Transferring Files Between Hosts [4]

Table 2 shows the file transfer utilities that UNICOS TCP/IP supports.

Table 2. Functions of TCP/IP utilities for file transfer

Utility	Function
`rcp`(1)	Copies files between operating systems based on the UNIX operating system (much like `cp`(1)).
`ftp`(1B)	General-purpose file transfer utility with an interactive interface. This method provides the fastest transfer rates.
`tftp`(1B)	Provides a very limited file access mechanism.

In addition, the network file system (NFS) remote file system feature allows you to directly access data in remote files. This method provides the most convenient access to remote file data.

Before using the file transfer utilities, set up authorization files according to the instructions provided in section 7, page 77. The `rcp` utility accesses your local host's `/etc/hosts.equiv` file or your `$HOME/.rhosts` file and/or both. The `ftp` utility and `rexec`(3) library routine access your `$HOME/.netrc` file. (If your `.rhost` file is world-writeable, this access will not work.) The `ftp` utility and `rexec`(3) library routine access your `$HOME/.netrc` file. `ftp` also uses the `/etc/ftpusers` and `/etc/shells` files at the remote system.

The following subsections explain the capabilities of each utility and describe how to use them.

Using the rcp utility
4.1

The rcp utility has the following features:

- Automatically logs you in to the remote host.
- Copies files between a Cray Research system and a remote host, or between two remote hosts (known as *third-party copying*).

  Note: When the UNICOS multilevel security (MLS) feature is enabled, you can use the rcp utility only if the following requirements are met:

  - The NETW_RCMD_COMPAT configuration parameter is disabled. Check with your system administrator.
  - The client host is named in the /etc/host.equiv file.
  - Your remote and local user IDs are identical.
  - Your user ID is specified in the .rhosts file of the server.
  - The client host is also specified in the .rhosts file of the server.
  - The workstation access list (WAL) specifies rcp permission for one or more of the following: the remote host you want to access, your account name, or your group ID entry.

You can use the rcp command to copy one file into another file or to copy multiple files into a directory. The command syntax is as follows:

```
rcp [-p] [-r] file1 file2
```

-p	Preserves in its copies the modification times and access modes of the source files, ignoring the user file-creation mode mask (see umask(1)).
-r	Tells rcp to copy each subtree of the directory if *file1* is a directory. In this case, *file2* must be a directory.
file1	The name of the file or directory you want to copy. See subsection 4.1.1 for additional information.
file2	The name of the file or directory to which you want to copy. See subsection 4.1.1 for additional information.

If the remote host does not support `rcp`, you can use the `ftp` utility as an alternative.

Specifying file names 4.1.1

Local and remote file names are specified as described in the following subsections.

Local file names 4.1.1.1

Specify the full path name of the file or the path name relative to the current directory. A local file name cannot contain a colon (:) unless the path name has a slash (/) at some point prior to the first colon (see example 8, page 40).

Remote file names 4.1.1.2

You can specify remote file names in the following ways:

- You can use shell metacharacters to specify a remote file name.
- You can specify files on a remote host, as follows:

  hostname:*filename*

hostname	Either the official name, alias, or Internet address of the remote host.
filename	Path name of the file. If *filename* is not a full path name, `rcp` interprets it relative to your home directory on *hostname*.

- When the account login name differs from your login name on the Cray Research system, you can specify file names on a remote host as follows:

  user@*hostname*:*filename*

user	Login name associated with the account
hostname	Official host name, alias, or Internet address of the remote host
filename	Path name of the file (full or relative to the home directory)

Remember to use an at-sign (@) symbol between the *user* and *hostname* specifications, and a colon (:) between the *hostname* and the *filename*.

`rcp` *utility examples* 4.1.2

The `rcp` utility is simple to use because one command completes numerous types of copies. Therefore, instead of providing step-by-step instructions, the `rcp` utility is described through examples.

Example 1:

To copy file `proposal`, which is on the Cray Research system, into file `report` on remote host `chemistry`, type the following command line and press RETURN.

```
$ rcp proposal chemistry:report
$
```

After the command is executed, the Cray Research prompt appears, indicating that the copy to `chemistry` was successful.

Example 2:

To copy file `whale`, which is on the remote host `biology`, into file `mammal` on the Cray Research system, type the following command line and press RETURN:

```
$ rcp biology:whale mammal
$
```

Again, the Cray Research prompt appears after a successful copy is completed.

Example 3:

If you want to copy a file to or from an account with a login name that differs from your login name on the Cray Research system, review the following example.

In this example, the remote file `letter` must be accessed under the login name `tami`. To copy the file from remote host `engineering` into a file called `memo` on the Cray Research system, type the following command line and press RETURN:

```
$ rcp tami@engineering:letter memo
$
```

Example 4:

To copy files between two remote hosts (same login names on the hosts), specify the host names and file names, separated with a colon. For instance, to copy the file `proposal` on host `biology` into the file `report` on host `chemistry`, type the following command line and press RETURN:

```
% rcp biology:proposal chemistry:report
%
```

Example 5:

To copy all of the files that have names that begin with `h2o` from remote host `chemistry` into your working directory on the Cray Research system, type the following command line and press RETURN:

```
$ rcp chemistry:"h2o*" .
$
```

The dot (.) designates the destination to be your working directory on the local host. The `h2o*` part of the source name designates all files that have names that begin with `h2o*`. Because the desired files are so named on the remote host, double quotation marks are necessary around this part of the name to prevent the local host from trying to interpret the * by substituting the names of any local files with similar names.

Example 6:

To copy the entire contents of local directory `work` to a directory with the same name in your home directory on remote host `eng`, type one of the following command lines and press RETURN:

```
$ rcp -r work eng:.
$
$ rcp -r work eng:
$
```

Example 7:

To copy multiple files on remote host `bio` into the `task` directory on the Cray Research system, type the following command line and press RETURN:

```
$ rcp bio:measure bio:data bio:facts task
$
```

Example 8:

To copy the local file `foo:bar`, you cannot specify the local file name in the usual manner because `rcp` interprets the colon as a separator and, therefore, interprets the operand as file `bar` on machine `foo`. The following example shows what happens if you specify the local file name in the usual manner:

```
$ rcp foo:bar chemistry:
foo: unknown host
$
```

In this case, the local file name must be specified as a full path name, or a path name relative to the current directory. Either of the following examples causes `rcp` to recognize `foo:bar` as the local file name:

```
$ rcp ./foo:bar chemistry:
$
$ rcp /usr/tami/foo:bar chemistry:
$
```

Using the `ftp` utility

4.2

The `ftp` utility offers the following features:

- Provides a full range of interactive file manipulation capabilities, such as copying, deleting, and appending.

- Automatically logs you in to the remote host if you have a `.netrc` file in your home directory on the Cray Research system (see subsection 7.2.2, page 83), or if Kerberos is used in conjunction with `ftp`.
- Communicates with all hosts on your network, regardless of the operating system.
- Offers a helpful, prompt-driven command mode that assists you in transferring files to and from remote hosts.

The syntax of the `ftp` utility is as follows:

```
ftp    [-c copybufsize] [-d] [-g] [-i] [-n] [-s  sockbufsize]
       [-t] [-v] [-Sc tos] [-Sd tos] [host [port]]
```

`-c` *copybufsize*
: Sets the copy buffer size. A numeric argument sets the buffer to that size. The letter `K` or `k` can follow a numeric buffer size to specify a multiple of 1024.

`-d`
: Enables debugging.

`-g`
: Disables file name globbing; that is, metacharacters (`*`, `?`, `[...]`) in file names are not expanded to the string they represent.

`-i`
: Turns off interactive prompting during multiple file transfers.

`-n`
: Tells `ftp` to ignore the `.netrc` file. This disables autologin.

`-s` *sockbufsize*
: Sets the socket buffer size. A numeric argument sets the buffer to that size. The letter `K` or `k` can follow a numeric buffer size to specify a multiple of 1024.

`-t`
: Enables tracing.

`-v`
: Turns off verbose mode, suppressing all responses from the remote server, including data transfer statistics. Verbose mode is set on by default.

`-Sc` *tos* Sets the Internet Protocol (IP) type-of-service (TOS) option for the `ftp` control connection to the value *tos*, which may be a numeric type of service value or a symbolic TOS name that is found in the `/etc/iptos` file.

`–Sd` *tos* Sets the Internet Protocol (IP) type-of-service (TOS) option for the `ftp` data connection to the value *tos*, which may be a numeric TOS value or a symbolic TOS name that is found in the `/etc/iptos` file.

host[*port*] Specifies the host or port with which `ftp` will communicate.

The `ftp` utility requires that you supply your login name and password to the remote system, unless Kerberos is used in conjunction with `ftp`. You can do this in three ways:

- Supply your login name and password each time `ftp` prompts you.
- Execute the `ftp user` command.
- Create a `.netrc` file that forwards your login name and password for autologin. For information on setting up this file, see subsection 7.2.2, page 83.

Common `ftp` *functions* 4.2.1

This subsection describes some of the common uses of the `ftp` utility. To use `ftp` as described in this subsection, you must have an account on the remote host that you are accessing, and you must create a `.netrc` file for autologin.

To use the `ftp` utility, type `ftp` and press RETURN. The `ftp` prompt (`ftp`) appears.

```
$ ftp
ftp>
```

At this point, you can execute any `ftp` commands that do not require a connection to a remote host (for example, `open` or `help`). The `ftp` commands are described in subsection 4.2.2, page 54.

Logging in to a remote host
4.2.1.1

To log in to a remote host from a Cray Research host, type `ftp`, followed by the name of the remote host to which you want to connect and press RETURN. You can use the official host name, an alias, or the Internet address to identify the remote host. (If you have already accessed the `ftp` utility, type `open` and then the remote host name at the `ftp>` prompt.) In the following example, the remote host name is `myhost`, and the login name is `mylogin`.

```
$ ftp myhost
Connected to myhost.
220 myhost FTP server (Version 4.15 Sat Nov 7 15:24:41 PST 1987)
ready.  Name (myhost:mylogin):
```

If `mylogin` is the correct account name on the remote host to which you want to connect, you can simply press RETURN. Otherwise, type in the name of the appropriate account and press RETURN. In the following example, the appropriate account name is `othername`. After you enter the account name, the remote host continues, as follows:

```
Name (myhost:mylogin): othername
331 Password required for othername.
Password:
```

Type the password for the account and press RETURN. You will receive confirmation that you are logged in, and the `ftp` prompt will appear, as follows:

```
Password:
230 User othername logged in.
ftp>
```

You are now logged in.

Copying a file from a remote host
4.2.1.2

To copy a file from a remote host to your working directory on the Cray Research system, type the `get` command and the name of the file you want to copy. If you want the file to have a different name in your Cray Research directory, type the new

name after the name of the file you want to copy. In the following example, `rem_file` is the name of the file you want to copy, and `loc_file` is the name by which the file will be known in your directory.

```
ftp> get rem_file loc_file
200 PORT command okay.
150 Opening data connection for rem_file (84.0.194.5,1038) (18 bytes).
226 Transfer complete.
local: loc_file  remote: rem_file
18 bytes received in 0.031 seconds (0.57 Kbyte/s)
ftp>
```

Copying multiple files
4.2.1.3

To copy multiple files from a remote host to your working directory on the Cray Research system, type the `mget` command and the names of the files you want to copy. The interactive prompt will ask you to verify the transfer of each file. You can toggle prompting on or off. If you do not want to be prompted for each file, follow the steps described for the use of the `ftp prompt` command, page NO TAG, or use the `-i` option when you invoke `ftp`.

The following example shows how the transfer occurs with the interactive prompt enabled. The interactive prompt lets you change your mind and not transfer one or more of the files designated on the command line.

```
ftp>  mget file1 file2 file3
mget file1? y
200 PORT command okay
150 Opening data connection for file1 (84.0.194.5,1043) (57 bytes).
226 Transfer complete.
local: file1 remote: file1
57 bytes received in 0.017 seconds (3.3 Kbyte/s)
mget file2? n
mget file3? y
200 PORT command okay
150 Opening data connection for file3 (84.0.194.5,1045) (60 bytes).
226 Transfer complete.
local: file3 remote: file3
60 bytes received in 0.063 seconds (0.94 Kbyte/s)
ftp>
```

Copying files to a remote host
4.2.1.4

To copy a file from the Cray Research system to your home directory on a remote host, type the `put` command and the name of the file you want to copy. In the following example, only the local file name (`file1`) is specified; therefore, the copy retains the original file name.

```
ftp> put file1
200 PORT command okay.
150 Opening data connection for file1(84.0.194.5,1059).
226 Transfer complete.
local: file1 remote: file1
60 bytes received in 0.00098 seconds (60 Kbyte/s)
ftp>
```

You also can use the `mput` command to copy multiple files from the Cray Research system. It works like `mget`, except that files are moving from the Cray Research system, rather than to the Cray Research system.

Appending files
4.2.1.5

To append a local (Cray Research system) file to a remote file, type the `append` command followed by the local file name and the remote file name, and press RETURN, as in the following example. In both cases, the file name can be full or relative.

```
ftp> append crayfile remfile
200 PORT command okay.
150 Opening data connection for remfile (84.0.194.5,1067).
226 Transfer complete
local: crayfile remote: remfile
60 bytes sent in 0.001 seconds (56 Kbyte/s)
ftp>
```

Deleting files
4.2.1.6

To delete a file from the remote host, type the `delete` command and the path name (full or relative) of the file you want to delete and press RETURN, as follows:

```
ftp> delete my_file
200 DELE command okay.
ftp>
```

You can use the `mdelete` command to delete multiple files, as in the following example. The `ftp` utility prompts you to confirm the deletion of each file, unless you execute the `ftp prompt` command prior to the `mdelete` command or specify the `-i` option when `ftp` is invoked.

```
ftp> mdelete file1 file2
delete file1? y
200 DELE command okay.
delete file2? y
200 DELE command okay.
ftp>
```

Defining macros
4.2.1.7

The `macdef` command lets you define macros within `ftp`. The macros also can be put into the `.netrc` file. The following example illustrates the use of the `macdef` command. Macros `doall` and `mrmdir` are defined. The `doall` macro issues the `pwd` and `dir` commands when executed. The `mrmdir` macro removes specified directories when executed.

```
$ ftp
ftp> open biology
Connected to biology.
220 biology FTP server (Version 4.3 Fri Dec 9 17:36:01 CST 1988)
ready.  331 Password required for bonnie.
Password:
230 User bonnie logged in.
ftp> macdef doall
Enter macro line by line, terminating it with a null line
pwd
dir

ftp> $doall
pwd
257 "/usr/bonnie" is current directory.
dir
200 PORT command successful.
150 Opening data connection for /bin/ls (84.0.194.5,1076) (0 bytes).
total 14
drwxr-x---   2 bonnie   grpA       288 Nov 28 14:25 X
drwx------   2 bonnie   grpA        64 Dec 12 11:31 a
drwx------   2 bonnie   grpA        64 Dec 12 11:31 b
drwxrwxrwx   2 bonnie   grpA       160 Dec  5 11:19 bin
drwx------   2 bonnie   grpA        64 Dec 12 11:30 c
drwx------   2 bonnie   grpA        64 Dec 12 11:31 d
drwxr-x---   2 bonnie   grpA       480 Nov  9 12:41 ip
drwxr-x---   2 bonnie   grpA       448 Aug 16 17:17 perf
drwxr-x---   2 bonnie   grpA       992 Oct  6 09:30 sim
drwx------   2 bonnie   grpA       544 Nov 11 18:50 socket
drwxr-x---   3 bonnie   grpA       384 Dec  6 13:47 src
drwxr-xr-x   2 bonnie   grpA       128 Aug 25 09:42 stress
drwxr-x---   6 bonnie   grpA       480 Dec  9 12:35 temp
drwxr-x---   9 bonnie   grpA       480 Oct 18 16:18 test
226 Transfer complete.
819 bytes received in 0.072 seconds (11 Kbyte/s)

ftp>
ftp> macdef mrmdir
Enter macro line by line, terminating it with a null line
rmdir $i

ftp> $mrmdir a b c d
rmdir a
250 RMD command successful.                                    (continued)
```

```
rmdir b
250 RMD command successful.
rmdir c
250 RMD command successful.
rmdir d
250 RMD command successful.
ftp> dir
200 PORT command successful.
150 Opening data connection for /bin/ls (84.0.194.5,1077) (0 bytes).
total 10
drwxr-x---   2 bonnie   grpA       288 Nov 28 14:25 X
drwxrwxrwx   2 bonnie   grpA       160 Dec  5 11:19 bin
drwxr-x---   2 bonnie   grpA       480 Nov  9 12:41 ip
drwxr-x---   2 bonnie   grpA       448 Aug 16 17:17 perf
drwxr-x---   2 bonnie   grpA       992 Oct  6 09:30 sim
drwx------   2 bonnie   grpA       544 Nov 11 18:50 socket
drwxr-x---   3 bonnie   grpA       384 Dec  6 13:47 src
drwxr-xr-x   2 bonnie   grpA       128 Aug 25 09:42 stress
drwxr-x---   6 bonnie   grpA       480 Dec  9 12:35 temp
drwxr-x---   9 bonnie   grpA       480 Oct 18 16:18 test
226 Transfer complete.
595 bytes received in 0.082 seconds (7.1 Kbyte/s)
ftp> quit
221 Goodbye
$
```

Connecting to two hosts
4.2.1.8

You can use the `proxy` command to connect to two different hosts in the same `ftp` session. In the following example, the user connects to host `biology` and then uses the `proxy` command to connect to host `chemistry` in the same `ftp` session. The `proxy` command is used to issue other `ftp` commands to host `chemistry` during the `ftp` session.

```
$ ftp
ftp> open biology
Connected to biology.
220 biology FTP server (Version 4.3 Fri Dec 9 17:36:01 CST 1988) ready.
331 Password required for bonnie.
Password:
230 User bonnie logged in.
ftp> proxy open chemistry
Connected to chemistry.
220 chemistry FTP server (Version 4.1 Fri Nov 4 22:53:26 CST 1988) ready.
Name (chemistry:bonnie):
331 Password required for bonnie.
Password:
230 User bonnie logged in.
ftp> dir
biology:200 PORT command successful.
biology:150 Opening data connection for /bin/ls (84.0.194.5,1065) (0 bytes)
total 10
drwxr-x---   2 bonnie   grpA       288 Nov 28 14:25 X
drwxrwxrwx   2 bonnie   grpA       160 Dec  5 11:19 bin
drwxr-x---   2 bonnie   grpA       480 Nov  9 12:41 ip
drwxr-x---   2 bonnie   grpA       448 Aug 16 17:17 perf
drwxr-x---   2 bonnie   grpA       992 Oct  6 09:30 sim
drwx------   2 bonnie   grpA       544 Nov 11 18:50 socket
drwxr-x---   3 bonnie   grpA       384 Dec  6 13:47 src
drwxr-xr-x   2 bonnie   grpA       128 Aug 25 09:42 stress
drwxr-x---   6 bonnie   grpA       480 Dec  9 12:35 temp
drwxr-x---   9 bonnie   grpA       480 Oct 18 16:18 test
biology:226 Transfer complete.
595 bytes received in 0.038 seconds (15 Kbyte/s)
ftp> proxy dir
chemistry:200 PORT command okay.
chemistry:150 Opening data connection for /bin/ls (84.0.194.5,1066) (0 bytes)
total 40
drwxr-xr-x   2 bonnie   grpA        512 Aug  5 16:27 conf
drwxr-xr-x   2 bonnie   grpA        352 Aug 17 11:07 csim
-rwxr-xr-x   1 bonnie   grpA     132056 Nov 28 07:02 mkhsxdev
-rw-r-----   1 bonnie   grpA       7950 Nov 28 07:01 mkhsxdev.c
drwxr-xr-x   2 bonnie   grpA        576 Nov  2 09:48 perf
drwxr-xr-x   3 bonnie   grpA         96 Dec  2 12:34 src
drwxr-xr-x   2 bonnie   grpA        128 Sep 22 09:29 stats
chemistry:226 Transfer complete.                                    (continued)
```

```
432 bytes received in 0.083 seconds (5.1 Kbyte/s)
ftp> proxy binary
chemistry: Type set to I.
ftp> proxy get X/benchmark.tar X/benchmark.tar
chemistry: 227 Entering Passive Mode (128,162,62,1,18,81)
biology: 200 Type set to I.
biology: 200 PORT command.successful.
biology: 150 Opening BINARY mode data connection for X/benchmark.tar (974848
bytes)
chemistry:150 Opening BINARY mode data connection for benchmark.tar
chemistry:226 Transfer complete.
biology:226 Transfer complete.
biology:200 Type set to A.
local: benchmark.tar remote: X/benchmark.tar
ftp> clo
biology:221 Goodbye.
ftp> proxy clo
chemistry:221 Goodbye.
ftp> quit
$
```

Closing the `ftp` *connection*
4.2.1.9

To close the `ftp` connection, type either the `bye` or the `quit` command and press RETURN.

```
ftp> bye
221 Goodbye.
$
```

Extended `ftp` *example*
4.2.1.10

The following extended example further illustrates the use of the `ftp` utility.

Example:

In this example, Adam is the user, and his login ID is `adam`. He is currently logged in to a Cray Research system, but he must access files that are not on the local system.

- First, he types `ftp` and the remote host name `moon` and presses RETURN.

```
$ ftp moon
Connected to moon.
220 moon FTP server (Version 4.7 Sun) ready.
331 Password required for adam.
Password:
230 User adam logged in.
ftp>
```

- Next, Adam turns off the interactive prompt.

```
ftp> prompt
Interactive mode off.
ftp>
```

- Adam is now ready to copy the files from host `moon`. Because all of the file names begin with the prefix `graphix.`, Adam uses file name globbing to copy all of the files at once.

```
ftp> mget graphix.*
200 PORT command okay.
150 Opening data connection for graphix.1 (84.0.194.5,1066)(45 bytes).
226 Transfer complete.
local: graphix.1 remote: graphix.1
45 bytes received in 0.16 seconds (0.27 Kbyte/s)
200 PORT command okay.
150 Opening data connection for graphix.2 (84.0.194.5,1067)(45 bytes).
226 Transfer complete.
local: graphix.2 remote: graphix.2
45 bytes received in 0.29 seconds (0.15 Kbyte/s)
200 PORT command okay.
150 Opening data connection for graphix.3 (84.0.194.5,1068)(45 bytes).
226 Transfer complete.
local: graphix.3 remote: graphix.3
45 bytes received in 0.49 seconds (0.09 Kbyte/s)
ftp>
```

- Adam closes the connection with host `moon`.

```
ftp> close
221 Goodbye.
ftp>
```

- Next Adam opens a connection with host `saturn`.

```
ftp> open saturn
Connected to saturn.
220 saturn FTP server (Version 4.7 Sun) ready.
331 Password required for adam.
Password:
230 User adam logged in.
ftp>
```

- Adam wants to copy a file called `oct.10` on the Cray Research system to directory `results` on host `saturn`. To do this, he uses the `ftp` command `cd` to change from his home directory on `saturn` to the `results` directory, and then he copies the file, as follows:

```
ftp> cd results
200 CWD command okay.
ftp> put oct.10
200 PORT command okay.
150 Opening data connection for oct.10 (84.0.194.5,1066).
226 Transfer complete.
local: oct.10 remote: oct.10
232 bytes sent in 0.0011 seconds (2e+02 Kbyte/s)
ftp>
```

- Adam wants to return to his home directory on `saturn`; however, he does not remember the name of the appropriate command, so he types `help` and presses RETURN.

```
ftp> help

Commands can be abbreviated.  Commands are as follows:

!       cr          macdef      proxy       send
$       delete      mdelete     sendport    status
account debug       mdir        put         struct
append  dir         mget        pwd         sunique
ascii   disconnect  mkdir       quit        tenex
bell    form        mls         quote       trace
binary  get         mode        recv        type
bye     glob        mput        remotehelp  user
case    hash        nmap        rename      verbose
cd      help        ntrans      reset       ?
cdup    lcd         open        rmdir
close   ls          prompt      runique

ftp>
```

Adam types `cdup` and then lists the names of the files in his directory to make sure that there is no other file he must access. Finally, he closes the `ftp` connection.

```
ftp> cdup
200 CWD command okay.
ftp> ls
200 PORT command okay.
150 Opening data connection for /bin/ls (84.0.194.5,1068)
   (0 bytes).
hypothesis
results
226 Transfer complete
147 bytes received in 0.24 seconds (0.61 Kbyte/s)
ftp> bye
221 Goodbye
$
```

`ftp` *commands*
4.2.2

The following lists the `ftp` commands and describes their functions. You can abbreviate command names to the minimum number of characters required to identify the command uniquely. In the following list, brackets enclose the part of a command name that you can omit.

`!` [*command* [*arguments*]]

Returns you to the shell on the Cray Research system. To return to `ftp` from the shell, press CONTROL-d. You may specify a UNICOS command as an argument to `!`. If you do this, `!` executes your command in the Cray Research system's UNICOS shell and then returns you to `ftp` command mode.

`?` [*ftp_command*]

(Synonymous with the `help` command.) Displays a list of valid `ftp` commands. You can then request help information for a specific command. If you specify a valid `ftp` command as an argument to `?`, `ftp` displays help information on that command.

`$` *macro-name* [*arguments*]

Executes the macro *macro-name*, which was defined with the `macdef` command. Arguments are passed to the macro unglobbed.

<INTERRUPT>

Pressing the interrupt key terminates any `ftp` operation in progress, or, if no operation is in progress, terminates `ftp` itself. You can redefine the interrupt key; by default, it is CONTROL-c.

`account` [*password*]

Supplies a supplemental password required by a remote system for access to resources after a login completes successfully. If you do not specify an argument on the command line, you are prompted for an account password; the password you enter is not echoed on the screen.

`allo` [*argument*]

Specifies whether the `ftp allo` and `ftp size` commands will be sent before `put` and `get` commands are executed. The `ftp allo` command informs the server of the size of the file that will be sent by using the `put` command so that it can preallocate the disk space, if necessary. The `ftp size` command inquires about the size of a file that will be retrieved by using the `get`(1) command, so that the disk space can be preallocated before the file is fetched.

`append` *lfile* [*rfile*]

Appends *lfile* to *rfile*; *lfile* is a valid file name on the local host, and *rfile* is a valid file name on the remote host. If you do not specify a file name for *rfile*, *rfile* is given the same file name as *lfile*. If *rfile* does not exist on the remote host, a new file is created.

`ascii`	Sets the file transfer type to ASCII; ASCII is the default.
`bell`	Toggles the terminal bell setting. Turning on the bell causes the terminal bell to ring when a file transfer is completed. The bell is turned off by default.
`binary`	Sets the file transfer type to binary.
`bye`	(Synonymous with the `quit` command.) Closes the connection to the remote host and exits from the `ftp` program.
`case`	Toggles `mget` uppercase and lowercase ID mapping. The default setting is off.
`cd` *rdir*	Changes the working directory on the remote host to *rdir*; *rdir* is a valid working directory on the remote host. The initial working directory on the remote system is the home directory of the user.
`cdup`	Changes the remote host's working directory to the parent of the working directory.
`chmod` *remote–file*	Changes file permissions on a remote file.
`clear`	Sets the file protection level to clear text for file transfer. All commands sent over the control channel are protected by cryptographic checksum until a different protection level is specified.
`close`	Closes the connection to the remote host but does not exit the `ftp` program. After you are disconnected from the remote host, you can open a connection to a new host by using the `open` command.
`copybuf` [*argument*]	Sets the copy buffer size. This buffer is used for reading and writing between the data socket and the source or destination file. `ftp` automatically chooses a copy buffer size; however, you can alter the selection. `copybuf` takes an optional argument. An argument of `off` turns off copy buffer sizing and the buffer defaults to the size specified in the `tcp_config.h` file by the `COPYBUFSIZE` variable. An argument of `auto` returns the buffer sizing to automatic. A numeric argument sets the buffer to that size. The letter `K` or `k` can follow a numeric buffer size to specify a multiple of 1024. Without an argument, `copybuf` shows the current copy buffering. The default setting is `auto`.

`cr`	Toggles carriage return stripping during ASCII-type file retrieval. The default setting is on.
`debug`	Toggles debugging mode. When debugging is on, `ftp` prints each command sent to the remote machine, preceded by the string —>. The default setting is off.
`delete` *rfile*	Deletes *rfile*; *rfile* is the name of a file on the remote host.
`dir` [*rdir*] [*lfile*]	Writes a listing of all file names in directory *rdir* (*rdir* is a valid working directory on the remote host) to *lfile* (*lfile* is the name of a file on the local host). If you do not specify *rdir*, `dir` lists the contents of your working directory on the remote machine when the command is executed. If you do not specify *lfile*, `dir` displays the listing on the standard output device. The listing that this command produces is in long format and includes the following information about each file: file name or names, length, owner, access permissions, and the date and time of last modification.
`disconnect`	An alias for the `close` command.
`form`	Sets file transfer format to `nonprint`.
`fullbuf`	Toggles the use of full buffers on write-to-disk functions. During a `get` operation, `ftp` fills the copy buffer with reads from the data socket before writing the contents of the buffer to the destination file. You can use the `fullbuf` command to disable this feature.
`get` *rfile* [*lfile*]	(Synonymous with the `recv` command.) Copies *rfile* into *lfile*; *rfile* is the name of a file on the remote host, and *lfile* is the name of a file on the local host. The transfer is done in ASCII mode unless another mode (for example, binary) was specified. If you do not specify *lfile*, `get` copies *rfile* into a new file called *rfile* on the local host. If you do not specify either file name, `get` prompts for both the remote and local file names. If verbose mode is on (see the `verbose` command in this list), various statistics, including the number of bytes transferred, the elapsed time in seconds, and the effective baud rate, are sent to the standard output device when the file transfer is complete. (By default, the standard output device is usually your terminal.)

`glob` — Toggles the status of file globbing. *Globbing* is the process of expanding ambiguous file references. With file name globbing enabled, each local file or path name is processed for the shell metacharacters `*`, `?`, `[`, and `]`; the metacharacters are expanded to match file names. With globbing disabled, all local file and path names are treated literally. File globbing is turned on by default unless you specify the `-g` option on the `ftp` command line; globbing is always on with reference to remote files. Globbing affects only the "m" commands (`mget`, `mput`, and so on).

`hash` — Toggles the status of `hash`. Turning `hash` on causes a # to be displayed on the standard output each time a packet is sent or received during a file transfer. This feature is turned off by default.

`help` [*command*] — (Synonymous with the `?` command.) Prints help information about *command*; *command* is a valid `ftp` command. If you do not specify *command*, `ftp` prints a list of valid commands.

`idle` [*seconds*] — Gets or sets idle timer on the remote side.

`image` — Synonymous with the `binary` command. Sets the file transfer type to binary.

`lcd` [*ldir*] — Changes the working directory on the local host to *ldir; ldir* is a valid working directory on the local host. If you do not specify *ldir*, the user's home directory is used by default.

`ls` [*rdir*] [*lfile*] — Retrieves a listing of the names of all files in directory *rdir*; *rdir* is a valid working directory on the remote host. The command puts the listing into file *lfile* on the local host. If you do not specify *rdir*, `ls` lists the working directory. If you do not specify *lfile*, the listing is displayed on the standard output device, which is usually your terminal. The default listing format resembles the short format of the `ls`(1) command.

`macdef` *macro-name* — Defines a macro. Subsequent lines are stored as the macro *macro-name*. A null line indicates the end of the macro definition.

`mdelete` *rfile* [*rfile2...*] — Deletes files on the remote host; each *rfile* is the name of a file on the remote host. You can specify any number of remote file names (separated by spaces) on the command line.

`mdir` *rfile* [*rfile2...*] *lfile*	Obtains a directory listing of *rfile* and places it in *lfile*; *rfile* is the name of a file on the remote host, and *lfile* is the name of a file on the local host. The listing that this command produces is in long format and includes file name(s), length, owner, access permissions, and the date and time of last modification. You can specify any number of remote file names (separated by spaces) on the command line. You must specify *lfile*.
`mget` *rfile* [*rfile2* ...]	Retrieves a copy of file *rfile* from the remote host and places it in a file of the same name in the working directory on the local host. You can specify any number of remote file names (separated by spaces) on the command line.
`mkdir` *rdir*	Creates a directory called *rdir* on the remote host.
`mls` *rdir1* [*rdir2* ...] *lfile*	Obtains a listing of directory *rdir1* and places it in *lfile*; *rdir1* is the name of a directory on the remote host, and *lfile* is the name of a file on the local host. You can specify any number of remote directory names (separated by spaces) on the command line. The format of the listing generated by this command resembles the short format of the `ls`(1) command. You must specify *lfile*.
`mode`	Sets file transfer mode to stream.
`modtime` *remote-file*	Shows last modification time of a remote file.
`mput` *lfile1* [*lfile2* ...]	Puts a copy of file *lfile1* in a file of the same name in the working directory on the remote host. *lfile* is the name of a file on the local host. You can specify any number of local file names (separated by spaces) on the command line.
`newer` *remote-file* [*local-file*]	Gets file if the remote file is newer than the local file.
`nlist` [*remote-directory* [*local-file*]]	Lists the contents of the remote directory.
`nmap` [*inpattern outpattern*]	Sets or unsets file name mapping. If you omit arguments, the file name mapping mechanism is unset. If you specify arguments, remote file names are mapped during the execution of `mput` commands and `put` commands that were issued without a specified remote target file name. Similarly, local file names are mapped during the execution of `mget` commands and `get` commands that were issued without a specified local target file name.

`ntrans` [*inchars outchars*]	Sets or unsets the file name character translation. If you omit arguments, the file name character translation mechanism is unset. If arguments are specified, characters in remote file names are translated during the execution of `mput` commands and `put` commands that were issued without a specified remote target file name. Similarly, characters in local file names are translated during the execution of `mget` commands and `get` commands that were issued without a specified local target file name.
`open` *rhost*	Opens a connection to *rhost*; *rhost* is a valid name, alias, or Internet address of a remote host.
`private`	Sets the file protection level to `private` for the control channel and file transfer. This command is not available outside of the United States and Canada. All messages sent on the control and data channels are protected by encryption.
`protect [clear | safe | private]`	Sets the protection level for file transfer. The `private` option is not available outside of the United States and Canada.
`prompt`	Toggles the status of interactive prompting. Enabling prompting causes all `mget`, `mput`, and `mdelete` commands to prompt with the names of the files in the working directory. If you respond to this prompt with an `n` or `N`, the specified command skips the file you select. Responding with anything else causes the file to be processed. If your respond with anything other than a `y`, the file is skipped. If prompting is turned off, commands process all files. Prompting is turned on by default unless you specify the `-i` option on the `ftp` command line.
`proxy` *ftp-command*	Executes an `ftp` command (*ftp-command*) on a secondary control connection. Allows simultaneous connection of two remote `ftp` servers for transferring files between servers.
`put` *lfile* [*rfile*]	(Synonymous with the `send` command.) Copies *lfile* into *rfile*; *lfile* is the name of a file on the local host, and *rfile* is the name of a file on the remote host. The transfer is done in ASCII mode unless you specified another mode (for example, binary). If you omit *rfile*, `put` copies *lfile* into a file called *lfile* on the remote host. If verbose mode is on, various statistics (including the number of bytes transferred, the elapsed time in seconds, and the effective baud rate) are printed to the standard output device when the file transfer is complete.
`pwd`	Prints the name of the working directory on the remote host.

`quit`	(Synonymous with the `bye` command.) Closes the connection to the remote host and exits from the `ftp` program.
`quote` *arg1 arg2...*	Passes *arg1* directly to the remote host, without parsing it on the local host. This lets you use `ftp` commands that are implemented by the remote host but not by the local host. To obtain the list of `ftp` commands that the remote host supports, use the `remotehelp` command.
`rawbuf`	Toggles the use of raw I/O on write-to-disk and read-from-disk functions. Usually, the system overhead is lower when raw I/O is used. The default is that raw I/O is enabled.
`recv` *rfile* [*lfile*]	(Synonymous with the `get` command.) Copies *rfile* into *lfile*; *rfile* is the name of a file on the remote host, and *lfile* is the name of a file on the local host. The transfer is done in ASCII mode unless you specify another mode (for example, binary). If you do not specify *lfile*, `recv` copies *rfile* into a new file called *rfile* on the local host. If you do not specify either file name, `recv` prompts for both the remote and local file names. If verbose mode is on (see the command `verbose` in this list), various statistics, including the number of bytes transferred, the elapsed time in seconds, and the effective baud rate, are sent to the standard output device when the file transfer is complete. (By default, the standard output device is usually your terminal.)
`reget` *remote-file* [*local-file*]	Gets file restarting at the end of the local file.
`rename` [*from*][*to*]	Renames the file *from* on the remote host to the file *to*.
`reset`	Clears the reply queue.
`restart` *bytecount*	Restarts the file transfer at *bytecount*.
`rhelp` [*command*]	Displays help information about *command* as it is implemented on the remote host; *command* must be a valid command on the remote host. If you omit *command*, `help` displays all the `ftp` commands that the remote host recognizes.
`rmdir` *rdir*	Deletes directory *rdir*; *rdir* is a valid directory on the remote host.
`rstatus`	Shows the status of the remote machine.

`runique` — Receive unique. Provides a unique file name when a file with the same name as the target local file for a `get` or `mget` command already exists. A `.1` is appended to the name, unless this matches another existing file. The process continues up to `.99`; at that point, an error message appears, and the transfer does not occur. This facility is turned off by default.

`safe` — Sets the file protection level to `safe` for the control channel and file transfer. All messages sent on the control and data channels are protected by cryptographic checksum.

`send` *lfile* [*rfile*] — (Synonymous with the `put` command.) Copies *lfile* into *rfile*; *lfile* is the name of a file on the local host, and *rfile* is the name of a file on the remote host. The transfer is done in ASCII mode unless another mode (for example, binary) was specified. If you omit *rfile*, `send` copies *lfile* into a file called *lfile* on the remote host. If verbose mode is on, various statistics (including the number of bytes transferred, the elapsed time in seconds, and the effective baud rate) are printed to the standard output device when the file transfer is complete.

`sendport` — Toggles the use of the `PORT` protocol command for each data connection.

`showbuf` — Toggles display of copy buffer and socket buffer sizing information during a transfer. Default is no display of information.

`site` *arguments* — Sends *arguments* to the remote `ftp` server as arguments to an `ftp site` command. In return, one `ftp` reply code is expected.

`size` *remote-file* — Shows the size of the remote file.

`sockbuf` [*argument*] — Sets the socket buffer size. This kernel buffer is used for data transfer on the data socket. `ftp` automatically chooses a socket buffer size; however, you can alter the selection. `sockbuf` takes an optional argument. An argument of `off` turns off socket buffer sizing and the buffer defaults to the kernel's default socket buffer size. An argument of `auto` returns buffer sizing to automatic. A numeric argument sets the buffer to that size. The letter `K` or `k` can follow a numeric buffer size to specify a multiple of 1024. Without an argument, `sockbuf` shows the current socket buffering. The default setting is `auto`.

`status` — Shows the current connection status, transfer mode, messages, and the settings for `hash`, `bell`, and other options of the current `ftp` session.

`struct` [*struct-name*] — Sets the file transfer structure to *struct-name*. By default, file structure is used. Currently, only the default is supported.

`sunique` — Send unique. Provides a unique file name when a file with a name equal to the target remote file name for a `put` or `mput` command already exists. A `.1` is appended to the name, unless this matches another existing file. The process continues up to `.99`; at that point, an error message appears, and the transfer does not occur. This facility is off by default.

`system` — Shows the remote system type.

`tenex` — Sets the file transfer type to the type needed to talk to TENEX machines.

`trace` — Toggles packet tracing. The default setting is off.

`type` [*type–name*] — Sets the file transfer type to *type-name*. If *type-name* is not specified, the current type is printed. Valid types are `tenex`, `binary`, and `ascii`; `ascii` is the default.

`umask` *mask* — Gets and sets `umask` to *mask* on the remote side.

`user` *login_name* — Initiates the login process on the remote host; *login_name* is a valid login name on a remote host. The `ftp` program prompts you for your password to complete the login process.

`verbose` — Toggles verbose mode. When verbose mode is on, all responses from the `ftp` server are displayed on your standard output device. Verbose mode is turned on by default.

`winshift` [*value*] — Gets or sets the value for the TCP window shift option to be used on data connections. If no argument is specified, this option reports the current value. If *value* is `off`, the option is disabled; if *value* is `on`, the option is enabled with a value of 4; if *value* is an integer value between 0 and 14, the TCP window shift option is enabled with that value. Because the client side of an `ftp` always performs a passive option operation, you do not have to disable the sending of the TCP window shift option. If the incoming `SYN` packet does not contain the option, none will be sent in the `SYN,ACK` packet, regardless of whether the application has enabled the option.

Using the `tftp` utility

4.3

The `tftp` utility is a very limited file access mechanism. Because this utility poses a security threat to the system, the system administrator might choose to turn it off. Contact your system administrator for information about the availability of and access permissions for `tftp`. Following is an example of `tftp` use.

```
$ tftp
tftp> connect chemistry
tftp> get /usr/bonnie/courses
Received 214 bytes in 0 seconds.
tftp> quit
$
```

For more information, see the `tftp`(1B) man page.