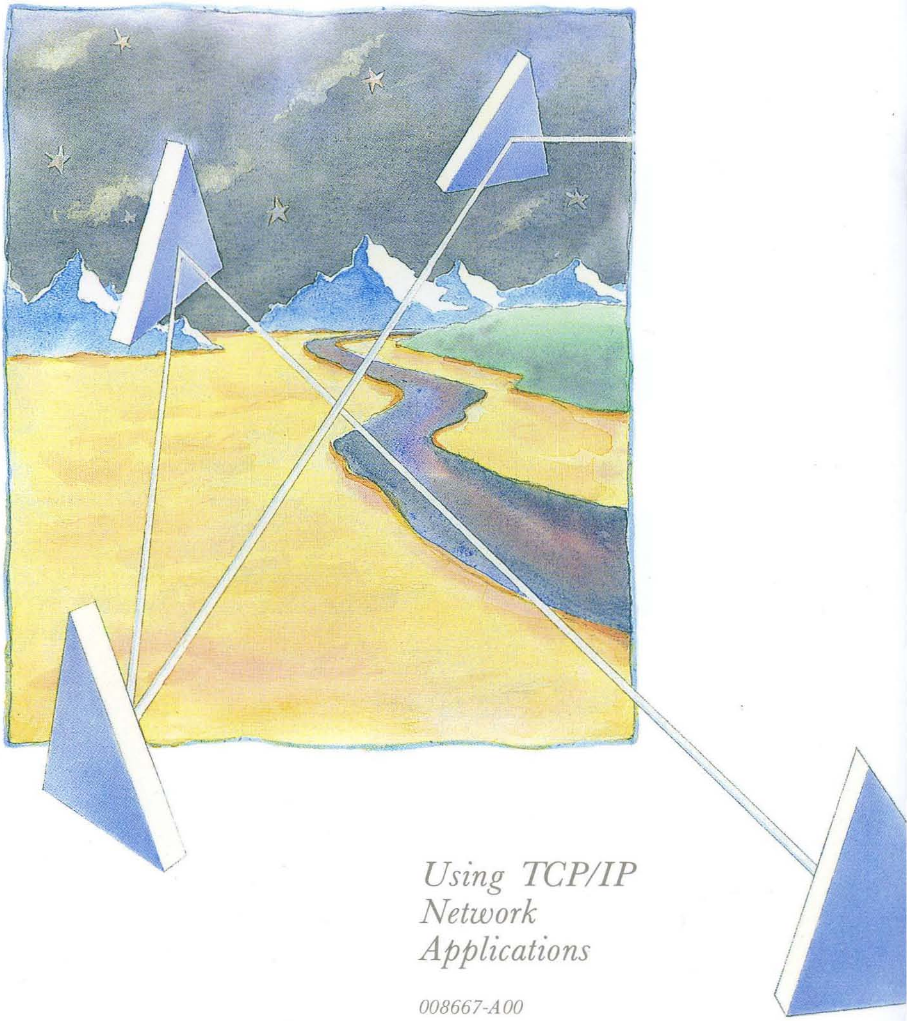

Using TCP/IP Network Applications

008667-A00

apollo

Using TCP/IP Network Applications

Order No. 008667-A00

Apollo Computer Inc.
330 Billerica Road
Chelmsford, MA 01824

First Printing: July 1988

This document was produced using the Interleaf Technical Publishing Software (TPS) and the InterCAP Illustrator I Technical Illustrating System, a product of InterCAP Graphics Systems Corporation. Interleaf and TPS are trademarks of Interleaf, Inc.

Apollo Computer Inc. reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should in all cases consult Apollo Computer Inc. to determine whether any such changes have been made.

THE TERMS AND CONDITIONS GOVERNING THE SALE OF APOLLO COMPUTER INC. HARDWARE PRODUCTS AND THE LICENSING OF APOLLO COMPUTER INC. SOFTWARE PROGRAMS CONSIST SOLELY OF THOSE SET FORTH IN THE WRITTEN CONTRACTS BETWEEN APOLLO COMPUTER INC. AND ITS CUSTOMERS. NO REPRESENTATION OR OTHER AFFIRMATION OF FACT CONTAINED IN THIS PUBLICATION, INCLUDING BUT NOT LIMITED TO STATEMENTS REGARDING CAPACITY, RESPONSE-TIME PERFORMANCE, SUITABILITY FOR USE OR PERFORMANCE OF PRODUCTS DESCRIBED HEREIN SHALL BE DEEMED TO BE A WARRANTY BY APOLLO COMPUTER INC. FOR ANY PURPOSE, OR GIVE RISE TO ANY LIABILITY BY APOLLO COMPUTER INC. WHATSOEVER.

IN NO EVENT SHALL APOLLO COMPUTER INC. BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATING TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF APOLLO COMPUTER INC. HAS BEEN ADVISED, KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY OF SUCH DAMAGES.

THE SOFTWARE PROGRAMS DESCRIBED IN THIS DOCUMENT ARE CONFIDENTIAL INFORMATION AND PROPRIETARY PRODUCTS OF APOLLO COMPUTER INC. OR ITS LICENSORS.

Preface

Using TCP/IP Network Applications describes how to use SR10–based TCP/IP network applications to communicate with other computers over a network. These applications allow you to get information about your network, login to another computer, copy files between computers, or remotely execute a program on another computer.

We've organized this manual as follows:

Chapter 1	**Understanding TCP/IP Network Applications** explains what TCP/IP is, what computers you can communicate with, and what kinds of tasks you can perform.
Chapter 2	**Getting Information about the Network** describes how to use the programs that provide information about your network.
Chapter 3	**Using Shell–Level TCP/IP Commands** describes how to use the shell–level TCP/IP commands; these commands provide a single service at each invocation, rather than an interactive communication environment. The **rsh** command, described in this chapter, is

the only command that can execute a command on a remote computer without explicitly logging in.

Chapter 4 — **Using TELNET for Remote Login** describes the TELNET program for logging in remotely to another computer on your network.

Chapter 5 — **Using FTP for File Transfer** describes the FTP file transfer program, which allows you to copy files back and forth between your computer and a remote computer on the network.

A Glossary and Index follow Chapter 5.

Related Manuals

The file **/install/doc/apollo/os.v.***latest software release number*__**manuals** lists current titles and revisions for all available manuals. For example, at SR10.0 refer to **/install/doc/apollo/os.v.10.0__manuals** to check that you are using the correct version of manuals. You may also want to use this file to check that you have ordered all of the manuals that you need. (If you are using the Aegis environment, you can access the same information through the Help system by typing **help manuals**.)

Refer to the *Domain Documentation Quick Reference* (002685) and the *Domain Documentation Master Index* (011242) for a complete list of related documents.

For detailed reference information on the commands described in this book, see the *BSD Command Reference* (005800), *SysV Command Reference* (005798), or *Aegis Command Reference* (002547).

For introductory information on Domain/OS and the available operating environments and shells, see *Getting Started with Domain/OS* (002348).

For information on differences between SR10 TCP/IP and previous versions, see *Making the Transition to SR10 TCP/IP* (011717).

For information on planning, configuring, and managing TCP/IP internets, see *Planning Domain Networks and Internets* (009916), *Managing Domain Routing and Domain/OS in an Internet* (005694), and *Configuring and Managing TCP/IP* (008543).

For reference information on BSD socket calls, the transport interface used to implement the applications described in this book, see the *BSD Programmer's Reference* (005801).

References of the form **foo**(N), where *N* is a number or a number followed by a letter, refer to pages in the *BSD Command Reference* or the *BSD Programmer's Reference.*

Problems, Questions, and Suggestions

We appreciate comments from the people who use our system. To make it easy for you to communicate with us, we provide the Apollo Product Reporting (APR) system for comments related to hardware, software, and documentation. By using this formal channel, you make it easy for us to respond to your comments.

You can get more information about how to submit an APR by consulting the appropriate Command Reference manual for your environment (Aegis, BSD, or SysV). Refer to the **mkapr** (make apollo product report) shell command description. You can view the same description online by typing:

$ **man mkapr** (in the SysV environment)

% **man mkapr** (in the BSD environment)

$ **help mkapr** (in the Aegis environment)

Alternatively, you may use the Reader's Response Form at the back of this manual to submit comments about the manual.

Documentation Conventions

Unless otherwise noted in the text, this manual uses the following symbolic conventions.

literal values
Bold words or characters in formats and command descriptions represent commands or keywords that you must use literally. Pathnames are also in bold. Bold words in text indicate the first use of a new term.

user–supplied values
Italic words or characters in formats and command descriptions represent values that you must supply.

sample user input
In examples, information that the user enters appears in color.

output
Information that the system displays appears in this typeface.

[ ]
Square brackets enclose optional items in formats and command descriptions. Square brackets also have special meaning to some **ftp** commands.

{ }
Braces enclose a list from which you must choose an item in formats and command descriptions. Braces also have special meaning to some **ftp** commands.

|
A vertical bar separates items in a list of choices.

< >
Angle brackets enclose the name of a key on the keyboard.

CTRL/
The notation CTRL/ followed by the name of a key indicates a control character sequence. Hold down <CTRL> while you press the key.

⋮	Vertical ellipsis points mean that irrelevant parts of a figure or examples have been omitted.
——— ⊞ ———	This symbol indicates the end of a chapter.

Contents

Chapter 1 Understanding TCP/IP Network Applications

Chapter 2 Getting Information about the Network

Chapter 3 Using Shell–Level TCP/IP Commands

Chapter 4 Using TELNET for Remote Login

Chapter 5 Using FTP for File Transfer

Chapter 5 Using FTP for File Transfer (cont.)

Glossary

Index

Figures

Tables

Chapter 1

Understanding TCP/IP Network Applications

This chapter explains what TCP/IP is, what computers you can communicate with using the TCP/IP network applications, what kinds of tasks you can perform, and how to get started using the applications.

1.1 What is TCP/IP?

In order to understand what TCP/IP is, you must first understand certain terms and concepts. A **network** may be thought of as two or more computers connected together by a cable or other medium, and running software which allows them to communicate over that medium. Computers connected to a network are called **nodes**. In this manual, the terms **node, computer, machine**, and **host** are used interchangeably to refer to computers on a network. Figure 1-1 shows an Apollo Token Ring network.

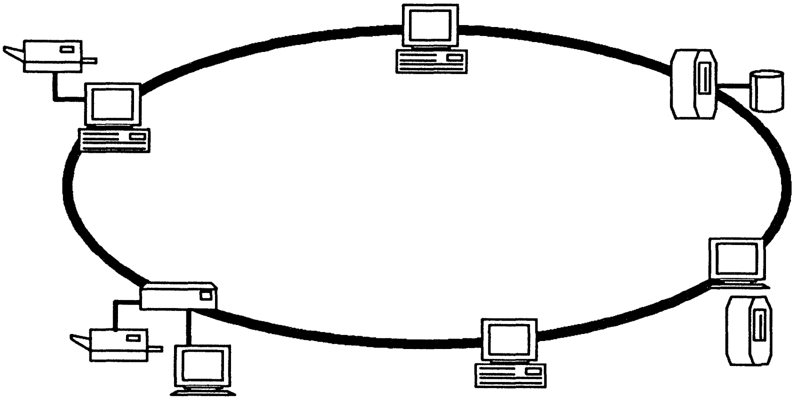

Figure 1-1. An Apollo Token Ring Network

Figure 1-2 shows an IEEE 802.3, or ETHERNET*, network, which supports communications among many different kinds of computers. Almost any kind of computer may be connected to an ETHERNET network. Such a network, with many different kinds of computers on it, is referred to as a **heterogeneous** network.

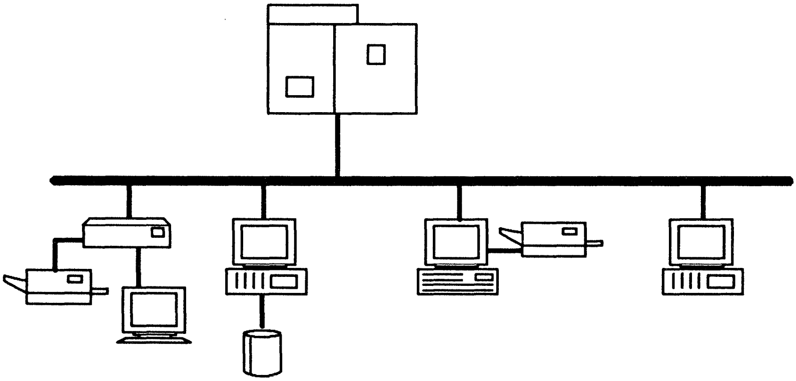

Figure 1-2. An ETHERNET Network

* ETHERNET is a registered trademark of Xerox Corporation.

An **internet** consists of two or more networks connected together by gateways and/or bridges, which route data between different networks. A **gateway** is a computer that is directly connected to two dissimilar networks and that is running software that can accept data from one network, repackage that data using the protocols prescribed by the second network, and send the data on to its final destination on the second network.

A **bridge** is a network connecting two computers, each of which is also connected to another, primary, network. A computer on one end of the bridge can accept and send data from its primary network across the bridge network to the computer at the other end of the bridge, which then sends it on to its destination. The networks connected by a bridge must be of the same type, as no protocol translation is done. In actual practice, however, many bridges have gateways at one or both ends and thus can connect dissimilar networks.

Figure 1–3 shows two Apollo Token Ring networks, each connected to the same ETHERNET network. The first (Network A) is connected to the ETHERNET (Network B) with a bridge/gateway combination. The second (Network C) is connected by a gateway.

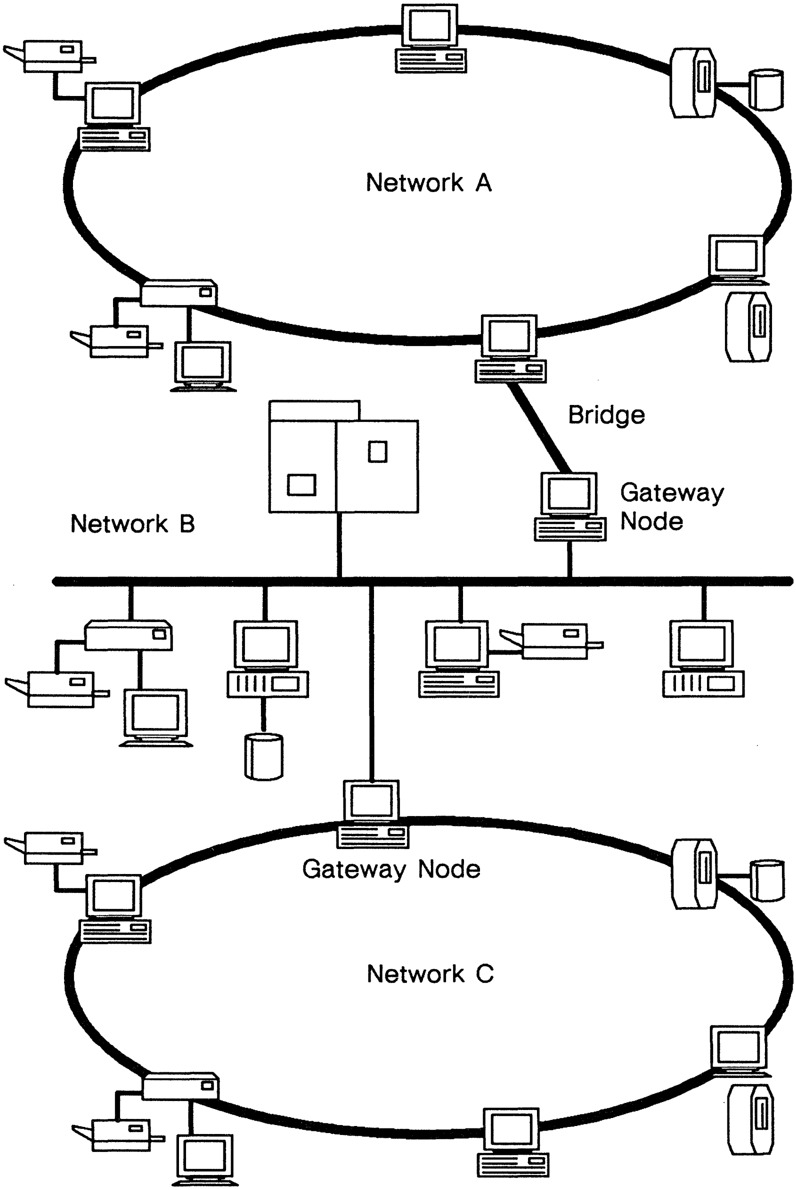

Figure 1–3. An Internet

SR10–based TCP/IP is a collection of software programs that provide data communications services over Apollo Token Ring or ETHERNET networks and internets. It provides the following:

- System software responsible for the actual transmission and reception of data
- Programming tools useful for developing specialized applications that communicate over a network
- A set of applications that allow users to copy files from one computer to another, login to another computer, or execute a command on a remote computer

The user applications provided by TCP/IP are the subject of this book. They provide Domain users with the ability to communicate with remote computers over heterogeneous networks and internets, such as the internet shown in Figure 1–3. They are especially useful for communicating with non–Apollo computers.

1.2 What Kinds of Computers Can I Communicate with?

The TCP/IP applications can communicate with any other computer on your network or internet, including Apollo nodes, provided the remote computer supports TCP/IP protocols and applications.

Implementations of TCP/IP vary in what services they provide and in how they provide them. Also, because TCP/IP was originally developed on UNIX* systems, many of the applications are UNIX oriented, and do not work as effectively when used with non–UNIX systems. Table 1–1 shows the systems each application can communicate with. Check with your system administrator if you are unsure whether an application can communicate with a particular computer.

* UNIX is a registered trademark of AT&T in the USA and other countries

For most operations, you must have a user login account on the remote computer you wish to communicate with. Exceptions are **tftp** and the commands that provide only information about the network (**hostid, hostname, netstat, ruptime,** and **rwho**). Your system administrator can provide information about user login accounts.

Table 1-1. TCP/IP Application Summary

Application	**Function**	**Works With**
ftp	File transfer	Any System
hostid	Print host ID	Local System
hostname	Print host name	Local System
netstat	Print network statistics	Local System
rcp	Remote file copy	UNIX Systems
rlogin	Remote login	UNIX Systems
rsh	Remote shell execution	UNIX Systems
ruptime	Print up-time statistics for computers on the network	UNIX Systems
rwho	List users logged into computers on network	UNIX Systems
telnet	Remote login	Any System
tftp	File transfer	Any System

1.3 What Kinds of Tasks Can I Perform?

There are four general categories of tasks you can perform with the TCP/IP applications:

- Getting information about the network
- Logging in to another computer
- Copying files to or from another computer
- Executing a program on another computer

Several commands provide information about the network. Information about computers on the network, including your own computer, and about users on those computers, can be displayed. Some of these commands will provide information only about other UNIX systems. Getting such information about the network is discussed in Chapter 2.

You can login to a remote computer with either the **rlogin** command or the **telnet** program. The **rlogin** command is simpler and easier to use. The **telnet** command is able to communicate with more kinds of computers, and provides more control over communications parameters. Chapter 3 contains a description of **rlogin**; **telnet** is discussed in chapter 4.

Three commands allow you to copy files between your computer and another computer. The **rcp** command allows you to copy files without explicitly logging in to the remote computer. The **tftp** command allows file transfer to or from computers that you do not have a login account on. The **ftp** command can communicate with a wider range of computers, and offers much greater control over file transfers. The **rcp** and **tftp** commands are discussed in Chapter 3. The **ftp** command is described in Chapter 5.

Remote program execution under TCP/IP is provided only by the **rsh** command, discussed in Chapter 3. This command allows you to run a command on another computer and redirect input and output from your local computer, without explicitly logging into the remote computer.

1.4 How Do I Access the Network Applications?

Domain/OS provides three **operating environments** in which you can work. Each operating environment is distinct from the Domain/OS operating system, which controls your workstation's hardware and provides a set of basic system services for the operating environments to use. An operating environment provides the user with the following facilities:

- A directory tree containing standard system directories
- A set of commands that provide basic computing functions
- One or more **shells**, the command interpreting programs that prompt you for commands and then execute them
- Other administrative and programming facilities

The available operating environments are:

Aegis This is the Apollo proprietary operating environment.

BSD This provides a BSD4.3 UNIX environment.

SysV This provides an AT&T UNIX System V Release 3 environment.

Figure 1-4 shows the relationship between the user, the three operating environments, and Domain/OS.

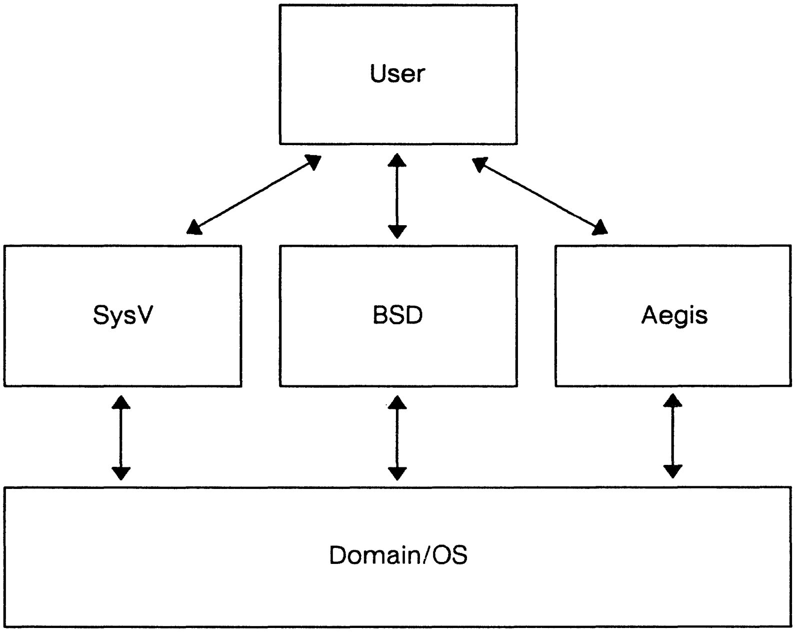

Figure 1-4. Operating Environments

TCP/IP applications available vary with the environment you use. Table 1-2 shows the applications available in each environment.

Table 1-2. Available TCP/IP Applications by Environment

Aegis	BSD	SysV
ftp	**ftp**	**ftp**
	hostid	**hostid**
	hostname	**hostname**
tcpstat *	**netstat**	**netstat**
	rcp	**rcp**
	rlogin	**rlogin**
	rsh	**remsh ***
	ruptime	**ruptime**
	rwho	**rwho**
telnet	**telnet**	**telnet**
	tftp	**tftp**
	whois	**whois**

NOTE: Commands marked with an asterisk perform the same function as the equivalent command in the same row of the BSD column, but have been given different names so as not to conflict with existing commands of the same name. Where a command name appears in a chapter or section title of this manual, the BSD name is used. Where a command's usage differs between environments, an example is provided for each environment.

The rules for accessing commands vary with your operating environment, but the default command location rules for each environment should provide access to the available TCP/IP commands. You should also be aware that different shells evaluate command lines differently. Conventions and methods for quoting arguments and evaluating variable expressions and pattern matching characters in filenames differ from shell to shell. Where there are significant differences, examples of each are provided. See *Getting Started with Domain/OS* for more information on operating environments and shells. See the Preface for a complete list of related manuals and order numbers.

1.5 Getting Started

Before getting started, make sure you are at an Apollo workstation connected to an Apollo Token Ring or ETHERNET network, and that the appropriate operating environment(s) and SR10–based TCP/IP are installed and configured correctly on your workstation.

If you are unsure about your network or software, or if you experience unexpected results when you try to use the TCP/IP applications, consult your system administrator.

Chapter 2

Getting Information about the Network

This chapter describes the TCP/IP applications for getting information about your network. Some of these commands provide information about your local computer, others provide information about other computers on the network and about users logged in to computers on the network.

It is important to understand how TCP/IP identifies computers. Each computer on the network has a unique set of identifier numbers, known as its **address**, or **hostid**. This address is in a form known as **Internet dot format**. It consists of four numbers separated by dots; for example, 192.54.3.9 is a valid internet address.

Most computers also have a unique name, called the **hostname,** that can be used in place of the hostid in many contexts. It is usually much easier to refer to a computer by name.

2.1 The hostid and hostname Commands

Two commands, **hostid** and **hostname**, tell you the network address and hostname, respectively, of the computer you are working on. They can also be used to set the address or hostname of your computer, provided you have appropriate access on your computer. This use of the commands is described in *Configuring and Managing TCP/IP*.

These commands are typed directly at your shell prompt. When the command completes, the results are shown on your screen, and a new prompt is displayed. The output looks like this:

```
$ hostid
192.89.7.2
$ hostname
pymatuning
$
```

2.2 The ruptime and rwho Commands

The **ruptime** and **rwho** commands provide information about computers on the network. You may want to find out if a particular computer is running and communicating normally with the network, or whether a particular user is logged on, and if so, to what computer or computers.

These commands are, generally speaking, useful only for getting information about UNIX computers on your network. UNIX implementations of TCP/IP usually arrange to broadcast certain information to the network once a minute, and to collect the information provided by other computers. Most non–UNIX implementations of TCP/IP do not provide appropriate information to the network.

NOTE: **ruptime** and **rwho** provide information only about computers on your *local network*, not about computers on the entire internet, even if you are connected to an

internet. See Chapter 1 for a discussion
of networks and internets.

2.2.1 Using ruptime

To find out what computers are *up*—running and communicating normally with the network—use the **ruptime** command. This command prints a list of all computers it knows about, including such information as how long the computer has been up (or down), and how many users are currently logged in. Computers from which no status reports have been received in the previous 11 minutes are assumed to be down.

The output of **ruptime** contains five fields:

name This field lists the hostname of the computer for which information is given.

up/down This field tells whether the given computer is up or down.

uptime This field tells, in days, hours, and minutes, how long the computer has been up, or down.

users If the computer is up, this field lists the number of users who are logged in and active on the specified computer, otherwise this field is not displayed. If the **-a** flag is used, users who have been idle for an hour or more, and who would thus not ordinarily be listed, are counted as well.

The example below shows what the output of **ruptime** typically looks like:

```
$ ruptime
abacus        up  4+20:28,    1 user
griffin     down  1+02:00
hal_9000      up  6+17:35,    0 users
python        up  3+02:50,    1 user
moment        up  2+19:51,    1 user
yo            up  1+19:22,    1 user
$
```

In the example, the machine **griffin** has been down for 1 day and 2 hours; **abacus** has been up for 4 days, 20 hours, and 28 minutes and has one user logged on.

Several options can be used with **ruptime**, most of which vary the order in which computers are listed. The **–a** option forces all logged–in users to be counted in the output. By default, users that have been idle for an hour or more are not listed. The available options are:

–a This option forces **ruptime** to count users who have been idle for an hour or more when it displays the number of users logged into a computer. Normally, only active users are counted.

–r Reverse the order in which output is sorted. By default the listing is in alphabetical order by machine name. This option may also be used in conjunction with any of the options below.

–t Sort the listing by up time.

–u Sort the listing by number of users.

2.2.2 Using rwho

The **rwho** command provides information about users logged in to computers on the network. The output is similar to the output of the UNIX **who** command, but lists users on other nodes. Like **ruptime**, this command depends on information broadcast by the other nodes on the network, and will not include information about users on computers that have not been heard from in the previous five minutes.

The example shows what output might look like:

```
$ rwho
john     abacus:display      Jan 21 09:19
henry    moment:display      Jan 21 10:57
user     hal_9000:display    Jan 21 11:45
user     yo:display          Jan 21 08:22
$
```

This output does not necessarily show all users logged in to computers on the network. Users who have been idle for an hour or more are not shown in the listing. The output contains three fields:

user This gives the name under which the user is logged into a node. Note that the same user name may appear more than once on the same or different nodes. It is possible for two different users on different nodes to have the same login name, or for the same user to be logged in more than once on the same node. In the example, **user** is logged in twice.

node:device This field gives the hostname of the machine the user is logged into, followed by a colon and the port on **hostname** that the user is logged into. The form that this port name takes varies with the type of system, and indicates the particular terminal line, communications port, etc., that the user is using. In the example, all users are logged into the **display** port of their respective machines.

date/idle time The last field gives the date and time that the user logged in. If the user has not typed into the system for a minute or more, this field shows how long he or she has been idle. If a user has been idle for an hour or more, that user will not be displayed at all, unless the **–a** option is used. There are no idle users shown in the example.

The only option to **rwho** is the **–a** flag, which specifies that all users should be listed, even if they have been idle over an hour.

2.3 The netstat Command

The **netstat** command provides detailed information about the state of TCP/IP software on your node, based on the contents of various network–related data structures. This information is generally used for administrative and troubleshooting purposes, but can be used by anyone to check the status of TCP/IP, or as a first line of inquiry when network applications do not appear to be functioning properly.

In the Aegis operating environment, this command is called **tcpstat**, but functions in exactly the same manner as **netstat** in the UNIX environments.

If you are not concerned with administrative or troubleshooting functions for TCP/IP, you may want to skip this section. If you continue, you should be familiar with the information in *Configuring and Managing TCP/IP*.

Several types of information are available from **netstat**. Three forms of the command can be used:

```
netstat [ -Aang ] [ -f addr_fam ]

netstat [ -himnrstT ] [ -f addr_fam ]

netstat [ -n ] [ -I interface ] interval
```

The **–T** option prints all available information. Use the other options singly or in various combinations to print specific subsets of the available information.

Valid command line options are:

–A With the first form of the command, show the address of any **protocol control blocks** associated with the sockets; this is used primarily for debugging purposes.

–a With the first form of the command, show the state of all sockets; normally sockets used by server processes are not shown.

–g With the first form of the command, show the first gateway used.

–h With the second form, show the state of the IMP host table. This table contains a mapping of Internet addresses (hostids) to physical addresses. The Address Resolution Protocol (ARP) is used to acquire physical addresses when only an Internet address is known.

–i With the second form, show the state of interfaces which have been enabled with the **ifconfig** utility. These are the physical network interfaces that can be used to send and receive data.

–I *interface* With the third form, show information only about this interface. This option is used with an *interval*; current statistics are printed every *interval* seconds.

–m With the second form, show statistics recorded by the network–private memory management routines.

–n With all forms, show network addresses as numbers (normally **netstat** attempts to interpret addresses and provide the appropriate host– or portnames.

–s With the second form, show per–protocol statistics.

–r With the second form, show the routing tables. When –s is also present, show routing statistics instead.

–t With the **–i** option, show a timer column. This indicates whether the given interface has timed out.

–T Show all possible status information.

–f *addr_fam* With the first and second forms, specify the address family of the sockets for which you want information. This may be **inet**, for internet addressed sockets, or **unix**, for UNIX intra–machine socket addresses.

2.3.1 Displaying Active Sockets

The first form of the **netstat** command displays a list of active **sockets**. A socket is an endpoint for a connection between two processes; these processes may be on the same machine, or, as in the case of communications protocols, on different machines. The default output shows sockets for connections to processes on remote machines. The output may be altered by specifying an address family. The default output does not display sockets used by the TCP/IP **server** processes, which listen for incoming connect requests; you can request that **netstat** display these sockets with the **–a** flag. The output of **netstat** used in this manner might look like this:

```
$ netstat -a
Active connections (including servers)
Proto Recv-Q Send-Q  Local Address     Foreign Address  (state)
tcp        0      0  *.ftp             *.*               LISTEN
tcp        0      0  *.exec            *.*               LISTEN
tcp        0      0  *.shell           *.*               LISTEN
tcp        0      0  *.login           *.*               LISTEN
tcp        0      0  *.telnet          *.*               LISTEN
udp        0      0  *.tftp            *.*               LISTEN
$
```

The output shows that only server processes have active sockets. They are all currently listening for requests from any foreign address; none have any data queued for sending or receiving. The addresses shown are of the form

host.port

if a host is known, or

network.port

if a network, but not a specific host, is known. The asterisk (*) character indicates either that a specific address is not known, or that any address is allowed. The addresses in this case are given by their names, not numbers, so the ports for the **telnet**, **ftp**, etc. processes are immediately discernible as such. The actual port numbers may be seen by using the **–n** flag to force **netstat** to print all addresses as numbers.

2.3.2 Examining System Data Structures

The second form of the **netstat** command provides various kinds of information culled from system data structures. The descriptions of the valid flags explain in detail the specific information presented. As an example, **netstat -r** displays the current state of the routing tables:

```
$ netstat -r
Routing tables
Destination    Gateway       Flags  Hops  Refcnt  Use  Interface
abacus         serenity      UG     1      0      0    dr0
smith          serenity      UG     4      0      0    dr0
griffin        hal_9000      UG     2      1      12   eth0
yo             moment        UG     1      0      0    eth0
$
```

This output shows various destination networks, the gateway nodes used to reach them, and flags showing that the routes are up (**U**) and that they are routes to gateways (**G**). The **Hops** field shows the number of gateways a packet must travel through to reach its destination. Also shown are counts of the number of active uses of the route (**Refcnt**), and of the number of packets transmitted over that route (**Use**). The **Interface** field shows which network interface the route uses.

2.3.3 Monitoring Network Statistics

The third form of the **netstat** command continuously displays network traffic statistics for a given interface, at specified intervals. The command shown generates current information every five seconds:

```
$ netstat 5
    input   (dr0)     output            input  (Total)    output
packets errs packets errs colls packets errs packets err colls
34842   0    6558    15   0     34842   0    6558    15  0
1       0    0       0    0     1       0    0       0   0
1       0    0       0    0     1       0    0       0   0
0       0    0       0    0     0       0    0       0   0
0       0    0       0    0     0       0    0       0   0
1       0    0       0    0     1       0    0       0   0
3       0    2       0    0     3       0    2       0   0
0       0    0       0    0     0       0    0       0   0
2       0    0       0    0     2       0    0       0   0
1       0    0       0    0     1       0    0       0   0
$
```

The display consists of two parts: in the five left–hand columns, statistics for the primary network interface (**dr0**) appear; the default interface may be changed with **–I**. In the five right–hand columns, totals for all interfaces appear. The number of input packets and errors, and the number of output packets and errors, along with the number of collisions, are displayed for both the given interface and for totals.

The first row of output contains totals accumulated since the system was booted. Subsequent rows are generated at intervals; this interval is given on the command line. In these rows, the numbers reflect incremental statistics since the last row was generated.

For more information on **netstat**, see *Configuring and Managing TCP/IP*.

Chapter 3

Using Shell–Level TCP/IP Commands

TCP/IP provides four shell–level commands used for communication primarily with other UNIX machines. These commands provide three basic capabilities:

- Remote login, provided by **rlogin**
- File transfer, provided by **rcp** and **tftp**.
- Remote program execution, provided by **rsh**

We refer to these commands as shell–level because they do not have their own interactive environment. The **telnet** and **ftp** commands offer robust and comprehensive remote login and file transfer capabilities by providing an environment in which the user is prompted to enter commands from a command set specific to the particular program. A **telnet** or **ftp** *session* may last a long time, and involve communications with several remote computers.

By contrast, the commands described in this chapter perform a single task at each execution, and do not provide their own environment complete with command set and interactive prompting. For example, to copy a file to a remote computer with **rcp**, you enter the command at your shell prompt, giving all necessary arguments, and when the transfer is complete your shell prompt returns.

3.1 Establishing Remote Access

With the exception of **tftp**, all programs described in this manual require that you have a login account on every remote host you want to communicate with. (You are *not* required to use the same username or password for each login account.) The **rlogin**, **rcp**, and **rsh** commands also use the files **hosts.equiv** and **.rhosts** to establish remote access.

Because these commands execute at the shell level and do not negotiate an interactive login, the remote host must have some way to assure itself that you are authorized to access its resources. In the case of the **rcp** and **rsh** commands, if this authorization can't be established, the command fails. Behavior in the case of **rlogin** is implementation dependent: if authorization can't be established, you must go through the normal login sequence to gain access; however, some remote hosts require you to do this even if authorization could be established.

Even though an interactive login does not take place, access to a remote machine is always granted through a login account, using an automatic login procedure. The remote machine assumes you entered a valid password when you logged into your local machine, and checks **hosts.equiv** or **.rhosts** to see if your local hostname, and possibly your local username, are listed. If so, access is granted.

3.1.1 The hosts.equiv File

The **hosts.equiv** file, found in the **/etc** directory of the remote machine, contains a list of hostnames, one to a line. The machines that appear in this file are considered to be **trusted**, or **equivalent**, by the remote host. A user trying to login to the remote host from a trusted host is granted access without a password, if the user's local username is the same as that of the login account he or she is trying to access on the remote machine.

If a user is trying to login with a different username, or the local machine is not listed in **hosts.equiv**, the file **.rhosts** is checked.

Your system administrator is responsible for editing **hosts.equiv**. The file may be empty or non–existent if there are no trusted hosts.

3.1.2 The .rhosts File

The **.rhosts** file resides on the remote machine, in the home directory of the user account you are logging into, and contains a list of machines and users that are authorized to login to that account without specifying a password. A user can thus grant login privileges, on a case by case basis, in addition to those granted by the **hosts.equiv** file. A user may also elect not to have a **.rhosts** file, or to have an empty **.rhosts**. For security reasons, **.rhosts** must be owned by the user who owns the account, or by **root**. (**root** is a special UNIX user with access to all system resources.)

> **NOTE:** In the SysV and BSD environments, and on remote UNIX machines, files beginning with a period (.) do not appear in default file listings. To see if **.rhosts** is present in a directory, use the **–a** option of the **ls** command. This causes all files to be listed.

Here is an example **.rhosts** file, assumed to be in the home directory of user **john** on host **smith**:

```
vaxl
serenity hopkins
serenity kathleen
yo       henry
```

This file allows access to the login account by a user **john** from the host **vaxl**, by the users **hopkins** and **kathleen** from the host **serenity**, and by the user **henry** from the host **yo**.

The format of a **.rhosts** file is a list of hostnames, one to a line, each optionally followed by a single username. If only a hostname is given, authorization is granted to a user logging in from that machine, provided his or her local username is the same as that of the user that owns the **.rhosts** file. If a hostname/username combination is specified, authorization is granted to a specific user on that machine, who need not share a username with the owner of the **.rhosts** file. To grant access to several users on a particular machine, list the machine once for each user. Use a single space to separate the username from the machine name.

3.1.3 Specifying a Remote User Account

By default, the shell-level commands try to log you into the remote host using the username you are logged into your local machine with. For example, if you are logged into **serenity** as **kathleen,** and execute the **rcp** command to copy a file to a remote host, **rcp** tries to log you into the remote host using the username **kathleen**.

The **rlogin, rcp,** and **rsh** commands allow you to specify an alternate username to login to the remote machine with. If you are logged into machine **abacus** as **henry**, and want to access the remote account of **hopkins**, you can indicate that on the command line. Specific information on how to specify an alternate user account is given with the description for each command.

3.1.4 Error Messages

If you receive an error message that says:

```
Login Incorrect.
```

while using **rcp** or **rsh**, then you are trying to login to a non-existent account, or one that has not granted you access through **hosts.equiv** or **.rhosts**.

The message:

```
Permission Denied.
```

may indicate an authorization problem, but can also occur if you try to access a file for which you do not have permission.

3.2 The rlogin Command

Remote login capabilities are provided by the **rlogin** command. This command is typed at your shell prompt, with a hostname as an argument. If you are communicating with a non–UNIX machine, you may want to use the Domain VT100* terminal emulator program:

```
$ vt100
$ rlogin vax_host
Connected to vax_host.
Login: henry
Password: [your password is not displayed as you type it]

         *** Welcome to vax_host! ***

         System going down for backups at 6:00pm

$
```

In this example, **henry** was able to login successfully to **vax_host**. The system's login message was displayed, followed by a system prompt.

You may also specify a login name on the command line. If the remote computer accepts this, it prompts you only for a password. For example:

```
$ rlogin vax_host -l henry
Connected to vax_host.
Password: [your password is not displayed as you type it]

         *** Welcome to vax_host! ***

         System going down for backups at 6:00pm

$
```

* VT100 is a trademark of Digital Equipment Corporation.

If you are logging into a UNIX host, you may not need to enter your login name and password. See Section 3.1, "Establishing Remote Access," for a discussion of how UNIX hosts establish login authorization for **rlogin**.

When you have finished your work on the remote computer, log out normally. In most cases, you are disconnected from the remote host automatically, and returned to your local shell prompt. If this does not happen, you must explicitly disconnect following logout. Do this by entering the sequence tilde/period (~.). For example:

```
[normal login session]
.
.
.
$ logout
Login: ~.
Connection closed.
$ [local system prompt]
```

The tilde (~) character is an escape character that tells **rlogin** to directly interpret the characters that follow, instead of sending them on to the remote host. See **rlogin** in the SysV or BSD command reference for more information.

> **NOTE:** You can disconnect using ~. without logging out of the remote host. The remote host may or *may not* log you out automatically. If you are not logged out, another user connecting to the host may be connected to your still–active login session, but there is no way to insure that your next **rlogin** will connect you to your previous login session.

3.3 The rcp Command

The **rcp** command allows you to copy a file between any two machines on your internet. This means that you can copy files between two remote machines, as well as between a remote machine and your local node. The syntax is very similar to the UNIX **cp** command. The general form of the command is as follows:

rcp [–p] *file1 file2*
rcp [–p] [–r] *file1 ... directory*

The first form of the command copies a file from one machine to another; the second form copies several files and/or directories to a directory on another machine. The command line arguments are:

–p This flag directs **rcp** to try and preserve in copies the modification times and permission modes of the source files. By default, **rcp** preserves the attributes of **file2** if it already existed, or otherwise uses the default file creation attributes on the remote host.

–r If any of the specified source files are directories, this flag causes the entire directory tree to be copied. In this case, the destination must be a directory.

file1 This is the source file (or files) for the copy. If the **–r** flag is used, either directory or file names may be used. The name is either an absolute or relative local pathname, or is a name taking one of the following forms:

host:pathname
user@host:pathname

The first form specifies a hostname and either an absolute pathname or a pathname relative to the home directory of the user you are logging in as. The second form specifies an alternate user to log in as on **host**. See "Establishing Remote Access" in this chapter for more information on user access to remote machines.

file2 This is a destination filename, specified in the same manner as **file1**.

directory This is a destination directory name, specified in the same manner as **file1**.

For example, the following command:

```
$ rcp -p yo:/usr/henry/report  henry_rpt
```

copies the file **/usr/henry/report** from the host **yo** to the file **henry_rpt** in your current directory on your local host, preserving the file modification times and permissions of the original file. The command:

```
$ rcp -r henry@yo:src_dir henry@yo:prog  john@smith:yo_dir
```

copies the directory tree **src_dir** and the file **prog**, both found in **henry**'s home directory on the host **yo**, to the directory **yo_dir** in **john**'s home directory on the host **smith**.

3.4 The rsh Command

The **rsh** command provides remote command execution, in a manner similar to the UNIX **sh** command. In the SysV environment, this command is called **remsh** to distinguish it from the SysV **rsh** command (a restricted shell).

The function of **rsh** is to connect to a remote host and execute a given command on that host. The standard input, standard output, and standard error output of the **rsh** command are passed to the remotely executing command. Interrupt, quit, and terminate signals are also passed to the remote command. Normally, **rsh** exits when the remote command does.

The general form of the command is as follows:

rsh *host* [**-l** *username*] [**-n**] *command*

The allowed arguments are:

host The hostname of the computer you want to execute **command** on.

-l This flag, if present, must be followed by a **username** to use when logging in to the remote machine. See the discussion under "Establishing Remote Access" in this chapter. No provision is made for specifying a password, if one is needed.

–n Redirect standard input from **/dev/null**. This is useful mostly with the C shell (**csh**). If you run an **rsh** as a background process in the C shell, the process blocks even though the remote process does not post any reads. This problem can be avoided through the use of **–n** (or be explicitly redirecting input from **/dev/null** or another file).

command This argument specifies the command you want to execute. This may consist of multiple words. If the command contains shell metacharacters, these are evaluated locally, unless quoted. You can not execute interactive programs, such as text editors; use **rlogin** or **telnet** instead. If the **command** is omitted, you are logged into the remote host with **rlogin**.

The **rsh** command is useful when you want to execute a single command line without explicitly logging into the remote computer, and without copying necessary input and output files to the remote. For example, a remote directory can be listed with this command:

```
$ rsh serenity "ls /usr/public"
```

You can also offload compute–intensive tasks with **rsh**. To run the **troff** text formatter on a remote machine, taking input from a local file and saving output on the remote host, you could use the following command:

```
$ rsh yo "troff > report_output" < report_troff
```

3.5 The tftp Command

The **tftp** program implements the Trivial File Transfer Protocol (TFTP), which allows file transfers between your local node and a remote host without user-level access. This means you can transfer files even if you have no login account on that machine. This is the only TCP/IP application that allows such transfers.

In the interest of security, remote files must be referred to with absolute pathnames (that is, a pathname beginning with / or //), must contain the string /**tftp**/, and can not contain the string /../. This insures that all remote files reside under a directory called **tftp**. The **tftp** command is thus useful only in cases where remote files have been specifically placed in a location accessible to **tftp**.

The syntax of the **tftp** command is as follows:

tftp –{g|g!|p|r|w} *local_file host remote_file* [*mode*]

You must provide exactly one of the flags indicating direction of transfer, and the **local_file, host**, and **remote_file** arguments. The **mode** argument is optional and specifies the type of file transfer you want to make. In more detail, the command line arguments are:

put (–p, –w) Write the **local_file** to the remote file **remote_file** on the remote node. The word **put**, and the flags **–p**, and **–w** are all synonymous.

get (–g, –r) Read the remote file **remote_file** into the local file **local_file**. If **local_file** already exists, **tftp** fails, and prints an error message. The word **get**, and the flags **–g** and **–r** are all synonymous.

get! (–g!) Read the remote file **remote_file** into the local file **local_file**, but overwrite **local_file** if it already exists. Note that in the C shell (**csh**), the exclamation point must be escaped (usually with a backslash) to avoid interpretation by the shell.

local_file This is the name of the local file that you want to copy to or from. If copying to this file, the file is either created or overwritten, never appended to. It may be an absolute or relative pathname, with no restrictions placed on it.

host This is the hostname of the remote machine you want to exchange files with.

remote_file This is the name of the remote file you want to copy to or from. This file is always created, never overwritten or appended to. For security reasons, the pathname is subject to the restrictions outlined at the beginning of this section.

mode This argument is optional, and specifies the type of file transfer. By default, the **mode** is **netascii**, which transfers the file as ASCII characters. The alternative is **image** mode, which transfers the file in binary, with no character conversion.

This command copies the local file **public** to a remote host called **jackson**:

```
$ tftp  put  info_file  jackson  /usr/public/tftp/info_file
```

To retrieve the same file, overwriting the current copy:

```
$ tftp  get!  info_file  jackson  /usr/public/tftp/info_file
```

Chapter 4

Using TELNET for Remote Login

The TELNET protocol is a general communications facility that allows you to log into another computer on your network. TELNET uses TCP/IP to establish a connection between two **network virtual terminals** (NVTs). An NVT is a virtual device that provides a standard representation of a terminal. It has well defined characteristics, such as the use of 7–bit ASCII data characters in an 8–bit field.

Each end of the communications connection has software that emulates an NVT. This technique eliminates the need for the communicating devices or processes to know each other's characteristics. It ensures that your node can communicate meaningfully with a remote host, even though the remote's software has no knowledge of the nodes's characteristics.

The **telnet** command uses the TELNET protocol to communicate with other hosts. With **telnet** you can log in to a remote host and use your node as if it were a local terminal on that machine. You can use **telnet** directly as a dumb terminal emulator, or you can run it within the Domain VT100 emulator to use the VT100 functions.

You can use **telnet** to communicate with any computer on your network or internet, provided the computer supports the TELNET protocol. Typically, both UNIX and non-UNIX implementations of TCP/IP support TELNET.

4.1 Getting Started — The telnet Command

Before describing how to invoke the **telnet** command, a discussion of **command mode** and **input mode** (also called **normal mode**) is in order.

4.1.1 Input Mode and Command Mode

When connected to a remote host, **telnet** is normally in input mode. This means that all input you type is passed directly to the remote node, just as if you were typing at a terminal directly connected to the remote. When **telnet** is running but you are not connected to a remote host, **telnet** is in command mode. In command mode, **telnet** displays this prompt:

```
telnet>
```

You may also enter a **telnet** command while you are connected to a remote machine, by using the **escape character** described later in this chapter.

There are 11 commands that **telnet** understands. They are used to establish and close connections to remote computers, to change communication parameters, and to change the status information **telnet** displays. Commands must be typed in lower-case only. You only need to type enough of the command to uniquely identify it; for example, you may enter the **quit** command by typing **q**. Table 4-1 summarizes the **telnet** commands.

Table 4-1. Summary of telnet Commands

Command	Function
open	Open a connection
close	Close this connection
quit	Close this connection and exit **telnet**
z	Suspend operation (C shell only)
mode	Set either "line-by-line" input mode or "character-at-a-time" mode
status	Show current status of **telnet**
display	Display some or all of **set** and **toggle** values
?	Display help information
send	Send special character sequences to the remote TELNET process
set	Set one of the **telnet** variables to a value
toggle	Toggle (between true and false) one of the **telnet** variables

4.1.2 The Escape Character

The **telnet** escape character is used to enter a command while you are connected to a remote computer, and are thus in input mode. The escape character is initially CTRL/], but can be changed with the **set** command.

To enter a **telnet** command while in input mode:

1. Enter the **telnet** escape character.
2. Enter the **telnet** command.

3. Type **<RETURN>**.

The command executes immediately after you type <RETURN>.

Only the characters between the escape character and the <RETURN> are interpreted as a **telnet** command. The **telnet** command may even be surrounded by normal mode data. For example, you can use the **telnet send ec** command to erase a character entered in error, as follows:

```
$ heloCTRL/]  [No <RETURN> is typed, CTRL/] isn't echoed.]
telnet> send ec
p
```

This is equivalent to typing **help** in normal mode.

4.1.3 Invoking telnet

You may run **telnet** in any shell, including a shell that is already running the VT100 emulator. The VT100 emulator provides you with full VT100 functionality over your TELNET connection. If you do not use the VT100 emulator, **telnet** will emulate a dumb terminal. An example showing the use of the **vt100** command appears in Section 3.2 "The rlogin Command" in Chapter 3.

The general form of the **telnet** command is as follows:

telnet [*remote_host* [*port*]]

The arguments are:

remote_host This is the hostname or numeric internet address of the machine you want to connect to. If it is omitted, **telnet** enters command mode and awaits further instructions.

port Specifying a port number tells **telnet** to request a connection on a particular port on the remote host. If a TELNET process is not listening for connect requests on that port, the attempt will be unsuccessful. You may give this argument only if the remote host is also specified; if **port** is omitted, **telnet** will try to connect on the default port for that machine.

When you invoke **telnet** without arguments, your display should look like this:

```
$ telnet
telnet>
```

You must then use the **telnet open** command to connect to a remote node, usually after using other **telnet** commands to set communications and display options to your liking.

A more typical invocation specifies a remote host to connect to. In this case, **telnet** informs you that it is trying to connect. If it is successful, **telnet** says so, and the remote host's login message appears; otherwise an error message is displayed. The following shows what this sequence might look like:

```
$ telnet hal_9000
Trying...
Connected to hal_9000.
Login:
```

You may also see an error message, if the connect attempt was unsuccessful. Two of the most common are:

```
$ telnet hal_9000
Trying...
Connection timed out.
telnet>
```

```
$ telnet hal_9000
hal_9000: unknown host
telnet>
```

The first message indicates that **telnet** was unable to connect to the remote host, and timed out. The other computer may be down, or having communication problems; a gateway node on the route to the other computer may have failed; or there may simply be very high usage of the network resulting in communication delays. The commands described in Chapter 2 may help you determine the reason for the message.

The second message indicates that you have tried to connect to a host that does not appear in the **/etc/hosts** file. Check for typing errors and try again.

Your system administrator can add new hosts to the **/etc/hosts** file, or help you determine the cause of a repeated connect failure.

4.1.4 Exiting telnet

When you are finished communicating with the remote host, you may log off in the usual fashion. Usually, the connection closes automatically, and you return to the **telnet** command mode. You can also break the connection by executing either the **close** or **quit** command via the escape character. The **quit** command will close the connection and exit **telnet**, leaving you back at your shell prompt.

4.2 Using telnet Commands

You can enter these commands whenever **telnet** is in command mode (that is, when the `telnet>` prompt appears at the beginning of the line) or between a **telnet** command escape character and a <RETURN>.

4.2.1 Connecting to a Remote Host with open

The general form of the **open** command is as follows:

open [*remote_id* [*port*]]

The **open** command initiates a connection to a remote host. If you invoke the **open** command without a **remote_id** argument, you will be in **telnet** command mode. The remote host's identifier can be either a TCP/IP host name or an Internet address (in the 'dot' format), e.g. 192.10.9.6.

remote_id The remote node you wish to connect to. This identifier may be a TCP/IP host name, or an Internet address in 'dot' notation.

Default if omitted: **telnet** prompts you for **remote_id** with (to).

The following option controls the attempt to connect to the remote node. You can only enter this option if you also specify the **remote_id** argument.

port Specify the port number to which **telnet** should attempt to connect.

Default if omitted: Use default port specified in **/etc/services**.

4.2.2 Closing a Connection with close

The general form of the **close** command is as follows:

close

The **close** command closes a **telnet** session and returns you to **telnet** command mode.

4.2.3 Exiting telnet with quit

The general form of the **quit** command is as follows:

quit

The **quit** command causes **telnet** to exit and control to return to the shell level. If a connection is open to the remote, **telnet** closes it before quitting.

4.2.4 Suspending Operation with z

The general form of the **z** command is as follows:

z

This command suspends the operation of **telnet** temporarily and returns control to the local shell process. The command is similar to the CTRL/Z sequence in the C Shell and you must be running **telnet** in a C Shell (**csh**) for it to work. To re–enter **telnet** command mode after a **z** command, type a per cent sign (%), followed by a <RETURN>, after the C Shell prompt. See *csh*(1) for further details.

4.2.5 Setting Input Mode Type with mode

The general form of the **mode** command is as follows:

mode *type*

The **type** argument is one of the following:

line Enter "line–by–line" input mode. In this mode, your local node does not send your input to the remote mode until you have typed a complete line and entered it with a carriage return. You may perform editing on the line locally, before it is sent.

character Enter "character–at–a–time" input mode. In this mode, your local node sends each character as you type it.

If the remote host is capable of entering the requested mode, it does so.

4.2.6 Getting Help on telnet Commands with ?

The general form of the ? command is as follows:

? [*command*]

The ? command displays brief descriptions of the **telnet** commands. If you omit the **command** argument, a list describing all the available commands appears. If you enter a **command**, only the line describing that command appears.

command Specify the command for which you want help. You only need to enter enough characters to uniquely identify the command.

Default if omitted: displays descriptions of all available **telnet** commands.

4.2.7 Displaying Current Status with status

The general form of the **status** command is as follows:

status

The **status** command displays the current status of the **telnet** connection, including the remote host's identifier, and the current mode.

4.2.8 Displaying Settings with display

The general form of the **display** command is as follows

display [*argument* ...]

The **display** command shows the current value of all **arguments** specified, or of all **arguments** if none are specified. An **argument** is one of the variables set with the **set** or **toggle** commands described later in this chapter.

4.2.9 Sending Commands to the Remote Host with send

The general form of the **send** command is as follows:

send *argument ...*

The **argument** is a TELNET command. This command is sent to the remote host. The following are legal commands:

escape Sends the current **telnet** escape character (set with the **set** command, initially CTRL/]).

synch Sends the TELNET SYNCH sequence. This causes the remote system to discard all previously sent (but not yet read) input characters. The SYNCH is sent as TCP urgent data. It may not work if the remote system is a 4.2 BSD UNIX system—if it doesn't work, a lower case 'r' may be echoed on the display.

brk Sends the TELNET BRK (break) sequence, which may have significance to the remote system.

ip Sends the TELNET IP (interrupt process) sequence, which should cause the remote system to abort the currently running process.

ao Sends the TELNET AO (abort output) sequence, which could cause the remote system to flush all output from the remote system to the user's display.

ayt Sends the TELNET AYT (are you there) sequence, to which the remote system may or may not choose to respond.

ec Sends the TELNET EC (erase character) sequence, which should cause the remote system to erase the last character sent to it.

el Sends the TELNET EL (erase line) sequence, which should cause the remote system to erase the line currently being entered.

ga Sends the TELNET GA (go ahead) sequence, which likely has no significance to the remote system.

nop Sends the TELNET NOP (no operation) sequence.

? Prints out help information for the **send** command.

4.2.10 Setting Parameters with set

The general form of the **set** command is as follows:

set *argument value*

The **set** command sets the given **argument**, which is a variable controlling some aspect of **telnet**'s operation, to **value**. The special **value** *off* turns off the function associated with the variable. You can use the **display** command to find the current value of a variable. The possible **arguments** are:

echo This is the value (initially CTRL/E) that, when in line–by–line mode, toggles between local echoing of entered characters (for normal operation) and suppression of echoing of entered characters. It is useful to turn echoing off when entering information you don't want displayed, for example, a password.

escape This is the **telnet** escape character (initially CTRL/]). The escape character causes **telnet** to interpret characters typed after it as a command, when in input mode. A <RETURN> is used to end the command.

interrupt Sets the interrupt character. If **telnet** is in **localchars** mode (see **toggle localchars** below), and the interrupt character is typed, a TELNET IP sequence (see **send ip** above) is sent to the remote host.

quit Sets the quit character. If **telnet** is in **localchars** mode and the quit character is typed, a TELNET BRK sequence is sent to the remote host.

flushoutput Sets the flush character. If **telnet** is in **localchars** mode and the quit character is typed, a TELNET AO sequence is sent to the remote host.

erase Sets the erase character. If **telnet** is in **localchars** mode and is operating in character–at–a–time mode, then typing the erase character generates a TELNET EC sequence to the remote host.

kill Sets the kill character. If **telnet** is in **localchars** mode and is operating in character–at–a–time mode, then typing the kill character generates a TELNET EL sequence to the remote host.

eof Sets the eof character. If **telnet** is operating in line–by–line mode, entering this character as the first character on a line causes an EOF to be sent to the remote system.

The initial settings for all of these characters, except the **escape** character, are taken from the user's environment when **telnet** is invoked.

4.2.11 Toggling Parameters with toggle

The general form of the **toggle** command is as follows:

toggle *arguments ...*

This command toggles the value of the boolean variable **argument** between true and false. Multiple **arguments** can be specified. The current values of these variables can be seen with the **display** command. Valid **arguments** are:

localchars

If this variable is true, then the flush, interrupt, quit, erase, and kill characters (see **set** above) are recognized locally, and transformed into appropriate TELNET control sequences. The initial value for this variable is true in line–by–line mode, and false in character–at–a–time mode.

autoflush

If autoflush and localchars are both true, then when flushoutput, interrupt, or quit characters are recognized (and transformed into TELNET sequences), **telnet** refuses to display any data on the user's terminal until the remote system acknowledges that it has processed those TELNET sequences. The initial value for this variable is true if the terminal user had not done an **stty noflsh** prior to running **telnet**, and false otherwise (see **stty**(1)).

autosynch

If autosynch and local chars are both true, then when either the interrupt or quit characters is typed, the resulting TELNET sequences are followed by the TELNET SYNC sequence. This procedure should cause the remote system to begin throwing away all previously typed input until both of the TELNET sequences have been read and acted upon. The initial value of this variable is false.

crmod

This variable controls the interpretation of carriage returns received from the remote host. If **crmod** is true, carriage returns are changed to carriage return/line feed combinations. Carriage returns typed by the user are unaffected. This mode is useful with remote hosts that send only carriage return, never line feed. The initial value is false.

debug

This variable toggles socket–level debugging, and is useful only to system programmers. The initial value is false.

options If this is true, some internal **telnet** protocol processing data, having to do with TELNET options, is displayed. The initial value is false.

netdata If this is true, all network data is displayed, in hexadecimal format. The initial value is false.

? Displays the legal **toggle** variables.

4.3 An Example telnet Session

This section provides an example **telnet** session showing some of the features discussed in this chapter. The local host is a node called **hal_9000**. The remote host is a computer called **yo** running a version of 4.2 BSD UNIX. The user logs in as user **guest**.

```
$ telnet
telnet> set interrupt ^X
interrupt character is '^X'.
telnet> display interrupt
[^X]      interrupt.
telnet> toggle localchars
Will recognize certain control characters.
telnet> toggle localchars
Won't recognize certain control characters.
telnet> open yo
Trying...
Connected to yo.
Escape character is '^]'.

4.2 BSD UNIX (yo)

login: guest
Password: [Password is not displayed when typed.]
Last login: Fri Mar 25 13:16:38 from hal_9000
UNIX 4.2 Release 3.3
yo%
```

```
.
.
.
[The user types commands as if at a terminal connected to yo.]
.
.
.
yo% commandCTRL/]  [No <RETURN> is typed here.]
telnet> send ec
[A <RETURN> is typed here to complete the command.]
comman: Command not found.
yo% CTRL/]   [No <RETURN> is typed here.]
telnet> display
will flush output when sending interrupt characters.
won't send interrupt characters in urgent mode.
won't map carriage return on output.
won't recognize certain control characters.
won't turn on socket level debugging.
won't print hexadecimal representation of network traffic.
won't show option processing.

[^E]      echo.
[^]]      escape.
[^H]      erase.
[^O]      flushoutput.
[^C]      interrupt.
[^U]      kill.
[^?]      quit.
[^D]      eof.
[A <RETURN> is typed here to complete the command.]
yo% CTRL/]  [No <Return> is typed here.]
telnet> close
Connection closed.
telnet> quit
$
```

Chapter 5

Using FTP for File Transfer

This chapter describes the **ftp** command for transferring files between computers. You may transfer files from your local computer to another computer on the network, from the remote computer to your local computer, or between two remote computers.

The **ftp** command uses the File Transfer Protocol (FTP) to accomplish file transfers (and other functions). FTP, in turn, uses the TCP/IP protocols for communication over the network. Like **telnet, ftp** can communicate with both UNIX and non–UNIX machines on your network, provided the machine is running an **ftp** server that responds to connect requests from the network.

We describe the syntax and usage of the **ftp** shell command first, followed by descriptions of the commands you use to tell **ftp** what to do. These commands are broken up into several functional groupings, as shown in Table 5–1. The groupings are presented in the text in the same order in which they appear in the table.

Table 5-1. ftp Commands by Function

Functional Group	**Command**
Establishing and closing connections to remote nodes	**open, user, account, sendport, reset, close, disconnect, bye, quit**
Transferring files	**put, send, mput, get, recv, mget, append**
Listing and manipulating directories and files on the local and remote nodes	**lcd, cd, pwd, ls, dir, mls, mdir, mkdir, rmdir, rename, delete, mdelete**
Controlling interpretation and generation of file names during file transfer	**glob, case, ntrans, nmap, runique, sunique**
Specifying parameters for transfer of different types of files	**type, ascii, binary, tenex, struct, mode, form, cr**
Displaying help and status information	**status, verbose, prompt, hash, bell, help, ?, remotehelp**
Advanced features such as auto-login, third-party file transfer, macro definition and execution, local shell execution, and others	**trace, debug, quote, !, $, macdef, proxy, .netrc***

* **.netrc** is not a command, it is a user-controlled initialization file.

5.1 The ftp Shell Command

The **ftp** command is typed directly at your shell prompt. After invocation it operates interactively: displaying a prompt, waiting for a command to be entered, then executing the command. If a remote host is given on the command line, **ftp** tries to connect to that host before it displays the first prompt. Otherwise, it simply displays a prompt and waits for a command:

```
$ ftp
ftp>
```

You must have a login account on any machine you want to connect to.

Several command line options are available to control the operation of **ftp**; most of these options may also be set or overridden after **ftp** is invoked. The general form of the command is as follows:

ftp [-vding] [*host* **]**

The command line options may be specified together or separately in any order, but must appear before the **host** argument. The options are described below:

host If given, this must be a valid hostname, or a hostid specified in Internet "dot" format.

–v Show responses from the remote **ftp** server and report on data transfer statistics. This may also be turned on (or off) with the **verbose** command to **ftp**.

–d This options enables debugging output. Debugging output may also be turned on (or off) with the **debug** command.

–i Turn off interactive prompting during multiple file transfers. This may turned off (or on) with the **prompt** command.

–n Turns off **auto–login**. By default, **ftp** attempts to log you in to a remote machine using the login name you are logged in to your local node with. You can also specify different login names to use with specific remote hosts; see "Using Advanced ftp Features" in this chapter for a description of this facility.

–g Disable filename **globbing** (filename expansion using wildcards). Filename expansion may also be turned off (or on) with the **glob** command.

To exit **ftp**, enter the **bye** or **quit** command. These commands are synonymous with each other. Either one causes **ftp** to close any open connections to remote machines and to exit. You are then returned to your shell prompt.

> **NOTE:** Correct operation of most **ftp** commands depends on proper behavior by the remote FTP server.

The remaining sections of this chapter describe **ftp** commands in detail. Table 5–2 provides a summary of these commands.

Table 5–2. Summary of ftp Commands

Command	Description
!	Execute a local shell command.
$	Execute an **ftp** macro.
account	Supply a supplemental password.
append	Append a local file to a remote file.
ascii	Set file transfer type to network ASCII.
bell	Set bell to sound after each file transfer.
binary	Set file transfer type to binary.
bye	Close connection(s) and exit **ftp**. A synonym for **quit**.
case	Toggle remote filename case mapping.
cd	Change remote working directory.
cdup	Change remote directory to current remote directory's parent directory.
close	Close current connection and erase defined macros. Synonymous with **disconnect**.
cr	Toggle carriage return stripping.
delete	Delete a remote file
debug	Toggle debugging mode.
dir	List a remote directory.
disconnect	A synonym for **close**.

(Continued)

Table 5–2. Summary of ftp Commands (Cont.)

Command	Description
form	Set the file transfer form.
get	Copy a remote file to the local machine.
glob	Toggle filename expansion.
hash	Toggle hash–sign (#) printing for data blocks transferred.
help	Print help information.
lcd	Change local working directory.
ls	List a remote directory
macdef	Define a macro.
mdelete	Delete multiple remote files.
mdir	List multiple remote directories.
mget	Copy multiple remote files to the local machine.
mkdir	Make a directory on the remote machine.
mls	List multiple remote files.
mode	Set file transfer mode.
mput	Copy multiple local files to the remote machine.
nmap	Set or unset filename mapping.
ntrans	Set or unset filename character translation.

(Continued)

Table 5–2. Summary of ftp Commands (Cont.)

Command	Description
open	Open a connection to a remote host.
prompt	Toggle prompting for each file of a multi–file operation.
proxy	Execute an **ftp** command on a secondary connection.
put	Copy a local file to the remote machine.
pwd	Print remote current working directory.
quit	Close any open connection and exit **ftp**. Synonymous with **bye**.
quote	Send arguments verbatim to remote FTP server.
recv	Copy a remote file to the local machine. Synonymous with **get**.
remotehelp	Request help from the remote FTP server.
rename	Rename a file on the remote host.
reset	Clear the reply queue, resynchronize command/reply sequencing with the remote FTP server.
rmdir	Delete a remote directory.
runique	Toggle storing of files (on local machine) with unique filenames.
send	Copy a local file to the remote host. Synonymous with **put**.

(Continued)

Table 5–2. Summary of ftp Commands (Cont.)

Command	Description
sendport	Toggle the use of PORT commands.
status	Show the current status of **ftp**.
struct	Set the file transfer structure.
sunique	Toggle storing of files (on remote machine) with unique filenames.
tenex	Set file transfer type to that needed for TENEX machines.
trace	Toggle packet tracing.
type	Set or display file transfer type.
user	Enter login name, password and account.
verbose	Toggle verbose output from **ftp**.
?	Display help information, synonymous with **help**.

NOTE: Some commands require arguments. See the description for each command for complete information on command syntax and arguments. Arguments that contain spaces may be quoted with the double–quote character (″).

5.2 Establishing and Closing Connections

The commands in this section are used to establish a connection to a remote host; to enter your remote user name, password, and account; to close the connection when you are finished with your work; and to exit **ftp**. Note that **ftp** automatically establishes a connection to a remote host that is given on the **ftp** command line—you do not need to use the **open** command again if you are already connected. If auto–login is not disabled, **ftp** also tries to log you in to the remote host automatically.

You may close one connection and open another one as many times as you like during an **ftp** session. It is also possible to connect to more than one remote host, and to transfer files between those hosts. See "Using Advanced ftp Features" for information on connecting to more than one remote host at a time and on using the auto–login feature.

5.2.1 Connecting to a Remote Host with open

The general form of the **open** command is as follows:

open [*host*] [*port*]

This command tries to establish a connection to an FTP server at the remote site identified by **host**. If you specify the optional **port** number, **ftp** attempts to contact a server at that port on **host**. If auto–login is enabled, **ftp** tries to log you in automatically, using, by default, the user name you are logged in to the local node with. The success or failure of the connect (and possibly login) is reported.

host Specify the host to which you wish to connect. If **host** is not given, **ftp** prompts you to enter it. The **host** may be specified as either a hostname or a numeric internet address (hostid).

port Specify a port number at which to open the connection. This is optional; a default port is used if **port** is not specified.

5.2.2 Logging into a Remote Host with user

user *user_id* [*password*] [*account*]

Use this command to log in to the remote host with your user ID. You usually use this command immediately after you establish the connection to the remote host with an **open** command. You may specify the **password** and **account** arguments on the command line, or wait to see if the remote host prompts you for them.

If the auto-login feature is enabled, all this takes place automatically.

user_id Specify the user name by which the remote host will be able to identify you.

password Specify the password for your account on the remote host. This argument is optional; the remote host prompts for a missing password if one is required. Local echo is turned off during password entry to preserve security. Note that if the password is supplied as an argument, it is visible on the display.

account Specify the account password for the remote host. This argument is optional; the remote host prompts for a missing password if one is required, and turns off local echo during entry. If given as an argument, the account is visible on the display. If an account is given, but not required during the login process, **ftp** sends it with an **account** command after the login sequence is complete.

5.2.3 Specifying an Account with account

account [*password*]

The **account** command allows you to provide a supplemental password if required by the remote system for access to resources after a login has been completed.

password	Specify the account password for the remote host. This argument is optional; the remote host prompts for the password if it is not given, and turns off local echo during entry. If given as an argument, the password is visible on the display.

5.2.4 Allowing Port Commands with sendport

sendport

Toggling port selection is the most common use of **sendport**. By default, **ftp** attempts to use an FTP PORT command to establish a connection for data transfer. If the PORT command fails, **ftp** uses the default port number. When **sendport** is used to turn port selection off, **ftp** does not attempt to use PORT commands. This command is useful when connected to remote FTP implementations that ignore PORT commands, but indicate (incorrectly) that they have accepted them.

By default, PORT commands are enabled; **sendport** indicates the current status when invoked.

5.2.5 Resynchronizing Communications with reset

reset

The reset command resynchronizes command/reply sequencing with the remote FTP server. This may be necessary following a violation of the FTP protocol by the remote host.

5.2.6 Disconnecting from a Host: close and disconnect

close
disconnect

The **close** and **disconnect** commands are synonymous. Either command closes the current connection to a remote host, and erases any macros that have been defined. See "Using Advanced ftp Features" in this chapter for a description of the macro facility. There is no need to explicitly log off the remote machine.

The **close** and **disconnect** commands *do not* cause **ftp** to exit. The **ftp** prompt (`ftp>`) is redisplayed, and you may then use other **ftp** commands to establish a new connection or perform other functions.

5.2.7 Exiting ftp: bye and quit

bye
quit

Both of these commands terminate any **ftp** session with a remote host and exit the local **ftp** process. Control is returned to your shell.

5.3 Transferring Files

The commands described in this section are used for copying files either from your local node to a remote host, or from the remote host to your local node. Before using any of these commands, you must first establish a connection to the remote machine to or from which you want to copy files.

When you are connected to a remote host, you have a current working directory on that host. You have a local current working directory at all times, whether connected to a remote host or not. All file name arguments are taken to be files in the local or remote current working directory, as appropriate, unless a pathname is given that specifies a different directory. See "Using Directory and File Control Commands" in this chapter for a more detailed discussion.

File transfer commands that accept multiple file arguments can also expand wildcard characters in filenames and perform various character and pattern translations. This capability, if present, is noted in the description for each command. See "Controlling Filename Interpretation" in this chapter for a full discussion.

In all cases, files are transferred using the transfer type, form, structure, and mode currently in effect. You do not need to change any of these parameters to transfer ASCII files between most computers. See "Setting File Transfer Parameters" in this chapter for a discussion of these parameters.

5.3.1 Sending a File to the Remote: put and send

put [*local_file*] [*remote_file*]
send [*local_file*] [*remote_file*]

The **put** and **send** commands are synonymous. Use either of them to copy the file **local_file** to the remote host under the name **remote_file**. If you don't specify the **remote_file** name argument, the file will be stored remotely with the name **local_file**.

local_file Specify the name of the local file you want to copy. If this argument is omitted, **ftp** prompts for the name of a file to copy.

remote_file Specify the filename you want the file to be stored as on the remote machine. If this argument is omitted, the file is stored in a file with the same name as the local file.

5.3.2 Sending Multiple Files with mput

mput *local_files*

The **mput** command transfers more than one local file to the current working directory on the remote system. Each file is stored on the remote host under the same name it had on the local system.

If wildcard expansion is turned on (see the **glob** command), wildcards are expanded on the local machine. They are otherwise taken literally. After wildcard expansion, filenames are processed according to the **ntrans** and **nmap** settings. (See the **ntrans** and **nmap** commands.)

Note that **mput** is not intended for transferring entire directory trees. This transfer may be done by using an appropriate utility on the local system, for example the SysV **cpio** command or the SysV and BSD **tar** command, to create a single archive file containing the directory tree, and then transferring that file. The remote system must have an equivalent utility to extract the directory tree from the archive file. The Aegis environment does not have a file archive utility, but the Aegis **cpt** command can be used from a shell to copy a directory tree between two Apollo nodes.

local_files Specify the names of the files to be transferred from the local host. If this argument is omitted, **ftp** prompts for filenames.

5.3.3 Getting a File From the Remote: get and recv

get [*remote_fil*e] [*local_file*]
recv [*remote_file*] [*local_file*]

The **get** and **recv** commands are synonymous. Use either to transfer **remote_file** from the remote host to the local file system. The file is stored on the local system with the name **local_file** if you specify that argument; otherwise, it is stored on the local system with the same name it had on the remote (i.e., **remote_file**).

remote_file Specify the name of the file to be transferred from the remote system. If this argument is omitted, **ftp** prompts for a remote filename.

local_file Specify a name for the file to be stored as on the local host. If this argument is omitted, the file is stored on the local node under the same name it had on the remote system.

5.3.4 Getting Multiple Files with mget

mget *remote_files*

Use the **mget** command to transfer **remote_files** from the remote host to the local file system. The files are stored in the local current working directory.

If wildcard expansion is turned on (see the **glob** command), wildcards are expanded on the remote machine. They are otherwise taken literally. After wildcard expansion, filenames are processed according to the **case, ntrans** and **nmap** settings. (See the **case, ntrans** and **nmap** commands.)

Note that **mget** is not intended for transferring entire directory trees. This may be done by using an appropriate utility on the remote system to create a single archive file containing the directory tree, and then transferring that file.

remote_files	Specify the names of files to be transferred from the remote system. If this argument is omitted, **ftp** prompts for the filenames.

5.3.5 Appending to a Remote File with append

append *local_file* [*remote_file*]

Use this command to append the contents of **local_file** to the end of **remote_file**. If **remote_file** does not exist, the **append** command creates it. If you don't specify a **remote_file** name, the remote file is taken to have the same name as **local_file**.

local_file	Specify the local file you want to append to the remote file. If this argument is omitted, **ftp** prompts for it.
remote_file	Specify the name of the remote file to which you want **local_file** appended. If **remote_file** does not exist, it will be created. If omitted, **remote_file** is assumed to have the same name as **local_file.**

5.4 Using Directory and File Control Commands

The notion of **current working directory** is used by **ftp** to interpret file pathnames. On your local node, your working directory is the directory you were in when you invoked **ftp**, until such time as you change it with the **lcd** command, described in this section. On the remote machine, your working directory is initially your login directory on that machine, until it is changed with the **cd** command. All **relative pathnames** (names that do not start with / or //) are interpreted relative to the current directory on both local and remote machines. All **absolute pathnames** refer to unambiguous directories or files, regardless of your current directory. This is also true on both local and remote machines.

This section describes the commands for directory and file control. The ability to change your working directory and list directory contents, both locally and remotely, greatly facilitates the process of transferring files. In addition, **ftp** commands for creating and deleting remote directories, and for deleting and renaming remote files are described here.

With the exception of the **lcd** command for changing your local working directory, all of the following commands operate on remote directories and files. To perform operations on local directories and files, use the local shell command (!) to execute a local command. This mechanism provides you with greater flexibility than could be provided by implementing a limited set of **ftp** commands for local operations. The ! command is described in "Using Advanced ftp Features" later in this chapter.

5.4.1 Changing the Local Working Directory with lcd

lcd [*pathname*]

The **lcd** command changes the working directory on the local machine. The **pathname** argument specifies the new directory.

NOTE: Using the local shell command (!) to execute a change directory command locally is ineffective when using UNIX shells—your directory is changed for only as long as the shell process that changed it is running. When that shell exits and you are returned to the **ftp** prompt, you are still in the same directory you were in before you executed the ! command. However, an Aegis **/com/sh** change directory command remains in effect even after the shell that executed it is no longer running.

pathname Specifies the new working directory. If **pathname** is omitted, change to the user's home directory.

5.4.2 Changing the Remote Working Directory with cd

cd *pathname*

Use this command to change the remote working directory to **pathname**. Unlike the **lcd** command, the **pathname** argument is required.

pathname Specify the pathname of the new directory on the remote host to which you want to change. If omitted, **ftp** prompts for a new working directory.

5.4.3 Reporting the Remote Working Directory with pwd

pwd

The **pwd** command prints the name of the current working directory on the remote host.

5.4.4 Listing Remote Files: ls and dir

ls [*pathname*] [*local_file*]
dir [*pathname*] [*local_file*]

The **ls** and **dir** commands both list the remote directory or file specified by **pathname**.

The **ls** command provides a brief listing, similar to the output of the UNIX **ls** command. For a directory, the file and/or directory names found in that directory are listed. For a file, only the name of that file is listed. If the **pathname** given cannot be found, that fact is reported.

The **dir** command provides a detailed listing of **pathname**, similar to the output of the UNIX **ls -lg** command. The filename, owner, group, and permissions of each file or directory found are listed, as well as some other information. Again, if **pathname** is a directory, its contents are listed, if a file, only that file is listed.

In either case, if **pathname** is not specified, the remote working directory is listed.

You may specify a local file in which the listing should be stored by including a **local_file** argument. If you don't include **local_file,** the listing will be output to your display.

pathname	Specify the name of the remote directory or file you want to list. If omitted, list the current working directory.
local_file	Specify the name of the local file to store the listing in. If omitted, or if a hyphen (–) is given, output the listing to the local display.

5.4.5 Listing Multiple Remote Files: mls and mdir

mls *pathnames local_file*
mdir *pathnames local_file*

The **mls** and **mdir** commands provide the same listing capabilities as **ls** and **dir**, but can accept multiple remote **pathnames** as arguments. These commands require that the **pathnames** argument be present; they also require a **local_file** argument for storing the output.

pathnames	Specify the directories and/or files on the remote host that you want to list. If this argument is omitted, **ftp** prompts for pathnames.

local_file Specify the file on the local machine that you want to store the listing in. If this argument is omitted, **ftp** prompts for it; you may specify a filename of hyphen (-) to send output to your display.

5.4.6 Creating a Remote Directory with mkdir

mkdir *pathname*

The **mkdir** command creates a new directory named **pathname** on the remote machine.

pathname Specify the name of the directory you are creating.

5.4.7 Removing a Remote Directory with rmdir

rmdir *pathname*

The **rmdir** command deletes the directory named **pathname** on the remote machine. The directory must be empty; all files must have been removed with the **delete** or **mdelete** commands before the directory can be removed.

pathname Specify the name of the directory you wish to delete.

5.4.8 Renaming Remote Files with rename

rename *old_name new_name*

Use this command to change the name of a directory or file on the remote system from **old_name** to **new_name**.

old_name Specify the name of the directory or file on the remote system that you want to change. If omitted, **ftp** prompts for a pathname.

new_name Specify the new name of that directory or file. If omitted, **ftp** prompts for the new name.

5.4.9 Deleting Remote Files with delete

delete *remote_file*

Use this command to delete a single file named **remote_file** on the remote host.

remote_file Specify the name of the file you wish to delete.

5.4.10 Deleting Multiple Files with mdelete

mdelete *remote_files*

Use this command to delete several files named by **remote_files** on the remote host. If wildcard filename expansion is enabled, the **remote_files** will be expanded before the command is executed. See the **glob** command.

remote_files Specify the files you wish to delete.

5.5 Controlling Filename Interpretation

The facilities provided by **ftp** allow for wildcard filename expansion, case translation, character translation, and pattern substitution in interpreting filenames. These capabilities may be controlled through the use of the commands described in this section.

The most widely available facility is **globbing**, or wildcard filename expansion. When globbing is enabled with the **glob** command, any **ftp** command that accepts multiple file arguments will interpret wildcard characters in filenames given to it.

Case translation, if enabled, is available only during transfer of files from a remote host to the local node, and allows filenames that are all uppercase to be translated to lowercase when they are transferred. Other character translation and pattern translation facilities are also available during file transfers.

The commands for controlling these filename interpretation facilities are described here. The individual descriptions of commands that accept filenames as arguments provide specifics as to which facilities operate on the filenames they use.

5.5.1 Toggling Filename Expansion with glob

glob

The **glob** command turns filename expansion on or off, and reports the current setting. By default, **globbing** is on.

The term globbing refers to the way certain characters, called wildcards or metacharacters, are interpreted when used in filenames. If globbing is off, these characters are taken literally, just as other characters in filenames are, and match only filenames in which those specific characters appear. If globbing is on, filenames that contain wildcard characters may match several files, as described (the interpretation of these characters is identical to the way the BSD **csh** interprets them):

These characters, which include the asterisk (*), the question mark (?), square brackets ([]), tilde (~), and curly braces ({}), are interpreted as they would be in the C Shell. If **glob** is disabled, these characters are treated literally. The **glob** command reports its current setting, as well.

*

The asterisk character matches any number (including zero) of any characters. So, for example, the name ***.foo** matches any file that ends with the characters .foo. The name **report*1988** matches any filename beginning with the word report and ending with the number 1988, including the file **report1988**.

?

The question mark character matches any single character. The name ? matches any file with a single character filename, and the name **data?** matches any five–character filename beginning with the word data.

[ ] Square brackets are used to match a single character in a filename to one of a list or range of characters appearing within the brackets. For example, the name **chapter[1–9]** matches any file beginning with the word chapter and ending in a single digit from one to nine. The name **[A–Z][A–Z].info** matches any filename beginning with two uppercase letters and ending with the string .info. An example of a list of characters is the name **foo[apx0–9]bar**, which matches any file beginning with foo, followed by *one* of the characters a, p, or x, or a digit from zero to nine, and ending with the word bar.

{ } Curly brackets expand to a list of file, each containing one of the comma–separated strings enclosed. The name **a{x,y,z}b** expands to the three names **axb**, **ayb**, and **azb**. The name **{old,new}.db** expands to the names **old.db** and **new.db**.

~ The tilde character, if it is the first character in a filename and followed by a slash (/) character, expands to your home directory. So, the name **~/myfile** would match the file **myfile** in your home directory. A tilde followed by a name expands to that person's home directory. The name **~ken/hisfile** expands to the file **/usr/ken/hisfile**.

Any of these special characters may be preceded by a backslash (\) to prevent its special meaning. In addition, unmatched square or curly brackets, or empty sets of brackets, are interpreted literally.

> **NOTE:** The interpretation of these characters, particularly the tilde, in remote pathnames is dependent on the remote machine's ability to provide that capability.

5.5.2 Toggling Case Translation with case

case

The **case** command toggles case translation of filenames during **mget** transfers. By default case translation is off; when it is on, remote filenames that are completely uppercase are written in the local directory with the letters translated to lowercase.

5.5.3 Setting Filename Character Translation with ntrans

ntrans [*inchars* [*outchars*]]

If no arguments are specified, the filename character translation mechanism is turned off. If arguments are specified, characters in remote filenames are translated during **mput** commands and **put** commands issued without a specified remote filename. If arguments are specified, characters in local filenames are translated during **mget** commands and **get** commands issued without a specified local filename. This command is useful when connected to a non–UNIX remote computer with different file naming conventions or practices.

When operative, characters in a filename matching a character in **inchars** are replaced by the character appearing in the same position in the string **outchars**. If a character's position in **inchars** is greater than the length of **outchars**, the character is deleted from the filename.

inchars A string of characters used to match characters in filenames given to the **get**, **mget**, **put**, and **mput** commands. If a character in a filename is also in the string **inchars**, that character is replaced by the corresponding character from **outchars**. If this argument is omitted, the character translation mechanism is turned off.

outchars A string of characters used to replace characters matched by **inchars**. A character from **inchars** that is found in a filename is replaced by the character found in the same position in **outchars**. For example, if **inchars** is **abc** and **outchars** is **def**, then the file **xxaa** is changed to **xxdd**, and the file **yycc** is changed to **yyff**. If the string **outchars** is shorter than **inchars**, then characters in **inchars** that have no corresponding character in **outchars** are deleted. Thus, if **outchars** is not given at all, then all characters appearing in **inchars** are discarded. If **outchars** is longer than **inchars**, the extra characters are ignored.

5.5.4 Setting Character Pattern Mapping with nmap

nmap [*inpattern outpattern*]

If no arguments are specified, character pattern mapping is turned off. Otherwise, set character pattern mapping as specified by the arguments. Pattern mapping is used to generate remote filenames during **mput** and **put** commands, and local filenames during **mget** and **get** commands. This facility can be useful when connected to non–UNIX machines with different file naming conventions or practices.

inpattern This argument is a template for incoming filenames, which may already have been processed by the **case** and **ntrans** settings. The **inpattern** argument is used to break up filenames into component parts, which are then available for use in **outpattern**. The sequences **$1, $2, ... , $9** can be used in **inpattern** to specify parts of a filename, accessible then in **outpattern**.

outpattern This pattern uses the values set for **$1** through **$9** by **inpattern** to construct a new filename. **$0** is always set to the original filename. Additionally, the pattern **[seq1,seq2]** is replaced by **seq1** if it is not null, and by **seq2** otherwise.

The backslash (\) character can be used to prevent special treatment of the **$**, **[**, **]**, and **,** characters. Here are some examples showing how **nmap** works:

```
ftp> nmap $1 $0.new
```

This mapping appends the string **.new** to each filename as it is transferred.

```
ftp> nmap $1.$2.$3 [$1,$2].[$2,file]
```

This mapping generates the output filename **myfile.data** for the input filenames **myfile.data** and **myfile.data.old**; the output filename **myfile.file** for the input filename **myfile**; and the output filename **myfile.myfile** for the input filename **.myfile**.

5.5.5 Toggling Unique Local Filenames with runique

runique

The **runique** command toggles storage of files on the local file system with unique names. When **runique** is enabled (default is off), if a file already exists with the same name as the target local filename for a **get** or **mget** command, the string **.1** is appended to the name. If this results in another match, **.2** is appended. This continues until **.99** is reached, at which point an error message is printed and the transfer does not take place. The generated unique filename is reported. This parameter only affects filenames generated during **get** and **mget** commands.

5.5.6 Toggling Unique Remote Filenames with sunique

sunique

The **sunique** command toggles storage of files on the remote file system with unique names. When **sunique** is enabled (default is off), if a file already exists with the same name as the target remote filename for a **put** or **mput** command, the string **.1** is appended to the name. If this results in another match, **.2** is appended. This continues until **.99** is reached, at which point an error message is printed and the transfer does not take place. The generated unique filename is reported. This parameter only affects filenames generated during **put** and **mput** commands.

The remote FTP server must support the FTP protocol STOU command for successful completion.

5.6 Setting File Transfer Parameters

Use these commands to describe the format of the data being transferred. Since all data transfer parameters have default values, you need execute these commands only when you are transferring special types of data. Specified parameters remain in force until reset. Use the **status** command to check current parameter values. **ftp** permits only an 8-bit transfer byte size (the network default).

5.6.1 Specifying Transfer Type with type

type [ascii | binary | image | tenex]

Without an argument, the **type** command reports the current setting for the data type used for transfers. Use this command with one of the arguments to change the data type, after you've connected to a host. Specify **ascii** for ASCII files. This mode permits transfer of text between systems that have different conventions for storing ASCII text. Specify **binary** for binary files. The transfer type **image** can be used for both ASCII and binary transfers between two UNIX systems. You can use **tenex** for exchanges with TENEX systems. The default type is **ascii**.

Any of these types, except **image**, may be submitted directly as an **ftp** command, as well. That is, instead of typing:

```
ftp> type binary
```

you may type:

```
ftp> binary
```

The effect will be the same.

ascii Specifies that ASCII data is being transferred.

binary Specifies that data being transferred is binary.

image Specifies that ASCII *or* binary files are being transferred to a system running UNIX.

tenex Specifies that ASCII *or* binary files are being transferred to a system running TENEX.

5.6.2 Specifying Transfer Type: ascii, binary and tenex

ascii
binary
tenex

Use the **ascii, binary,** or **tenex** command to set that file transfer type. The effect is identical to using the **type** command with an argument.

5.6.3 Setting File Structure with struct

struct [*struct_name*]

The **struct** command specifies the structure of the file being transferred. The default structure is **stream; record** structure may be set to transfer record-structured files preserving record separators. You must be connected to a remote host to use this command. The current structure setting is reported by the **struct** command.

5.6.4 Setting Transfer Mode with mode

mode [*mode_name*]

The **mode** command sets the mode of the file transfer to **mode_name**. The default value for **mode_name** is **stream**; this is the only value supported.

5.6.5 Setting Transfer Format with form

form [*format*]

This command sets the file transfer format to **format**. The default file format, as well as the only format supported, is **file**.

5.6.6 Toggling Carriage Return Stripping with cr

cr

The **cr** command toggle stripping of carriage returns during **ascii** type file retrieval. Records are delimited by a carriage return/linefeed sequence during this form of transfer; when **cr** is enabled (the default), carriage returns are stripped from this sequence to conform with the UNIX single linefeed delimiter. Records on non–UNIX machines may contain single linefeeds; when an **ascii** type transfer is made, these linefeeds may be distinguished from a record delimiter only when **cr** is off.

5.7 Displaying Help and Status Information

There are several **ftp** commands that perform functions such as reporting status, providing help, controlling interactive prompting, and changing the default display information. These commands are described in this section.

5.7.1 Reporting Status Information with status

status

Use this command to obtain status information about the current **ftp** parameter settings.

5.7.2 Toggling Verbose Mode with verbose

verbose

When **verbose** is on, all responses from the remote **ftp** server are displayed at the local terminal. Statistics regarding the efficiency of the data transfer are also shown. The default setting for **verbose** is on. The command reports its current setting, as well.

5.7.3 Toggling Prompt Mode with prompt

prompt

Interactive prompting during multiple file transfers allows you to retrieve or store files selectively. If **prompt** is toggled off, which is the default case, an **mget** or **mput** command transfers all files without further prompting. The command reports its current setting, as well.

5.7.4 Reporting on Blocks Transferred with hash

hash

The **hash** command toggles hash–sign printing on and off. If hash–sign printing is enabled, a hash sign (#) is printed each time a data block (1024 bytes) is transferred. The command reports its current setting, as well.

5.7.5 Toggling Completion Bell with bell

bell

The **bell** command causes **ftp** to sound a terminal bell (or beep) after each file transfer command sequence is completed. The command reports its current setting, as well.

5.7.6 Getting Local Help with help or ?

help [*command*]
? [*command*]

Use the help commands to see a description of the available **ftp** commands. With the **command** argument, only the available information for that one **command** will appear.

command	Specify what **ftp** command you wish information about. Default if omitted: Displays a list and brief description of all local **ftp** commands.

5.7.7 Getting Remote Help with remotehelp

remotehelp [*command*]

Use this command to see a description of the **ftp** commands available on the remote machine. With the **command** argument, you will only see the available information for that one **command**, if it is available remotely.

command Specify what **ftp** command you wish information about. Default if omitted: Displays a list of all the remote **ftp** commands available.

5.8 Using Advanced ftp Features

5.8.1 Toggling Packet Tracing with trace

trace

The **trace** command switches the packet tracing facility on and off. The command reports its current setting, as well.

5.8.2 Setting Debug Mode with debug

debug

The **debug** command turns debugging off and on. When debugging is on, **ftp** prints each command sent to the remote machine, preceded by the string "-->".

5.8.3 Sending a Command with quote

quote *command*

Use this command to send an arbitrary **command** to the remote **ftp** server program. This arbitrary command is a command that would be recognized by the remote host, but not by **ftp**. A single **ftp** reply code is expected in return.

command Specify the command you want to send to the remote **ftp** process. If omitted: **quote** will prompt for the name of a command to send to the remote.

5.8.4 Executing a Local Shell with !

!

Typing ! creates a temporary Shell process on the local machine. In this process, you can use shell commands. While running the temporary shell process created by !, connections to the remote host remain open. If you need to run lengthy shell processes that aren't related to the **ftp** connection, you should use a different window altogether.

5.8.5 Executing a Macro with $

$*macroname* [*arguments*]

Typing $ executes the macro **macroname** with the argument list **arguments**. A macro is a named collection of **ftp** commands, which are executed when the macro is invoked. For more information, see the **macdef** command.

5.8.6 Defining Macros with macdef

macdef *name*

This command defines a macro called **name**. The definition itself begins on the next line and continues until an empty line is typed (two consecutive carriage returns at a terminal, two newlines in a file). The text of the definition consists of **ftp** commands, which are executed when the macro is invoked.

The macro processor interprets '$' and '\' as special characters. A $ followed by a number is replaced by the corresponding argument from the macro invocation command line during execution. The special case **$i** specifies that the executing macro is to be looped. On the first execution, **$i** is replaced by the first argument, on the second execution by the second argument, etc. A \ followed by any character is replaced by that character. It can thus be used to prevent the special interpretation of $.

Macros remain defined until a **close** or **disconnect** command is executed. There is a limit of 16 macros and 4096 total characters in all defined macros.

5.8.7 Controlling Secondary Connections with proxy

proxy *ftp_command*

The **proxy** command executes an **ftp** command on a secondary control connection. This allows simultaneous connection to two remote hosts for transferring between the two remotes. The first **proxy** command should be an **open** to establish the secondary connection. Enter the command:

```
ftp> proxy ?
```

to see the other **ftp** commands executable on the secondary connection. The following commands behave differently when prefaced by **proxy**:

- A **proxy open** does not define new macros during the auto–login process.
- A **proxy close** does not erase macros.
- The **get** and **mget** commands transfer files from the host on the primary control connection to the host on the secondary control connection.
- The **put, mput,** and **append** commands transfer files from the host on the secondary control connection to the host on the primary control connection.

Third party file transfers depend on the support of the FTP protocol PASV command by the server on the secondary control connection.

5.8.8 Specifying Auto-Login Parameters in .netrc

By default, the auto-login process (when enabled), attempts to log you in to a remote host using the same login name as you are using on the local machine. Often, your account on another machine uses a different login name. If this is the case, you may either log in using the **user** command, or specify alternate login information in the **.netrc** file. This file resides in your home directory. When you specify a machine name and login information in the **.netrc** file, that information is used to log you in whenever you connect to that host.

Information in the **.netrc** file consists of tokens, which are separated by spaces, tabs, or newlines. When an attempt is made to connect to a particular host, **ftp** searches the file until it finds a **machine** token with the correct name, and then processes all tokens following it until it reaches the end of the file, or another **machine** token. Allowed tokens are:

machine *name*

The token **machine** is followed by the name of a remote host. During login, if a **machine** token followed by the name of the host you are logging into is found, all following tokens are read until another **machine** token or an end-of-file are is found.

login *name*

The **login** token is followed by your login name on that host.

password *string*

The **password** token is used to supply a password, if needed.

account *string*

The **account** token is used to supply an account password, if needed.

macdef *name*

The **macdef** token allows you to define a macro during the login process, and functions 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 a macro named **init** is defined, it is automatically executed as the last step in the auto–login process.

Glossary

Terms in **bold** type that appear in the definitions are also defined in this glossary.

Address

A series of numbers that uniquely identifies a **node** on a **network**. Each **network** and **protocol** family has its own **address** format.

Apollo Token Ring (ATR) network

A 12–megabit–per–second **LAN** developed by Apollo Computer Inc. The **network** access method is a token passing scheme. A special bit pattern, a token, is always circulating around the ring. Any **node** may claim a free token and append a message thereby transmitting a packet on the ring. When the target node receives the packet it sets a bit pattern acknowledging receipt. When the packet returns to the sender, the sender removes the packet and returns a free token to the ring.

Bridge

A link between two similar **networks**, formed by creating a third **network** connecting a single **node** on each of the first two **networks**. Two or more **networks** connected together form an **internet**.

DARPA

The Defense Advanced Research Projects Agency of the U.S. Department of Defense.

ETHERNET

A 10–megabit–per–second **local area network**, developed by Digital Equipment Corporation, Intel, and Xerox Corporation, upon which the **IEEE 802.3 network** is based. **ETHERNET** uses coaxial cable as its transmission medium and operates at 10 megabits per second. The **network** uses an access method commonly called CSMA/CD (Carrier Sense Multiple Access with Collision Detection). Each station monitors the **network** and can transmit a message at any time that no other stations are transmitting. If several stations transmit messages simultaneously, the messages collide on the medium. Then each station waits for a random period before transmitting the message again.

File Transfer Protocol (FTP)

A **protocol** for transmitting files between **host** computers. **FTP** is defined by **DARPA**. **FTP** uses **TCP** and **IP** as underlying **protocols**.

FTP

See **File Transfer Protocol**.

Gateway

A link between two dissimilar **networks**, formed when a single **node** is directly connected to each of the two **networks**. A **gateway** must perform **protocol** translation between the **networks** it connects. Two or more **networks** connected together form an **internet**.

Heterogeneous network

A **network** composed of dissimilar **host** computers, such as those of different manufacturers. See **homogeneous network** for contrast.

Homogenous network

A **network** composed of similar **host** computers, such as those of one model or one manufacturer. See **heterogeneous network** for contrast.

Hop

A packet's passage through a routing **node** on its way to its final destination. The number of **hops** in a route is the same as the number of **gateways** a packet passes through.

Host

A computer or workstation that communicates over a **network**. A **host** can both initiate communications and respond to communications that are addressed to it.

Hostid

A **host's Internet address**.

Hostname

A name you can use in place of a **hostid** to refer to a **host**.

IEEE

The Institute of Electrical and Electronics Engineers. A national association, whose activities include publishing standards applicable to various electronic technologies. The **IEEE** technical committees are numbered and grouped by area. For example, the 800 committees study **local area network** technologies.

IEEE 802.3

A standard formulated by the Institute of Electrical and Electronics Engineers. The standard defines the access method and physical layer specifications for **networks** that use a CSMA/CD **protocol**. See **ETHERNET** for a description of how these **networks** operate.

Internet

Two or more connected **networks** that may or may not use the same communication **protocol**. The device that connects the **networks** may perform routing and/or **gateway** functions. A **TCP/IP Internet** conforms to the **Internet Protocol** and the **Transmission Control Protocol**.

Internet address

1. An **address** that conforms to the **DARPA**–defined **Internet protocol**. A unique, four–byte number that identifies a **host** or **gateway** on the **Internet**, consisting of a **network** number followed by a **host** number. The **host** number can be further divided into a subnet number. **Internet addresses** are expressed as four decimal numbers, ranging between 0–255 and separated by periods.

2. An **address** that uniquely identifies a destination on an **internet**.

Internet Protocol (IP)

A **protocol** defined by the Defense Advanced Research Projects Agency (**DARPA**) for connecting **networks** through **gateways**.

IP

See **Internet Protocol**.

LAN

See **Local Area Network**.

Local Area Network (LAN)

A communications **network** linking a number of devices that are located within a relatively short distance, typically less than a mile.

Local network

The **network** to which a **node** is directly attached.

Local node

The **node** executing the commands. For example, the processes created by the Domain **crp** command execute on the node specified in the **-on** option. For contrast, see **remote node**.

Network

Transmission media and software that links **nodes** and peripherals.

Node

Any point in a **network** where services are provided or communications channels are interconnected. Domain **nodes** include workstations and server processors.

Port

A software access point between a **network** controller and the rest of the system. Data exits or enters the system through the **network port**.

Protocol

A set of rules that governs the procedures used in exchanging information between two communication processes.

Remote

Not directly connected or processed at another location.

Remote node

A **node** other than the **node** executing commands.

TCP

See **Transmission Control Protocol**.

TCP/IP

Transmission Control Protocol/Internet Protocol. An acronym used to refer to the **TCP** and **IP** protocols and related Internet protocols, such as **FTP** and **TELNET**, defined by **DARPA**.

TELNET protocol

A **remote** terminal emulation **protocol** defined by the Defense Advanced Research Projects Agency for internetwork communications. **TELNET** uses **TCP** and **IP** as underlying **protocols**.

Transmission Control Protocol (TCP)

A **protocol** for sending datagrams from one **network** to another. It was defined by the Defense Advanced Research Projects Agency for internetwork communications.

Index

Symbols are listed at the beginning of the index. Entries in color indicate task-oriented information.

Reader's Response

Please take a few minutes to send us the information we need to revise and improve our manuals from your point of view.

Document Title: *Using TCP/IP Network Applications*
Order No.: 008667-A00
Date of Publication: July, 1988

What type of user are you?

____ System programmer; language ____________________
____ Applications programmer; language ____________________
____ System maintenance person
____ System Administrator
____ Manager/Professional
____ Technical Professional
____ Student
____ Novice
____ Other

How often do you use the Apollo system?____________________

What additional information would you like the manual to include?_____
____
____

Please list any errors, omissions, or problem areas in the manual by page, section, figure, etc.____________________________________
____
____

____
Your Name   Date

____
Organization

____
Street Address

____
City   State   Zip

No postage necessary if mailed in the U.S.

cut or fold along dotted line

fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

APOLLO COMPUTER INC.
Technical Publications
P.O. Box 451
Chelmsford, MA 01824

fold

Reader's Response

Please take a few minutes to send us the information we need to revise and improve our manuals from your point of view.

Document Title: *Using TCP/IP Network Applications*
Order No.: 008667-A00
Date of Publication: July, 1988

What type of user are you?

____ System programmer; language ____________________
____ Applications programmer; language ____________________
____ System maintenance person
____ System Administrator
____ Manager/Professional
____ Technical Professional
____ Student
____ Novice
____ Other

How often do you use the Apollo system?____________________

What additional information would you like the manual to include?_____

____

____

Please list any errors, omissions, or problem areas in the manual by page, section, figure, etc.______________________________

____

____

____
Your Name Date

____
Organization

____
Street Address

____
City State Zip

No postage necessary if mailed in the U.S.

cut or fold along dotted line

fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 78 CHELMSFORD, MA 01824

POSTAGE WILL BE PAID BY ADDRESSEE

APOLLO COMPUTER INC.
Technical Publications
P.O. Box 451
Chelmsford, MA 01824

fold

apollo

Using TCP/IP Network Applications

008667-A00