands on *nevada,* such as

```
$ who
npm        term/17        Feb  2 14:45
ddr        term/04        Feb  2 09:06
khr        term/01        Feb  2 12:07
greg       term/12        Feb  2 12:53
```

Commands Used with cu

One reason for using **cu** is that it supports simple ASCII file transfer, even with computers not running the UNIX System. (The parity and half-duplex options may be required for other systems; computers running the UNIX System normally do not require these.) As long as the remote system provides the **stty**, **echo**, and **cat** commands, **cu** can exchange files.

You use the **cu** command **~%take** to copy files from the remote system to your local computer. The general form of this command is

```
~%take there [here]
```

The preceding command "takes" (copies from the remote system) the file *there* and copies it to the file *here* on the local system. You use the **cu** command **~%put** to copy files from the local computer to the remote computer. The general form of this command is

```
~%put here [there]
```

This command "puts" (copies to the remote system) the file *here* from the local system with the name *there.* Note that after you type **~%**, **cu** will put the name of your local system between the ~ and the %.

For example, suppose you have logged in to nevada by running a **cu** session from jersey. You can take the file *memo* on nevada and copy it into the file named *newmemo* on jersey using the following command:

```
~[jersey]%take memo newmemo
```

Similarly, you can put the file named *data* from jersey, naming the copied file *data.new,* on nevada using the following command:

```
~[jersey]%put data data.new
```

By installing these **cu** commands under operating systems such as Windows/DOS, TSO, and GCOS, you can enable UNIX systems to communicate with a large variety of other computers.

cu Command Usage To run a command on the local system during a **cu** session, use the tilde sequence ~!*command.* To run the command locally but send its output to the remote system, use the tilde sequence ~$*command.* To send ~*line* (tilde line) to the remote machine, type ~~*line* (tilde tilde *line*).

You can temporarily escape from your **cu** to an interactive shell on the local system by typing ~! (tilde bang). (When you type the **!**, **cu** will put the name of the local system within brackets after the ~.)

After running your commands on the local system, you terminate this shell on the local machine as you normally would (using **exit**). After terminating this shell, you return to your session on the remote machine, whereupon **cu** echoes a ! (bang). For instance, suppose you have started a session on arizona by running a **cu** command on jersey:

```
$  ~[jersey]!
$ pwd
/var/home/khr
$ exit
!
```

Changing Directories To change directories on your local system during a **cu** session, use the tilde sequence **~%cd** followed by the name (or pathname) of a directory. (The name of the local system will be put after the tilde once the % has been typed.) This change will persist during your remote login session. For instance, suppose you have called nevada from the system jersey. Use the tilde sequence

```
~[jersey]%cd /home/khr/tools
```

to change to the directory */home/khr/tools* on jersey during the **cu** session.

You may be wondering why a special **cu** command is needed for changing directories on the local system. The reason is that the **cu** command **~!cd** does not change the directory on the local system. This fails because commands in **cu** are executed by a subshell, so that the change of directory does not persist after the subshell terminates.

Multiple cu Connections

Once you have used the **cu** command to log in to a second computer, you can use **cu** on that computer to log in to a third computer. For instance, while logged in to jersey you can use **cu** to log in to nevada. Once logged in to nevada, you can use **cu** to log in to arizona. You can execute the **uname** command on arizona, jersey, and nevada, respectively, as follows:

```
uname
arizona
~[jersey]!uname
jersey
~~[nevada]!uname
nevada
```

Terminating the Session

You terminate your session on the remote system by typing **~.** (tilde dot). When you type the . (dot), **cu** puts the name of the remote system within brackets after the tilde. For instance, to terminate a connection with jersey,

```
$  ~[jersey].
Disconnected
```

Additional cu Commands

Several other tilde sequences are recognized by **cu**, including these:

~%b	Send a break to the remote system.
~%d	Toggle debugging mode on and off.
~t	Print the values of the termio structure for the user's terminal.
~l	Print the values of the termio structure for the communication line.
~%ifc	Toggle between DC3/DC1 (CTRL-S/CTRL-Q) input flow control and no input flow control. This is useful when the remote system does not respond to flow control characters.
~%ofc	Toggle between DC3/DC1 (CTRL-S/CTRL-Q) output flow control and no output flow control. This allows flow control to be handled by the remote system.
~%old	Toggle to old-style (pre–Release 4) **cu** syntax for received diversions. This is used for receiving files on UNIX System V Release 4 from an earlier system.

The ct Command

The **ct** (*c*all *t*erminal) command is a convenient way to instruct your system to place a call to a terminal attached to an auto-answer modem. The command

```
$ ct -s1200 9=5551234
```

uses the same syntax as **cu** and instructs your system to use a 1200-baud line to call out, then dial a 9, wait for a secondary dial tone, and dial 555-1234.

The **ct** command is most often used with an **at** job to automatically call out to a terminal. For example, the script

```
at 8:00pm
ct -s1200 9=5551234
```

can be used to call your home terminal at 8:00 P.M., saving you the time and expense of calling in to the system.

Transferring Files Using uuto

Small ASCII files can most readily be sent as mail. However, it is awkward to receive large messages or files using **mail**. When a mail message is large, the entire mail message scrolls across the screen before the user has a chance to save or delete it. Long memos, manuscripts, or articles just cannot be handled this way. It is much easier to send a long message using a different facility and send a notification via **mail**.

The UUCP System provides several facilities for copying files between computers. The simplest way to use the UUCP System to send a file to a user on a remote machine is to use the **uuto** command. You specify the name of the file you are sending as the first argument to **uuto**, and you specify the recipient as the second argument, using bang-style addressing (described in Chapters 2 and 8).

The **uuto** command copies a file to a public directory. When you use the **uuto** command to send a file, the recipient receives notification via **mail** that the file was sent. For example, when you type

```
$ uuto memo jersey!fred
```

uuto transfers your file *memo* to the system jersey (known by your system and listed by **uuname**). This other system puts the file into a public directory in an area accessible by user *fred.* In Release 4, this public directory is */var/spool/uucppublic.* The copy of the file sent to you receives the absolute pathname

```
/var/spool/uucppublic/receive/fred/jersey/memo
```

The general form of the absolute pathname of a file sent to your system by **uuto** is

```
/var/spool/uucppublic/receive/user/system/filename
```

(In earlier versions of the UNIX System, the public files were usually placed in the directory */usr/spool/uucppublic*, but this directory may vary depending on the system.)

In the preceding example, you sent a file to the user fred on jersey, which is a system known to your machine. You can also route files through a series of intermediate systems when you know a path to a remote system (but this system is not known to your own system). You specify the path using bang-style addressing. For instance, the command

```
$ uuto data arizona!nevada!oregon!hawaii!yvette
```

sends the file *data* to the user *yvette* on *hawaii,* routing the file through the intermediate systems arizona, nevada, and *oregon,* in that order, before sending it to hawaii.

Although **uuto** is most often used for file copying between different computers, you can use **uuto** to send a file to another user on your local system. For instance, to send the file *memo* to the user *linda* on your system, you type

```
$ uuto memo !linda
```

You precede the logname of the user on your system with a bang (!).

You can also send directories using **uuto**. For instance, if *tools* is a directory, the following will send the subtree under *tools* to fred on jersey:

```
$ uuto tools jersey!fred
```

Options to uuto

When you use the **–m** option with **uuto,** mail is sent to you, the sender, when the copy is complete. For instance,

```
$ uuto -m report jersey!abby
```

sends your file report (on the local system arizona) to user abby on jersey and sends mail to you when this transfer is complete, as well as sending notification to the user abby. Your mail will look like

```
$ mail
>From uucp Sun Feb  4 00:59 EST 1999 remote from arizona
REQUEST: arizona!khr/report --> jersey!~
/receive/abby/arizona/ (abby) (SYSTEM jersey)   copy succeeded
```

The **uuto** command is based on the **uucp** command. Programs based on **uucp**, such as **uuto**, operate in a batch mode. The file being sent is not immediately copied when you issue the command, but may wait to be copied until its turn in the queue of UUCP System requests. If you **uuto** a file to someone and then delete it from your directory, nothing may get sent. By the time **uuto** attempts to copy the file, it is gone. In this case, it is a good idea to use the **–p** option to **uuto**. The command

```
$ uuto -p -m memo jersey!alvin
```

copies the file memo into the spool directory before it is transmitted to jersey.

Since it is a good idea to copy a file into the spool directory when you use **uuto**, and to have mail sent telling you that a file has been sent, you may want to alias the **uuto** command to **uuto –p –m**.

The public directory used by **uuto** and other **uucp** commands is just that, a public directory. Permissions are set so that all directories are writable (so that files can be created in them), and all files are readable (so that they can be retrieved). On most systems, the default permissions for files in uucppublic are set to rw–rw–rw–; that is, anyone can read or write the files. During the time files reside in the public directory, they are accessible to all other users who have access to your machine. If you need to exchange private files, use **crypt** (discussed in Chapter 12), or better, physically exchange a floppy disk or tape.

Moving Files Received via uuto with uupick

When someone has sent you files using **uuto**, you will receive mail telling you that you have received files. When you receive a file sent by **uuto**, you do not have to access this file using its long absolute pathname. Instead, you can use the **uupick** command to retrieve the file. The **uupick** command checks the public directory for files sent to you and then asks how these files should be dealt with.

For instance, running **uupick** may give

```
$ uupick
from system mozart: file fugue ?
```

which shows that a file named *fugue* was received from the system *mozart.* The question mark is a prompt indicating that **uupick** expects a command regarding the disposition of the file. One of the most common responses at the prompt is **m**, which moves the file to the current directory. The **uupick** command will tell you how many blocks were used for this file when it was moved. For example, if you enter **m** at the prompt shown in the last example,

```
m
4 blocks
```

you have moved the file *fugue* into the current directory, and **uupick** has told you that this file used four blocks. Similarly, the command

```
m /music
4 blocks
```

will move *fugue* into the directory */music.*

Once you have told **uupick** what to do with the file *fugue*, it will move to the next file received via a **uuto** command from a remote system, and so on. For instance, your **uupick** session may look like this, where the RETURN character typed at the end of each line is not shown:

```
$ uupick
from system mozart:  file fugue ? m /music
4 blocks
from system mozart:  file symphony ? m
38 blocks
from system mozart:  file opera ?
from system beethoven:  file symphony ?
from system vivaldi:  file concerti ? a
116 blocks
$
```

In this session, you first moved the file *fugue,* received from mozart, to your directory / *music.* Next you moved the file *symphony,* received from mozart, to your current directory. Next you decided not to do anything with the files *opera,* received from mozart, and *symphony,* received from *beethoven.* Finally, you moved all files, including the file *concerti,* received from the system *vivaldi,* to the current directory. You then received a prompt from the shell, since you ran through all the files you received from remote systems. The commands you can give **uupick** at its prompt are displayed in Table 2.

You may not want to see all the files sent to you via **uuto** from remote systems. Instead, you may only want to see files sent from a particular system. You can do this using the **–s** option to **uupick**. For instance, to see only those files sent to you via **uuto** from the system mozart, you use

```
$ uupick -s mozart
```

The uucp Command

The **uuto** command has been designed to make it convenient for a user to copy a file to a remote system, but it has limitations. The **uuto** command can only be used to transfer a file from your local machine to a remote machine, but the **uucp** command can be used to transfer files between two machines, where either or both machines can be remote.

You use the **uucp** command directly by issuing a command of the following form:

```
$ uucp source-file destination-file
```

Command	Action
RETURN	Go to next file; if no next file, return to shell.
m *dir*	Move the file to the named directory (default is the current directory).
a *dir*	Move all the files to the named directory.
d	Delete the file.
p	Print the file.
q	Quit.
!*command*	Escape to the shell and execute command.
*	Print **uupick** command summary.

TABLE 2 uupick Commands

The source file and the destination file may be located on your local machine or on remote machines known by your system. The source and destination files are specified by giving a system name, or the bang-style address of a system, followed by a bang and the full pathname of the source or destination file, for example, *jersey!var/home/ken/data.*

Transferring Local Files to Remote Systems

The most direct and the most common use of the **uucp** command is to transfer a local file to a remote machine. The remote machine must be known to your local machine. For instance, to send the file *memo* in your current directory to jersey, giving the file the pathname */var/home/fred/memo,* you can type

```
$ uucp memo jersey!/var/home/fred/memo
```

Note that this may not be allowed. UUCP System security is covered later in this chapter.

Transferring Remote Files to Your System

You can also use **uucp** when the source file is not located on your local machine, as long as the system where this file is located is known to your machine (listed by the **uuname** command). For instance, the command

```
$ uucp michigan!/var/home/donna/memo /var/home/fred/memo
```

copies the file */var/home/donna/memo* on *michigan* to your local machine, giving it the pathname */var/home/fred/memo.*

Transferring Files Between Remote Systems

You can also use **uucp** to transfer files between two remote systems. For example, the command

```
$ uucp michigan!/var/home/carol/memo  jersey!/var/home/fred/memo
```

sends the file */var/home/carol/memo* on michigan to jersey, giving it the pathname */var/home/fred/memo.*

Sending Files to Machines Not Directly Connected to Yours

You can use **uucp** to send files to a machine not directly connected to your machine, if you know a route that connects the two. For example, when you run the command

```
$ uucp memo alpha!beta!gamma!ferdie!/var/home/dan/memo
```

the **uucp** command contacts alpha, giving it the file *memo* and the path *beta!gamma!ferdie!/var/home/dan/memo.* The machine *beta* contacts *gamma* and gives it the file *memo* and the path *ferdie!/var/home/dan/memo.* The machine gamma contacts ferdie and attempts to put the file *memo* into */var/home/dan/memo.* Each machine only needs to know of the next machine in the chain.

Using Abbreviations for Paths

In specifying paths, you can use the full pathname of a file, or certain abbreviations. You can specify a file by preceding its name by ~/*user,* where *user* is the logname of its owner. For instance, in the command

```
$ uucp memo ferdie! ~jerry/memo
```

~jerry is expanded into the path of jerry's home directory. If you use the expression *~/user,* as in

```
$ uucp memo ferdie! ~/jerry/memo
```

uucp expands the *~/ into /var/spool/uucppublic* and puts *memo* into */var/spool/uucppublic/jerry/memo.*

Options to uucp

The **uucp** command supports a variety of options. You can receive notification that your request for a copy has been carried out by using the **–m** option. And if you use the **–n***user* option, mail will be sent to the user with logname *user* on the remote system. For example,

```
$ uucp -m -nfred memo jersey!/var/fred/memo
```

sends you mail when the copy is complete and sends mail to fred, on jersey, stating that a file has been sent to him.

A file transfer may not occur immediately. If you want to change or delete a file after you issue a **uucp** command, but you want the version of the file before you make changes to be sent, you can use the **–C** option to have **uucp** copy your file to the spool directory */var/spool/uucp.* Then the version of your file at the time you issued the **uucp** command will be sent. Table 3 summarizes some important actions of the **uucp** options.

Grades of uucp Service

Several grades of service may be available on your system for **uucp** commands. These grades of service receive different priorities from the UUCP System. You can use the **uuglist** command with the **–u** option to see which grades of service are available to users. There are three default grades (high, medium, and low); your system administrator can define and configure other grades.

To see what grades of service are available, type

```
$ uuglist -u
high
low
medium
```

Option	Action
–c	Do not copy file to spool directory (default).
–C	Copy file to spool directory before transfer.
–d	Make necessary directories for file copy (default).
–f	Do not make directories for file copy.
–g*grade*	Use grade of service specified.
–j	Output the **uucp** job ID string on standard output.
–m	Notify sender by mail when copy is complete.
–n	Notify recipient by mail that file was sent.

TABLE 3 Important uucp Options

This shows that the grades are the three default grades.

To transfer a file with a specified grade, use the **–g** option to **uucp**. For example,

```
$ uucp -ghigh memo jersey!/var/fred/memo
```

sets the grade of this **uucp** request to "high." In versions of UNIX based on Release 4, standard English names, such as "Smalljobs," assigned by the system administrator, can also be used as grades. Also, more than one interaction between computers is possible in versions of UNIX based on Release 4, as long as these interactions have different grades. This is helpful when an extremely large **uucp** job is tying up the connection, since smaller jobs that have been waiting can be sent using a second link.

The uustat Command

When you issue a **uucp** request, the job is entered in a queue. If you wish to see the status of any of your jobs (whether a job is still waiting in queue or is finished), use the **uustat** command. Without any arguments, **uustat** provides you with the status of all recent **uucp** commands you have issued, as shown here:

```
$ uustat
wongF3af4   10/23-22:06   S   wong   bill   2064 /home/fred/text
```

In the preceding example, the first field is the *job ID* of the request, the second is the date and time the request was issued, the next is an "S" or an "R" depending on whether the job *s*ent or *r*equested a file, the next field ("wong") is the system name where the file is to be sent, "bill" is the user ID of the user who requested the job, "2064" is the size of the file, and */home/fred/text* is the name of the file being transferred.

You use the **–j** option to **uucp** if you want **uucp** to tell you the job ID of a request.

Killing uucp Jobs The **uustat** command has several options. Among the more often used is the **–k** option used to kill existing queued jobs. To kill a job, use a command of the following form:

```
$ uustat -kjobid
```

For example, to kill the **uucp** job in the previous example, use the following command:

```
$ uustat -kwongF3af4
Job: wongF3af4 - successfully killed
```

Other uustat Options If you want to see how long the queues are, that is, how long it will take to send something to a remote system, use the **uustat** command with the **–t***system* and **–c** options. The **–t***system* option allows you to see the transfer rate or queue time for a specific system. The **–c** option checks queue time. For example,

```
$ uustat -twong -c
average queue time to wong for last 60 minutes:    0.07 minutes
data gathered from 01:20 to 02:20 GMT
```

To check the data transfer rate, use **–t***system* without the **–c** option. For example,

```
$ uustat -twong
average transfer rate with wong for last 60 minutes: 206200.00 bytes/sec
data gathered from 01:26 to 02:26 GMT
```

You will find other commands and options used to administer the UUCP System described later in this chapter.

Remote Execution Using the uux Command

The **uux** command (*U*NIX-to-*U*NIX e*x*ecution) can be used to collect various files from different computers, execute a command on a certain system (including a remote system) and send the output of the command to a specified system. Generally, you give the name of the remote system and the command you want executed. The UUCP System queues the request for the remote system, and when a connection is established, the command is sent.

In principle, virtually any command can be executed via **uux**. In practice, the commands you can run on a remote system are restricted for security reasons. Without this restriction, **uux** would provide a huge security hole if any command could be executed on any remote system. (For instance, you would not want a user on a remote system to call your system and remotely execute an **rm** * command.) On many systems, only commands associated with mail and news (such as the **rmail** command, used by systems when mail is sent, and the **rnews** command) can be executed with **uux**.

On UNIX systems based on Release 4, remote execution permissions are defined in */etc/uucp/Permissions.* On earlier systems, the *Permissions* file is in the path */usr/lib/uucp/Permissions.* On many systems, the *Permissions* file is not readable, which prevents anyone from determining what remote commands can be executed. Contact your system administrator to add a command to the list of allowable **uux** commands; consult the material later in this chapter to find how to add a command on your own system.

uux in Action

To see how **uux** works, assume that commands are enabled on both local and remote systems. If you use the command

```
$ uux "!cat jersey!/home/fred/file ohio!/home/bill/text > iowa!/home/bill/tmp"
```

uux is instructed to get one file from jersey *(/home/fred/file),* get one file from ohio *(/home/bill/text),* and concatenate them to a file on iowa *(/home/bill/tmp).*

You can also use **uux** to share facilities among machines. For example, if you have access to a system with a high-quality printer, you can print your file using the remote printer with the command

```
$ uux "jersey!lp -dlaser !memo"
```

which sends the file *memo* from the local system to be printed on jersey using the **lp** command with the **–d***laser* option.

Special shell characters such as >, <, ;, and | must be quoted in a **uux** command string. You can do this either by quoting the individual characters or, more easily, by quoting the

Option	Action
–	Use standard input to **uux** as standard input to command string.
–p	Same as **–**.
–a*job*	Use *job* as the job identifier.
–g*grade*	Assign *grade* specified.
–c	Do not copy the local file to the spool directory.
–C	Copy the local file to the spool directory before transfer.
–n	Do not notify user if command fails (useful in background jobs and daemons).
–z	Send notification to user if job succeeds.

Table 4 uux Command Options

entire command string. An expression of the form *!command* refers to the command on the local machine.

uux Options

The **uux** command takes several options. The most often used **uux** options are shown in Table 4.

UUCP System Administration

As discussed earlier, the UUCP System is used to communicate between computers. This section shows how to administer the UUCP System in its most common configuration, with your system connected to a modem over which telephone calls are made to other systems. You will learn how a file is passed through the UUCP queues. You will also learn about administering both the hardware and software used by UUCP: installing the needed modems and cables (hardware administration) and editing the associated databases. As the system is set up, you will also learn how often each step is performed. Finally, you will learn how to debug problems with the UUCP System.

UUCP Flow

When a file is transferred with the **uucp** command, or mail or netnews is queued with the **uux** command, the command verifies that the remote system exists. If it does, a control file is placed into the */var/spool/uucp* directory, along with the associated data files. (These files constitute a UUCP job.) **uux** and **uucp** then check to see if the program **uucico** (*U*NIX to *U*NIX *c*opy *i*n *c*opy *o*ut) is already connected to that system; if not, it is started. **uucico** attempts to connect to the system, using its control files to decide how to make the connection. If the connection is made, the files will be transferred, along with any other files queued up for that system. After a file is transferred, the control and data files for that job in */var/spool/uucp* are deleted. If **uucico** cannot make the connection, the control and data files will be left queued; once an hour, a **cron** job runs that will search */var/spool/uucp* and retry all connections for which jobs are queued. The **cron** program is used to execute other UUCP programs. In particular, if a job fails repeatedly for a week, a UUCP cleanup daemon will send a warning message to the user who queued the job, and eventually it will remove the files.

Installing UUCP

Before hooking up the modems, you have to install the software, a task you need to do only once. The UUCP software is part of the Basic Networking Utilities package and may or may not already be installed on your system. To check for the BNU package, look for the existence of the directories */usr/lib/uucp*, */etc/uucp*, */var/uucp*, */var/spool/uucp*, and */var/spool/uucppublic.* If the directories exist, and the files listed in the text that follows exist, then BNU is already installed. Otherwise, use the simple administration commands, **sysadm** or **pkgadd**, to install the package.

The UUCP Directories

The directories used by UUCP are displayed in Table 5.

The */usr/lib/uucp* Directory

This directory contains the administrative and internal programs used by the UUCP System, including **Uutry**, a program that attempts to establish a connection to another system; **uudemon.hour**, **uudemon.cleanup**, **uudemon.admin**, and **uudemon.poll**, programs executed out of the **uucp** *cron* file; **uucheck**, a program that checks the databases for consistency and looks for problems; **uusched**, **uuxqt**, and **uucico**, which do the actual work for UUCP.

uudemon.hour looks for systems that have traffic queued and for commands received from other systems to execute. **uudemon.cleanup** returns traffic that has been queued too long and cleans up the log files and directories. **uudemon.poll** checks for systems that are supposed to be contacted regularly. **uudemon.admin** runs the **uustat** command and mails the results to the *uucp* administrative login. Neither **uudemon.cleanup** nor **uudemon.admin** is run by default.

uusched looks through the UUCP queues and starts up a connection to the systems for which jobs are stored. **uuxqt** executes the jobs that have come into the system, such as mail. **uucico** sets up the connection between two systems and transfers the files. In addition, it logs the results and (optionally) notifies users by mail regarding the success or failure of the transfers.

The */var/spool/uucp* Directory

This directory contains a directory for each system to which traffic is queued.

/usr/lib/uucp	Administrative and internal programs
/etc/uucp	Administerable files
/var/uucp	Log and status files
/var/spool/uucp	Queued traffic
/var/spool/uucppublic	Standard location to place files

Table 5 UUCP Directories

The */var/uucp* Directory

This directory contains a series of directories, all with names beginning with a dot (.), into which various log files and status files are placed.

The */var/spool/uucppublic* Directory

Transferring files to this directory is usually permitted, but to nowhere else. Some systems also permit files to be transferred from this directory.

The */etc/uucp* Directory

This directory contains all of the administerable files for UUCP. They will each be dealt with later in this chapter. These files are

Config
Devconfig
Devices
Dialcodes
Dialers
Grades
Limits
Permissions
Poll
Systems
Sysfiles

UUCP Commands in */usr/bin*

In addition to the UUCP commands already described, **uulog**, a command that displays the log file for a given system, is found in */usr/bin.*

Setting Up UUCP: An Example

Let's follow the steps necessary to set up a pair of modems so that they can be used for UUCP traffic between your system, *foocorp,* and the systems *abcinc* and *xyzinc.* One modem will be set up for incoming traffic and the other, for outgoing traffic. (Setting up a single modem to be used for both incoming and outgoing UUCP traffic will be described later.) The tty lines to be used for the two modems are */dev/term/21* and */dev/term/22.*

The *Permissions* File

This file describes the access rights for other systems when they call your machine, and when your machine calls them. The access rights are listed in a series of *NAME=value* fields, which may be continued onto multiple lines by placing a backslash at the end of the lines. The rights given when your system calls another system are given by lines that have a MACHINE= field. A reasonably secure entry permits the other system to receive files from */var/spool/uucppublic* (READ=), write files into */var/spool/uucppublic* (WRITE=), and send mail and netnews to your system (COMMANDS=). You should usually disallow other systems from requesting arbitrary files (REQUEST=). The following entry gives permission to receive and write files to */var/spool/uucppublic* and disallows other systems from requesting arbitrary files:

```
MACHINE=OTHER \
    READ=/var/spool/uucppublic WRITE=/var/spool/uucppublic \
    COMMANDS=rmail:rnews \
    REQUEST=no
```

When another system calls your system, the rights given are indicated by lines that have a LOGNAME= field. A reasonably secure entry permits other systems to call your system using the nuucp login, read files from */var/spool/uucppublic* (READ=), write files into */var/spool/uucppublic* (WRITE=), and send mail and netnews to your system (COMMANDS=). If your system has files queued up, your system can call the other system back to deliver the files (SENDFILES=). (This prevents other systems from pretending to be systems that they are not and receiving files intended for someone else.) For example,

```
LOGNAME=nuucp \
    READ=/var/spool/uucppublic WRITE=/var/spool/uucppublic \
    COMMANDS=rmail:rnews \
    REQUEST=no SENDFILES=call
```

The *Devices* File

You edit this file when adding new **uucp** tty lines. It lists the devices present on your system and specifies how to manipulate those devices to get through to the hardware connected to the devices.

For each tty line you want to be used by UUCP, you will want to add two lines to the *Devices* file, one line that starts with the letters "ACU" and one line that starts with the word "Direct." The ACU lines specify the type of modem being used. The Direct lines are used by the **cu** command. Both types of lines also specify the tty lines and the speeds permitted on those lines.

Because this example is using ttys 21 and 22 with modems that we assume can run at 28800, 24000, 21600, and 14400 bits per second, the entries will look like this:

```
ACU    term/21 - 24000 att2224a
ACU    term/21 - 14400 att2224a
ACU    term/21 - 28800 att2224a
Direct term/21 - 24000 direct
Direct term/21 - 28800 direct
Direct term/21 -   direct
ACU    term/22 - 24000 att2224a
ACU    term/22 - 21600 att2224a
ACU    term/22 - 14400 att2224a
Direct term/22 - 28800 direct
Direct term/22 - 24000 direct
Direct term/22 - 28800 direct
```

The *Dialers* File

You edit this file when adding a new type of modem. This file describes how to dial a phone number on the different types of modems attached to the system. Because this example uses modems that are already described within the file, there is no need to modify the file here.

If a new type of modem were to be added to the system, a description of the modem would be entered here. Each modem description consists of

- A field that describes what characters must be sent to the modem to make it pause or to wait for a secondary dial tone.

- Several fields that make up a *chat script.* Chat scripts will be described in detail later in this chapter; they describe what strings must to be sent to the modem to dial the phone number, and the expected responses from the modem.

Further descriptions of the fields within the file may be found in the *System Administrator's Reference Manual.*

The *Systems* File

You edit the *Systems* file when you add a system to be contacted. This file describes the systems that are known to UUCP and how to connect to each system. For each system, the type of device to be used is described, and the speed to use on that device, the phone number to dial, and a chat script that describes how to connect to the remote system are provided.

These are the fields within the file used here:

- The name of the system
- The times to call this system, usually "Any" or "Evening"
- The device type, in this case, "ACU"
- The speed to use when calling that system, such as 28800 baud
- The phone number
- The chat script

The phone number may include names from the *Dialcodes* file (described later in this chapter) as well as the characters = and –, which specify waiting for a secondary dial tone and a pause.

Creating a Chat Script The easiest method of figuring out what goes into the chat script is to try it using the **cu** command. The usual sequence is to wait for the login prompt, send a login name, wait for the password prompt, and send the password.

On a piece of paper, create two columns, one for what you want to receive from the remote system and the other for your response. You type the command this way (assume the phone number is 555-1234):

```
cu -lterm/21 5551234
```

Once the connection is made to the remote system, write down what you see and what you have to respond with, as in the following list:

You See	Your Response
nothing	RETURN
login:	nuucp
password:	*xyzzy*

Now record these responses into a chat script. It is common to record only the last few characters of what to look for because communications are often garbled for the first few seconds of transmission until things settle down.

One way to write a chat script for this example is shown here:

```
"" "" in: nuucp word: xyzzy
```

The first two sets of quotes (" ") mean to look for nothing and then send nothing (followed by a RETURN).

Another common way of writing the same thing is shown here:

```
in:--in: nuucp word: xyzzy
```

This says to look for the characters "in:" If they are not found within a few seconds, press RETURN and look again.

Typical **uucp** entries for the two systems you wish to contact, using some made-up phone numbers and passwords, look like the following:

```
abcinc Any ACU 24000 5551212 in:--in: nuucp word: xyzzy
xyzinc Any ACU 24000 5551234 in:--in: nuucp word: xyzzy
```

Using Other Networks Another common setup is to run UUCP across another network, such as TCP/IP. In the case of TCP/IP, the *Device* field would indicate that the connection is to be made via TCP, and the phone number would instead indicate the Ethernet address, as in the following example:

```
pixie Any TCP - \x0009090990099999
```

The *Dialcodes* File

This file describes symbolic names for phone number prefixes that can be used within the *Systems* file. This permits numbers to be shortened or a common *Systems* file to be shared by multiple machines in different geographical areas by changing only the *Dialcodes* file.

For example, if you connect to many systems in New York City or in Chicago, you may wish to use symbolic names in the *Systems* file:

```
abcinc Any ACU 24000 chicago5551212 in:--in: nuucp word: xyzzy
xycinc Any ACU 24000 newyorkcity 5551234 in:--in: nuucp word: xyzzy
```

The entries within the *Dialcodes* file consist of two fields—the symbolic name and the phone number the name represents. The preceding example requires the following two entries:

```
chicago 1312
newyorkcity 1212
```

The *Sysfiles* File

The *Systems*, *Devices*, and *Dialers* files can actually consist of multiple files. The default name used for each type of file was listed earlier; a list of files may also be given in the *Sysfiles* file for each of the preceding files, similar to how the *$PATH* environment variable gives a list of directories.

For example, you may wish to share a common *Systems* file between multiple systems but still have a local *Systems* file *(Sys.local)* on each machine. This would be specified in *Sysfiles* with the following entries:

```
service=uucico systems=Sys.local:Systems
service=cu     systems=Sys.local:Systems
```

Now when the UUCP programs need to look for system information, the file */etc/uucp/Sys.local* will be looked at before */etc/uucp/Systems.*

The *Config* File

This file allows you to override some parameters used by the different protocols supported by UUCP. The defaults are almost always sufficient. For further information, see the *System Administrator's Reference Manual.*

The *Devconfig* File

This file allows you to override some parameters used by devices other than modems. As for the *Config* file, the defaults for the *Devconfig* are almost always sufficient. Further information can be obtained in the *System Administrator's Reference Manual.*

The *Grades* File

This file permits jobs to be partitioned into multiple queues of different priorities. The defaults provide for three priority grades, which is usually sufficient. For further information, see the *System Administrator's Reference Manual.*

The *Limits* File

This file allows you to set the maximum number of **uucico**, **uusched**, and **uuxqt** processes that are permitted to run simultaneously.

The *Poll* File

You edit this file when setting up polled sites. This file is used by **uudemon.poll** to determine the times at which a system will be polled. This is useful for systems that cannot call your system for some reason, but that need to be checked regularly to pick up any mail or netnews waiting to be transferred. For further information, see the *System Administrator's Reference Manual.*

Adding Cleanup and Administration Scripts to cron

The cleanup script is usually run once each day late at night. This is done by adding the following line to the *crontab* file for root:

```
45 23 * * * ulimit 5000; /usr/bin/su uucp -c "/etc/uucp/uudemon.cleanup" >
/dev/null 2>&1
```

This increases the maximum file size to 5,000 blocks and then runs the cleanup script as **uucp**.

The admin script is usually run around the same time each day, using an entry in **uucp**'s *crontab* file.

```
55 23 * * * /etc/uucp/uudemon.admin > /dev/null 2>&1
```

Setting Up UUCP Logins

The UNIX System is usually distributed with two UUCP logins: uucp and nuucp. The uucp login owns all of the UUCP commands and files. It is never logged in to, and its password should be locked. The entry for uucp in */etc/passwd* should look like this:

```
uucp:x:5:5:0000-uucp(0000):/usr/lib/uucp:
```

The nuucp login is used by other systems to log in. The entry for nuucp in */etc/passwd* should look like this:

```
nuucp:x:10:10:0000-uucp(0000):/var/spool/uucppublic:/usr/lib/uucp/uucico
```

Note that this entry does *not* have the same user ID as the uucp login, and the shell for nuucp is */usr/lib/uucp/uucico.* The password you assign to nuucp will be advertised along with the rest of the information other system administrators need to set their systems up to reach your system.

Debugging UUCP Problems

You should be aware of the various programs available for debugging with UUCP. Some of the programs are appropriate for double-checking your work after you have just finished making modifications to the administerable files, others let you monitor active connections to other systems, and still others let you make quick checks to see if there are any problems.

Using uucheck

Whenever you are finished editing any of the files in */etc/uucp,* check them by running this:

```
uucheck -v
```

The following is sample output from a successful run of **uucheck** using the *Permissions* file shown earlier:

```
*** uucheck:  Check Required Files and Directories
*** uucheck:  Directories Check Complete

*** uucheck:  Check /etc/uucp/Permissions file
** LOGNAME PHASE (when they call us)

When a system logs in as: (nuucp)
        We DO NOT allow them to request files.
        We WILL NOT send files queued for them on this call.
        They can send files to
            /var/spool/uucppublic
        Sent files will be created in /var/spool/uucp
         before they are copied to the target directory.
        Myname for the conversation will be foobar.
        PUBDIR for the conversation will be /var/spool/uucppublic.

** MACHINE PHASE (when we call or execute their uux requests)

When we call system(s): (OTHER)

        We DO NOT allow them to request files.
        They can send files to
            /var/spool/uucppublic
        Sent files will be created in /var/spool/uucp
         before they are copied to the target directory.
        Myname for the conversation will be foobar.
        PUBDIR for the conversation will be /var/spool/uucppublic.

Machine(s): (OTHER)
```

```
CAN execute the following commands:
command (rmail), fullname (rmail)
command (rnews), fullname (rnews)

*** uucheck:  /etc/uucp/Permissions Check Complete
```

Listing the Systems Your Machine Can Contact

After editing */etc/uucp/Sysfiles*, */etc/uucp/Systems*, or any other file listed in */etc/uucp/Sysfiles*, you can use **uuname** to list out the systems your machine can contact. Type

```
# uuname
```

and you will see a list similar to this:

```
foobar
george
thomas
```

Displaying the UUCP Log File

The log files keep information on all phases of the conversations between two machines. To display the contents of the log file for a given system, use a command of the form

```
# uulog system
```

For example, the following **uulog** output shows two successful connections between the machines foobar and george; the first conversation was initiated locally on foobar, and the second conversation was initiated remotely by george:

```
uucp george (1/2-23:04:22,7730,0) SUCCEEDED (call to george - process job grade Z )
uucp george (1/2-23:04:26,7730,0) OK (startup)
root george georgeZ683c (1/2-23:04:29,7730,0) REQUEST (foobar!D.fooba686619b -->
george!D.fooba686619b (root))
root george georgeZ683c (1/2-23:04:32,7730,1) REQUEST (foobar!D.georg683cfd5 -->
george!X.georgeA683c (root))
uucp george (1/2-23:04:34,7730,2) OK (conversation complete tcp 21)
uucp george (1/2-23:05:30,7733,0) OK (startup)
uucp george (1/2-23:05:30,7733,0) REMOTE REQUESTED (george!D.georg46ba823 -->
foobar!D.georg46ba823 (hansen))
uucp george (1/2-23:05:32,7733,1) REMOTE REQUESTED (george!D.fooba4a7565d -->
foobar!X.foobarA4a75 (hansen))
uucp george (1/2-23:05:33,7733,2) OK (conversation complete notty 7)
```

The **uulog** command also permits you to check on the commands executed by the remote machines. To see these logs, use the **–x** option, as shown here:

```
# uulog -x george
```

You'll see the following output, indicating that the job seen in the last log was really a mail message being sent from george!root to foobar!root:

```
uucp george  (1/2-23:05:36,7734,0) george!root XQT (PATH=/usr/bin  LOGNAME=uucp
UU_MACHINE=george UU_USER=george!root export UU_MACHINE UU_USER PATH; rmail root )
```

Checking Connections Using uustat

To check if jobs are getting through, type this:

```
# uustat -q
```

This prints a list of all systems currently connected and those that could not be contacted, along with the last known reason for why they could not be contacted. For example, the following shows an active connection with george, a job to james that had to be postponed until a later time, and a problem contacting jesse because the remote side answered with the wrong name (and an indication as to when UUCP will try again):

```
george     01/02-23:36 TALKING
james      01/02-23:43 WRONG TIME TO CALL
jesse      01/02-23:47 WRONG MACHINE NAME Retry: 0:05
```

Watching a Live UUCP Connection with Uutry

Sometimes a job will just sit in the queue, and **uustat** won't give you sufficient information. When this happens, your best choice is to watch a live connection attempt using **Uutry**. The most common problems with connecting to a system are because of bad phone number, login, or password information in the *Systems* entry for that system. Using **Uutry** will usually point this out. Use a command line of the form

```
# /usr/lib/uucp/Uutry -r system
```

This starts the **uucico** program with debugging information redirected to the file */tmp/system* and then runs **tail –f** on that file. (Type your interrupt character to stop the **tail** command.)

The following shows a successful connection being made to james:

```
mchFind called (james)
name (james) not found; return FAIL
attempting to open /usr/spool/uucp/.Admin/account
Job grade to process -
conn(james)
Device Type ACU wanted
set interface ACU
processdev: calling setdevcfg(uucico, ACU)
gdial(2224) called
expect: ("")
got it
sendthem (MM)
expect: (:)
MJJDATAPHONE II Automatic CallerMJ2400 bps MJJDial, Enter Command Or H For
HelpMJ:got it
sendthem (w0M)
expect: (:)
w0MJMInvalid CommandMJ:got it
sendthem (9+18005555555M)
expect: (ered)
9+18005555555MJMJDialingMJ9+18005555555MJRinging.....MJGAnsweredgot it
```

```
getto ret 6
expect: ("")
got it
sendthem (MDELAY
MM)
expect: (gin:)
  Mlogin:got it
sendthem (nuucpM)
Login Successful: System=james
msg-ROK
  Rmtname james,  Version 'unknown',  Restart NO, Role MASTER,  Ifn - 6,
Loginuser - root
rmesg - 'P' got Pgx
wmesg 'U'g
Proto started g
*** TOP ***  -  Role=1, wmesg 'H'
rmesg - 'H' got HY
PROCESS: msg - HY
HUP:
wmesg 'H'Y
cntrl - 0
send OO 0,exit code 0
Conversation Complete: Status SUCCEEDED
```

Seeing the File Being Transmitted

Sometimes you may have to queue a job without having **uucico** automatically started so that you can see the file being transmitted using **Uutry**. This is accomplished by using the **–r** option on the **uucp** command, in a command line of the form

```
$ uucp -r file system! /file
```

UUCP Security

The version of UUCP that comes with UNIX System V Release 4 is considerably more secure than previous versions of UUCP. As distributed, UUCP comes as a restricted system that doesn't allow anything to be performed, and the purpose of most of the administration described in this section is to open up the system in an organized manner. The setup procedure described in this chapter opens the system the minimum amount necessary to provide a usable system, while providing reasonable security. Be particularly aware of the following items while administering your UUCP system:

- Always make certain that your *Systems* and *Permissions* files are mode 400 and owned by the login uucp.
- The *Permissions* file is the outside world's gateway to your machine; be very careful when you make changes to it. For example, do not ever specify COMMAND=ANY within the *Permissions* file.
- Check your log files frequently; in particular, look for attempts to access system files such as */etc/passwd.*
- Always remember to use **uucheck** after you finish modifying any of the UUCP files.

Administering the Mail System

Your UNIX System comes already configured to send mail between users on your machine. Once you have UUCP configured, as described earlier in this chapter, electronic mail will automatically work to all of the machines defined in the UUCP databases. So what has to be administered in the mail system if everything already works? The first part of the answer is that all mail system administration is optional. The second part of the answer is that

- You can create *mail aliases* (names to which you can mail that translate into one or more other names).
- You can configure mail to use the Simple Mail Transfer Protocol (SMTP), which is a protocol primarily used to exchange mail across TCP/IP networks.
- You can control to whom mail may or may not be sent.
- You can add logging of mail traffic.
- You can establish a connection to a *smarter host* (another system that knows how to connect to more systems than your local machine).
- You can establish a *domain name* for your system.
- You can configure a set of machines as a *cluster* (a set of machines that all appear to have the same name when they send mail).
- You can share the mail directory between multiple machines using a Distributed File System.

The Mail Directories and Files

All administration of mail is done by editing files that are found under the directory */etc/mail*. Other programs and files you should be aware of will be found under */usr/lib/mail* and */usr/share/lib/mail*; these normally will not need to be touched. These are the primary files you need to modify under */etc/mail*:

- */etc/mail/mailsurr* is the *mail surrogate* file. You can use it to control how login names are interpreted, which login and system names are permitted to receive mail, how mail is delivered (such as via UUCP), and any postprocessing to be performed after mail is successfully delivered. The surrogate file will be discussed in detail later in this chapter, in the section "Mail Surrogates."
- */etc/mail/mailcnfg* is the *mail configuration* file. You can edit it to set several optional parameters to control how mail works. These options will be discussed in detail later in this chapter.
- */etc/mail/namefiles* contains a list of files and directories that contain aliases. The default file contains one filename *(/etc/mail/names)* and one directory name *(/etc/mail/lists).*
- */etc/mail/mailx.rc* contains settings to be used by all invocations of the **mailx** command. An example will be given later in this chapter.

Mail Aliases

A *mail alias* is a name that is translated into one or more other names by the mail command while the mail is being delivered. For example, if you have users on your machine who can be grouped together, such as a class of students or a work group, you can create an alias for the group. You can then send mail to the single alias name and the mail will be delivered to all of the users on the list. To create a mail alias, such as "cs101," you would add cs101 to */etc/mail/names* and list all of the login names of the students in the class, like this:

```
cs101    sonya george nancy tom linda sam
```

The line can be added anywhere within the file */etc/mail/names.*

If the list of user names becomes too long to fit onto a single line, the list may be continued onto additional lines by placing a backslash (\) at the end of each line that needs to be continued, like this:

```
cs101    sonya george nancy \
         tom linda sam
```

An alternative to listing the names in */etc/mail/names* is to place each alias into its own file under */etc/mail/lists.* This has two advantages: The search time to find the alias is reduced, and the ownership of the alias file can be given away. This way you can let someone else, such as the teacher of the class or a secretary, do the administrative work on that file.

The program **mailalias** is used to translate mail names. It may be used to verify that an address has been entered into */etc/mail/names* or */etc/mail/lists* properly by executing it with an alias as its argument:

```
$ mailalias cs101
sonya george nancy tom linda sam
$
```

Mail Surrogates

The file */etc/mail/mailsurr* contains the instructions to the **mail** command on how to translate mail addresses and deliver messages to remote sites using UUCP. It can do other things as well, such as indicate how to deliver messages using SMTP, do postprocessing after successfully delivering the mail message, and control who is permitted to send and receive mail. These capabilities will be discussed later in this chapter.

The Format of the Surrogate File

The surrogate file, */etc/mail/mailsurr,* contains a series of instructions consisting of two regular expressions (one each to match the sender and recipient of the mail message) and a command. The regular expressions and commands are surrounded by single quotes (') and separated by blanks or tabs.

The command is one of the following:

- Accept
- Deny
- Translate R= translation

- Translate R=| command
- < exit-codes; delivery UNIX command
- > postprocessing UNIX command

The regular expressions are the same as the regular expressions used within **ed**, with the additions of the **egrep** operators ? and + and the use of ( ) in place of \(\). For example, the regular expression

```
.+
```

matches *any* mail address.

As another example, here is a regular expression that matches an address of a user on another system:

```
 ([!]+)!(.+)
```

This regular expression looks for one or more characters (which are not exclamation points), followed by an exclamation point, and then one or more additional characters. Everything up to that exclamation point is returned as a subexpression, and everything after the exclamation point is returned as a second subexpression.

Combining these with the **uux** command tells the **mail** command how to deliver mail to users on other machines, giving the following command line for the surrogate file:

```
'.+'  '([!]+)!(.+)'  '< /usr/bin/uux - 1!rmail 2'
```

This line matches all senders, '.+', and all recipients that contain an exclamation point, '([!]+)!(.+)'. When a match is found, the **uux** command is executed with pieces of the recipient's address pulled into the command line, just as is done with substitution commands in **ed**.

That is, the recipient's address *funny!george!thomas* will be matched and the **uux** command executed will be this:

```
/usr/bin/uux - funny!rmail george!thomas
```

(The command **rmail** is the *r*estricted *mail* program used for network mail.)

Deny Commands

Deny commands are used to specify sender/recipient address combinations that are not permitted to send or receive mail. For example, you may wish to prevent some restricted users from sending mail anywhere. This example prevents the user *andrew* from sending mail:

```
'andrew'  '.+'  'Deny'
```

As another example, mail is not permitted to be sent to an address that contains a shell metacharacter as part of the address. To express this, you will find the following **Deny** instruction in the mail surrogate file:

```
'.+'  '.+[<>()|;&].+'  'Deny'
```

Accept Commands

You may want to connect your system to an external service for which you must pay money, such as a Telefax service or a commercial mail service such as AT&T Mail. For these services, you may want to restrict mail going to those systems to local users and not permit any remote systems to send mail to those services through your system. It is easy to express this by using a combination of the **Accept** command with a subsequent **Deny** command. The first step is to state that local users (the sender's address will not contain an exclamation point) are permitted to send mail to the service, like this:

```
'[!]+'  'telefax!.+'  'Accept'
```

Next state that everyone else should be denied access to the service:

```
'.+'  'telefax!.+'  'Deny'
```

Even though this instruction says that all senders, '.+', should be denied access to the Telefax service, the earlier presence of an **Accept** instruction overrides the subsequent **Deny** instruction.

Translate Commands

The **Translate** command specifies how one address should be converted into another address, such as for alias processing. The command line that does this inside the surrogate file looks like this:

```
'.+'  '[!]+'  'Translate R=|/usr/bin/mailalias %n'
```

All local names ('[!]+' matches any address that does not have an exclamation point) are passed through the **mailalias** for possible translation. (This example also shows the use of *%keyletters,* which will be explored further later in this chapter. Here *%n* expands to the recipient's name.)

The **Translate** command may also be used without external commands. For example, both bang-style addresses (those with an exclamation point in them, as in *funny!george!thomas*) and domain-style addresses (those with @ signs in them, as in *thomas@george.com*) are supported by the **mail** command. The domain-style addresses are converted into bang-style addresses through this translation command within the surrogate file:

```
'.+'  '(.+)@([^@]+)'  'Translate R=\\2!\\1'
```

Postprocessing Commands

The postprocessing commands are executed for each successfully delivered message. This can be useful for logging mail messages or running delivery notification programs. For example, suppose you want to log all mail messages successfully delivered locally in one log file, and you want to log all mail messages passing through the system in another log file. The following shell script could be installed in */usr/lib/mail/surrcmd* (the normal location for surrogate commands to be placed):

```
# logmail
log=$1
shift
echo `date` $* >> /var/mail/$log
```

and the following lines added to the surrogate file:

```
'.+'  '[!]+'  '> /usr/lib/mail/surrcmd/logmail :loclog %R %n'
'.+!.+'  '.+!.+'  '> /usr/lib/mail/surrcmd/logmail :thrulog %R %n'
```

In this way, a line will be added to the file */var/mail/:loclog* for every mail message successfully delivered locally, and a line added to */var/mail/:thrulog* for every mail message passing through the system. (%*R* is replaced with the sender's return path.)

%keyletters

You have already seen examples of %*keyletters*, %*n* and %*R*. There are about a dozen %*keyletters* that can be used in surrogate instructions. All %*keyletters* can be used in the command fields; a subset may also be used as part of regular expressions. For example, the local system name is automatically stripped off of addresses using %*L* in the following **Translate** command found in the surrogate file:

```
'.+'  '%L!(.+)'  'Translate R=1'
```

The complete set of %*keyletters* is documented on the **mailsurr** manual page, which can be found in the *System Administrator's Reference Manual.*

Debugging the Surrogate File

Before installing a new surrogate file, you should check your modifications. This can be done using the **–T** option to **mail**. The **–T** option will provide considerable output showing how the surrogate file is parsed, and then showing how a given mail address will be treated by the surrogate file. For example,

```
# echo foo | mail -T nsurr test!address
```

tests the new surrogate file *nsurr* with the address *test!address.* The mail message will not actually be sent when testing with the **–T** option.

If you find that you are having problems with a surrogate file that is already in place, a similar test may be run by using the **–x** option, as shown here:

```
# echo foo | mail -x 3 test!address
```

This creates a file under */tmp* (named */tmp/MLDBG**) while the message is being delivered. The value used with **–x** (here 3) determines how much tracing output will be produced: The higher the number, the more output. If the debug level is positive, the file will be automatically removed once the message is successfully delivered. If the debug level is negative, the file will be left there for your perusal.

The Mail Configuration File

The file */etc/mail/mailcnfg* contains several optional parameters that may be set for the mail command. It consists of a list of variable names and their values, separated with an equal sign (=).

The *DEBUG* Variable

The *DEBUG* variable may be given a value in */etc/mail/mailcnfg*:

```
DEBUG=-99
```

This variable gives a default value for the **–x** option. Assigning this value to *DEBUG* causes all invocations of the **mail** command to execute as if you had run them with the **–x –99** option.

Other configuration variables will be discussed later.

Adding New %keyletters

Sometimes you may wish to use a long string in several places within the surrogate file without having to retype the string each time. It is possible to introduce new *%keyletter*s to be used in the surrogate file by defining them in the mail configuration file, like this:

```
d=/usr/lib/mail/surrcmd
```

Any lowercase variables (that do not already have a predefined value) can be defined like this in the mail configuration file and can be referenced with the corresponding *%keyletter* within the surrogate file. For example, the definition of *%d* shown in the preceding example can be used to refer to the **logmail** command shown earlier:

```
'.+!.+'  '.+!.+'  '> %d/logmail :thrulog %R %n'
```

Smarter Hosts

A *smarter host* is defined as another system to which you can send remote mail when your system does not know how to send the mail. For example, you might have one machine named *brainy* that has numerous systems defined within its UUCP databases, while your other machines only have the local systems defined. The machine brainy would be set up as your smarter host. The first step is to define the variable *SMARTERHOST* in the configuration file:

```
SMARTERHOST=brainy
```

In the surrogate file, you will find a commented-out line at the end that looks like this:

```
#'.+'  '.+!.+'  'Translate R=%X!%n'
```

To enable the smarter host, just remove the #.

Mail Clusters

It may be useful to configure a set of machines so that they all appear as if they were a single machine to anyone receiving mail from any of them. For example, you might have a bunch of workstations at your company named *company1* through *company10,* but no one outside of your company needs to know that any machine name other than *company* exists.

To set up a mail cluster requires two steps. The first step is to identify the name of your machines used for sending mail. This is done in the mail configuration file:

```
CLUSTER=company
```

The second, optional, step is to set the name that UUCP uses to identify itself to other systems. This is done using the *MYNAME* setting in the UUCP */var/uucp/Permissions* file:

```
MYNAME=company
```

Networked Mail Directories

Another configuration you may find useful in a closely coupled environment is to use a Distributed File System, such as RFS or NFS, to share the directory */var/mail* between multiple machines. In this way, the mail is stored on only one file system. If you use the coupling to give redundancy of computing power, and if you have a way of mounting file systems from the machine even if the rest of the machine is down, you will want to be able to access your mail even if the machine breaks down.

First, decide which machine will be the primary machine, the one that will normally have the mail file system mounted, such as company1. Second, move all mail currently found on the secondary machines to the primary machine. Next, remove the directory */var/mail/:saved* from all of the secondary machines. (This directory is normally used as a staging area when **mail** is rewriting mail files.) Next, tell **mail** where it should forward the mail message if it finds that the */var/mail* directory is not mounted properly. Do this by adding the following variable to the mail configuration file:

```
FAILSAFE=company1
```

Finally, mount the mail directory from the primary machine using either RFS or NFS.

Setting Up SMTP

SMTP is a protocol specified for hosts connected to the Internet that is used to transmit mail. SMTP is used to transfer mail messages from your machine to another across a link created using a network protocol such as TCP/IP. The use of SMTP is an alternative to using UUCP—of particular use to non-UNIX systems that do not have UUCP or to systems that are located in another *domain.*

Mail Domains

The most commonly used method of addressing remote users on other computers is by specifying the list of machines that the mail message must pass through in order to reach the user. This is often referred to as a *route-based mail system,* because you have to specify the route to use to get to the user as well as the user's address.

Another method of addressing people is to use what are known as *domain-based mail addresses.* In a domain-based mail system, your machine becomes a member of a *domain.* Every country has a high-level domain named after the country; there are also high-level domains set aside for educational and commercial entities. An example of a domain address is *usermachine.company.com,* or equivalently, *machine.company.com!user.* Anyone properly registered can send mail to your machine if they know how to get directly to your machine or know the address of another smarter host (commonly referred to as the gateway machine) that does have further information on how to get to your machine; this may require the use of other machines on the way. This cannot be done unless your machine is registered with the smarter host, and you have administered the gateway machine on your system as the smarter host. If you have SMTP configured, your system may be able to directly access other systems in other domains.

Once you have registered your machine within a domain, you must set the domain on your system. This can be done in several ways.

- If your domain name is the same as the Secure RPC domain name, then both can be set by using the **/usr/bin/domainname** program, using a line of the form

  ```
  domainname.company.com
  ```

- If you have a nameserver, either on your system or accessible via TCP/IP, the domain name can be set in the nameserver files, */etc/inet/named.boot* or */etc/resolv.conf*, using a line of the form:

```
domain company.com
```

- The domain name can also be overridden within the mail configuration file using a line of the form:

```
DOMAIN=.company.com
```

Setting Up SMTP

When the UNIX System is delivered, SMTP is not configured. The following steps must be followed to configure your system to use SMTP.

- Set up mail to use SMTP. This is done by editing the */etc/mail/mailsurr* file and removing the # character from the beginning of the line that invokes **smtpqer**.
- The following commands may be used to do this (running as root):

```
# ed /etc/mail/mailsurr
g/smtpqer/s/^#//
w
q
#
```

- Set up the SMTP **cron** entries. Edit the **crontab** entry for root and remove the # character from the beginning of the lines which invoke **smtpsched**. The following commands may be used to do this (running as root):

```
# crontab -l > /tmp/cr.$$
# ed /tmp/cr.$$
g!/smtpsched!s/^#//
w
q
# crontab /tmp/cr.$$
# rm /tmp/cr.$$
#
```

How to Find Out More

You can learn about using the Basic Networking Utilities, including the UUCP System, in the User's Guide, which is part of the UNIX System V Release 4 Document Set. Useful references for using the UUCP System include the following books. Note that some of the specialized references on UUCP are out of print (reflecting the legacy nature of the UUCP System). You may be able to find copies of these out-of-print books at used bookstores or by accessing online booksellers and book search services that specialize in out-of-print books.

Anderson, Bart, and Bryan Costales. *UNIX Communications and the Internet.* 3rd ed. Indianapolis, IN: Howard W. Sams & Company, 1994 (out of print).

Redman, Brian. "UUCP UNIX-to-UNIX Copy." In Stephen G. Kochan and Patrick H. Wood, Consulting Editors. *UNIX Networking.* Indianapolis, IN: Hayden Books, 1989 (out of print).

Todino, Grace, and Dale Dougherty. *Using UUCP and USENET.* Newton, MA: O'Reilly & Associates, 1991 (out of print).

You can find out more about administering the Basic Networking Utilities (including the UUCP System) by consulting the UNIX System V Release 4 *System Administrator's Guide.* The manual pages for the corresponding administrative commands are found in the *System Administrator's Reference Manual.* Here are several useful references for administering the Basic Networking Utilities:

Anderson, Bart, Bryan Costales, and Harry Henderson. *UNIX Communications.* Indianapolis, IN: Howard W. Sams, 1987 (out of print).

O'Reilly, Tim, and Dale Dougherty. *Managing UUCP and Usenet* (revised version). Newton, MA: O'Reilly & Associates, 1988 (out of print).

Veeraraghavan, Sriranga, and James C. Armstrong, "UUCP Administration," in Robin Burk, et al., *UNIX Unleashed: System Administrator's Edition.* Indianapolis, IN: Sams, 1997.

Administration of the UNIX System V mail system is covered in the *System Administrator's Guide.*

For information about Taylor UUCP, which is used in Linux, consult:

Kirch, Olaf, and Andy Oram. *Linux Network Administrator's Guide.* Newton, MA: O'Reilly & Associates, 1995.

You may also find a rich history of the UUCP System on the web by using the term "UUCP" as a search term.