C−KERMIT 8.0 UNIX MANUAL PAGE AND TUTORIAL

Frank da Cruz, Christine M. Gianone
The Kermit Project, Columbia University

D R A F T −− STILL BEING WRITTEN −− C−KERMIT 8.0 NOT RELEASED YET
MANY OF THE LINKS POINT TO C−KERMIT 7.0 MATERIAL UNTIL C−KERMIT 8.0 IS RELEASED.
SOME ITEMS MENTIONED HERE POSTDATE THE LAST BETA.
CORRECTIONS AND SUGGESTIONS WELCOME.

This document is intended to give the beginner sufficient information to make basic (if not advanced) use of C−Kermit 8.0. Although it might be rather long for a Unix manual page (about 1600 lines), it's still far shorter than the C−Kermit manual, which should be consulted for advanced topics such as customization, character−sets, scripting, etc. We also attempt to provide a clear structural overview of C−Kermit's many capabilities, functional areas, states, and modes and their interrelation, that should be helpful to beginners and veterans alike, as well as to those upgrading to the new release.

Most recent update: Wed Oct 31 11:20:59 2001

CONTENTS

DESCRIPTION [Top] [Contents] [Next]

C−Kermit is an all−purpose communications software package from the Kermit Project at Columbia University that:

- Is portable to many platforms, Unix and non−Unix alike.
- Can make both serial and network connections.
- Can conduct interactive terminal sessions over its connection.
- Can transfer text or binary files over the same connection.
- Can convert text−file character sets in terminal mode or file transfer.

- Is customizable in every aspect of its operation.

C−Kermit is a modem program, a Telnet client, an Rlogin client, an FTP client, an HTTP client, and on selected platforms, also an X.25 client. It can make its own secure Internet connections using IETF−approved security methods including Kerberos IV, Kerberos V, SSL/TLS, and SRP and it can also make SSH (Secure Shell) connections through your external SSH client application. It can be the far−end file−transfer or client/server partner of your desktop Kermit client. It can also accept incoming dialed and network connections. It can even be installed as an Internet service on its own standard TCP socket, 1649 [RFC2839, RFC2840].

And perhaps most important, everything you can do "by hand" (interactively) with C−Kermit, can be "scripted" (automated) using its built−in cross−platform transport−independent script programming language, which happens to be identical to its interactive command language.

This manual page offers an overview of C−Kermit 8.0 for Unix ("Unix" is an operating system family that includes AIX, DG/UX, FreeBSD, HP−UX, IRIX, Linux, Mac OS X, NetBSD, OpenBSD, Open Server, Open Unix, QNX, Solaris, SunOS, System V R3, System V R4, Tru64 Unix, Unixware, Xenix, and many others). For thorough coverage, please consult the published C−Kermit manual and supplements (see DOCUMENTATION below). For further information about C−Kermit, Kermit software for other platforms, and Kermit manuals, visit the Kermit Project website:

`http://www.columbia.edu/kermit/`

This is a longer−than−average manual page, and yet it barely scratches the surface. Don't be daunted. C−Kermit is a large and complex package, evolving over decades of practice and experience, but that doesn't mean it's hard to learn or use. Its most commonly used functions are explained here with pointers to additional information elsewhere.

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ]

SYNOPSIS [Top] [Contents] [Next] [Previous]

Usage: `kermit` *`[filename] [−x arg [−x arg]...[−yyy]..] [ {=,−−,+} text ] ]`*
Or: `kermit` *`URL`*

- *`−x`* is an option requiring an argument;
- *`−y`* is an option with no argument.

If the first command−line argument is the name of a file, interactive−mode commands are executed from the file. The '=' (or "−−") argument tells Kermit not to parse the remainder of the command line, but to make the words following '=' available as `\%1`, `\%2`, ... `\%9`. The "+" argument is like "=" but for use in "kerbang scripts" (explained below). A second command−line format allows the one and only argument to be a Telnet, FTP, HTTP, or IKSD URL.

Order of execution:

1. The command file (if any).
2. The initialization file, if any, unless suppressed with `−Y`.
3. The customization file (if it is executed by the initialization file).
4. The command−line URL (if any, and if so, execution stops here).
5. Command−line options (if any).
6. Interactive commands.

Some command−line options can cause actions (such as `−s` to send a file); others just set parameters. If any action options are included on the command line, Kermit exits when finished unless also given the `−S` ("stay") option. If no action options are given, no initialization or command files contained an EXIT or QUIT command, and no fatal errors occurred, Kermit issues its prompt and waits for you to type commands.

> *Bear in mind that C−Kermit can be built with selected features disabled, and also that certain features are not available on all platforms. For example, C−Kermit can't be built with TCP/IP support on a platform that does not have TCP/IP header files and libraries (and even if Kermit does include TCP/IP support, it can't be used to make TCP/IP connections on a computer that does not have a TCP/IP stack installed). If your version of C−Kermit lacks a feature mentioned here, use its SHOW FEATURES command to see what might have been excluded.*

C−Kermit has three kinds of commands: regular single−letter command−line options, extended−format command−line options, and interactive commands.

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ]

OPTIONS [Top] [Contents] [Next] [Previous]

Like most Unix commands, C−Kermit can be be given options on the command line. But C−Kermit also can be used interactively by giving it commands composed of words, which are more intuitive than cryptic command−line options, and more flexible too. In other words, you don't have to use C−Kermit's command−line options, but they are available if you want to. (By the same token, you don't have to use its interactive commands either −− you can use either or both in any combination.)

C−Kermit is generally installed in the PATH as "kermit", and therefore is invoked by typing the word "kermit" (lowercase) at the shell prompt, and then pressing the Return or Enter key. If you wish to include command−line options, put them after the word "kermit" but before pressing Return or Enter, separated by spaces, for example:

```
$ kermit −s ckermit.tar.gz
```

('`$`' is the shell prompt; "`kermit −s ckermit.tar.gz`" is what you type, followed by Return or Enter.)

Here is a list of C−Kermit's single−letter command−line options, which start with a single dash (`−`), in ASCII ("alphabetical") order. Alphabetic case is significant (`−A` is not the same as `−a`). The *Action?* column contains Y for action options and N for non−action options.

Option	Action?	Description
`−0`	N	*(digit zero)* 100% transparent Connect state for "in−the−middle" operation: 8 bits, no parity, no escape character, everything passes through.
`−8`	N	*(digit eight)* Connection is 8−bit clean (this is the default in C−Kermit 8.0). Equivalent to the EIGHTBIT command, which in turn is a shortcut for SET TERMINAL BYTESIZE 8, SET COMMAND BYTESIZE 8, SET PARITY NONE.
`−9` *arg*	N	*(digit nine)* Make a connection to an FTP server. Equivalent to the FTP OPEN command. *Argument:* IP−address−or−hostname*[*`:`optional−TCP−port*]*. NOTE: C−Kermit also has a separate FTP command−line personality, with regular FTP−like command−line syntax. More about this below.

`−A`	N	Kermit is to be started as an Internet service (IKSD) (only from `inetd.conf`).
`−B`	N	Kermit is running in Batch or Background (no controlling terminal). To be used in case Kermit doesn't automatically sense its background status. Equivalent to the SET BACKGROUND ON command.
`−C` *arg*	N	Interactive−mode Commands to be executed. *Argument:* Commands separated by commas, list in doublequotes.
`−D` *arg*	N	Delay before starting to send in Remote mode. Equivalent to the SET DELAY command. *Argument:* Number of seconds.
`−E`	N	Exit automatically when connection closes. Equivalent to SET EXIT ON−DISCONNECT ON.
`−F` *arg*	N	Use an open TCP connection. *Argument:* Numeric file descriptor of open TCP connection. *Also see:* `−j`, `−J`.
`−G` *arg*	Y	Get file(s) from server, send contents to standard output, which normally would be piped to another process. *Argument:* Remote file specification, in quotes if it contains metacharacters. *Also see:* `−g`, `−k`.
`−H`	N	Suppress program startup Herald and greeting.
`−I`	N	Tell Kermit it has a reliable connection, to force streaming to be used where it normally would not be. Equivalent to the SET RELIABLE ON command.
`−J` *arg*	N	"Be like Telnet." Like `−j` but implies `−E`. *Argument:* IP hostname/address optionally followed by service. NOTE: C−Kermit also has a separate Telnet command−line personality, with regular Telnet−like command−line syntax. More about this below.
`−L`	N	Recursive directory descent for files in `−s` option.
`−M` *arg*	N	My user name (for use with Telnet, Rlogin, FTP, etc). Equivalent to the SET LOGIN USER command. *Argument:* Username string.
`−O`	Y	*(Uppercase letter O)* Be a server for One command only. *Also see:* `−x`.
`−P`	N	Don't convert file (Path) names of transferred files. Equivalent to SET FILE NAMES LITERAL.
`−Q`	N	Quick Kermit protocol settings. Equivalent to the FAST command. This is the default in C−Kermit 7.0 and later.
`−R`	N	Remote−only (this just makes IF REMOTE true).
`−S`	N	Stay (enter command parser after action options).
`−T`	N	Force Text mode for file transfer; implies `−V`. Equivalent to SET TRANSFER MODE MANUAL, SET FILE TYPE TEXT.
`−V`	N	Disable automatic per−file text/binary switching. Equivalent to SET TRANSFER MODE MANUAL.

`−Y`	N	Skip (don't execute) the initialization file.
`−a` *arg*	N	As−name for file(s) in `−s`, `−r`, or `−g`. *Argument:* As−name string (alternative filename). When receiving files, this can be a directory name.
`−b` *arg*	N	Speed for serial device. Equivalent to SET SPEED. *Argument:* Numeric Bits per second for serial connections.
`−c`	Y	Enter Connect state before transferring files.
`−d`	N	Create a `debug.log` file with detailed debugging information (a second `−d` adds timestamps). Equivalent to LOG DEBUG but takes effect sooner.
`−e` *arg*	N	Maximum length for incoming Kermit file−transfer packets. Equivalent to SET RECEIVE PACKET−LENGTH. *Argument:* Length in bytes.
`−f`	Y	Send a FINISH command to a Kermit server.
`−g` *arg*	N	Get file(s) from a Kermit server. *Argument:* File specification on other computer, in quotes if it contains metacharacters. Equivalent to GET. *Also see:* `−a`, `−G`, `−r`.
`−h`	Y	Print Help text for single−letter command−line options (pipe thru 'more' to prevent scrolling).
`−i`	N	Force binary (Image) mode for file transfer; implies `−V`. Equivalent to SET TRANSFER MODE MANUAL, SET FILE TYPE BINARY.
`−j` *arg*	N	Make a TCP/IP connection. *Argument:* IP host name/address and optional service name or number. Equivalent to the TELNET command. *Also see:* `−J`, `−F`.
`−k`	Y	Receive file(s) to standard output, which normally would be piped to another process. *Also see:* `−r`, `−G`.
`−l` *arg*	N	*(Lowercase letter L)* Make a connection on the given serial communications device. Equivalent to the SET LINE (SET PORT) command. *Argument:* Serial device name, e.g. `/dev/ttyS0`.
`−m` *arg*	N	Modem type for use with the `−l` device. Equivalent to the SET MODEM TYPE command. *Argument:* Modem name as in SET MODEM TYPE command, e.g. "usrobotics".
`−n`	Y	Enter Connect state after transferring files (historical).
`−p` *arg*	N	Parity. Equivalent to the SET PARITY command. *Argument:* One of the following: e(ven), o(dd), m(ark), n(one), s(pace).
`−q`	N	Quiet (suppress most messages). Equivalent to SET QUIET ON.
`−r`	Y	Receive file(s). Equivalent to the RECEIVE command. *Argument:* (none, but see `−a`)
`−s` *arg*	N	

		Send file(s). *Argument:* One or more local file specifications. Equivalent to the SEND command. *Also see:* `-a`.
`-t`	N	(Historical) Xon (Ctrl−Q) Turnaround character for half−duplex connections (used on serial linemode connections to old mainframes). Equivalent to SET DUPLEX HALF, SET HANDSHAKE XON.
`-v` *arg*	N	Window size for Kermit protocol (ignored when streaming). Equivalanet to SET WINDOW−SIZE. *Argument:* Number, 1 to 32.
`-w`	N	Incoming files Write over existing files. Equivalent to SET FILE COLLISION OVERWRITE.
`-x`	Y	Enter server mode. Equivalent to the SERVER command. *Also see:* `-0`.
`-y` *arg*	N	Alternative initialization file. *Argument:* Filename.
`-z`	N	Force foreground behavior. To be used in case Kermit doesn't automatically sense its foreground status. Equivalent to the SET BACKGROUND OFF command.

Extended command−line options (necessary because single−letter ones are about used up) start with two dashes (`--`), with words rather than single letters as option names. If an extended option takes an argument, it is separated from the option word by a colon (`:`). Extended options include:

Plus several other IKSD−Only options.

See the file−transfer section for examples of command−line invocation.

COMMAND LANGUAGE [Top] [Contents] [Next] [Previous]

C−Kermit's interactive command language is the subject of a 622−page book and another several hundred pages of updates, far too much for a manual page. But it's not hard to get started. At the shell prompt, just type "kermit" to get C−Kermit's interactive command prompt:

```
$ kermit
(/current/directory) C-Kermit>
```

Begin by typing "help" (and then press the Return or Enter key) for a top−level overview, read it, and go from there. Your second command should probably be "intro" (introduction). Note the prompt shows your current directory (unless you tell Kermit to prompt you with something else).

Interactive commands are composed mainly of regular English words, usually in the form of imperative sentences, such as:

```
send oofa.txt
```

which tells Kermit to send (transfer) the file whose name is `oofa.txt`, or:

```
set transfer mode automatic
```

which sets Kermit's "transfer mode" to "automatic" (whatever that means).

While typing commands, you can abbreviate, ask for help (by pressing the "?" key anywhere in a command), complete keywords or filenames (with the Tab or Esc key), and edit your typing with Backspace or Delete, Ctrl−W, Ctrl−U, etc. You can also recall previous commands, save your command history, and who knows what else. Give the INTRO command for details.

C−Kermit has hundreds of commands, and they can be issued in infinite variety and combinations, including commands for:

- Making connections (SET LINE, DIAL, TELNET, SSH, FTP, CONNECT, ...)
- Breaking connections (HANGUP, CLOSE)
- Transferring files (SEND, GET, RECEIVE, MOVE, RESEND, ...)
- Establishing preferences (SET)
- Displaying preferences (SHOW)
- Managing local files (CD, DELETE, MKDIR, DIRECTORY, RENAME, TYPE, ...)
- Managing remote files (RCD, RDEL, RMKDIR, RDIR, ...)
- Using local files (FOPEN, FCLOSE, FREAD, FWRITE)
- Programming (TAKE, DEFINE, IF, FOR, WHILE, SWITCH, DECLARE, ...)
- Interacting with the user (ECHO, ASK, ...)
- Interacting with a remote computer (INPUT, OUTPUT, ...)
- Interacting with local programs (RUN, EXEC, PTY, ...)
- Logging things (LOG SESSION, LOG PACKETS, LOG DEBUG, ...)

And of course QUIT or EXIT to get out and HELP to get help, and for programmers: loops, decision making, variables, arrays, associative arrays, integer and floating point arithmetic, macros, built−in and user−defined functions, string manipulation, pattern matching, block structure, scoping, recursion, and all the rest. To get a list of all C−Kermit's commands, type a question mark (?) at the prompt. To get a description of any command, type HELP followed by the name of the command, for example:

```
help send
```

The command interruption character is Ctrl−C (hold down the Ctrl key and press the C key).

The command language "escape character", used to introduce variable names, function invocations, and so on, is backslash (\). If you need to include a literal backslash in a command, type two of them, e.g.:

```
get c:\\k95\\k95custom.ini
```

Command Files, Macros, and Scripts

A file containing Kermit commands is called a Kermit command file or *Kermit script*. It can be executed with Kermit's TAKE command:

```
(/current/dir) C−Kermit> take commandfile
```

(where "commandfile" is the name of the command file). Please don't pipe a command file into Kermit's standard input (which might or might not work); if you have Kermit commands in a file, tell Kermit to TAKE the file.

In Unix only, a Kermit command file can also be executed directly by including a "kerbang" line as the first line of the file:

```
#!/usr/local/bin/kermit +
```

That is, a top line that starts with "#!", followed immediately by the full path of the Kermit executable, and then, if the Kermit script is to be given arguments on the command line, a space and a plus sign. The script file must also have execute permission:

```
chmod +x commandfile
```

Except for the " +" part, this is exactly the same as you would do for a shell script, a Perl script, etc. Here's a simple but useless example script that regurgitates its arguments (up to three of them):

```
#!/usr/local/bin/kermit +
if defined \%1 echo "Argument 1: \%1"
if defined \%2 echo "Argument 2: \%2"
if defined \%3 echo "Argument 3: \%3"
if defined \%4 echo "etc..."
exit
```

If this file is stored in your current directory as "commandfile", then:

```
./commandfile one two three four five
```

prints:

```
Argument 1: one
Argument 2: two
Argument 3: three
etc...
```

This illustrates the basic structure of a standalone Kermit script: the "kerbang line", then some commands. It should end with "exit" unless you want the Kermit prompt to appear when it is finished. \%1 is the first argument, \%2 the second, and so on.

You can also create your own commands by defining named macros composed of other Kermit commands (or macros). Here's a simple example:

```
define mydial {
    set modem type usrobotics
    set port /dev/ttyS0
    if fail end 1
    set speed 57600
    dial \%1
    if success connect
}
```

This shows how you can combine many commands into one command, "mydial" in this case (you can use any name you like, provided it does not clash with the name of a built−in command). When this macro definition is in effect, you can type commands like:

```
mydial 7654321
```

and it executes all the commands in macro definition, substituting the first operand ("7654321") for the formal parameter ("\%1") in the definition. This saves you from having to type lots of commands every time you want to make a modem call.

One way to have the macro definition in effect is to type the definition at the Kermit prompt. Another way is to store the definition in a file and TAKE the file. If you want the the definition to be in effect automatically every time you start Kermit, put the definition in your initialization or customization file (explained below).

Here's a somewhat more ambitious example:

```
define mydelete {
    local trash
    assign trash \v(home)trashcan/
    if not defined \%1 end 1 "Delete what?"
    if wild \%1 end 1 "Deleting multiple files is too scary"
    if not exist \%1 end 1 "I can't find \%1"
    if not directory \m(trash) {
        mkdir \m(trash)
        if fail end 1 "No trash can"
    }
    rename /list \%1 \m(trash)
}
define myundelete {
    local trash
    assign trash \v(home)trashcan/
    if not defined \%1 end 1 "Undelete what?"
    if wild \%1 end 1 "Undeleting multiple files is too hard"
    if not directory \m(trash) end 1 "No trash can"
    if not exist \m(trash)\%1 end 1 "I can't find \%1 in trash can"
    rename /list \m(trash)\%1 .
}
```

These macros are not exactly production quality (they don't handle filenames that include path segments, they don't handle multiple files, etc), but you get the idea: you can pass arguments to macros, they can check them and make other kinds of decisions, and the commands themselves are relatively intuitive and intelligible.

If you put the above lines into your initialization or customization file, you'll have MYDELETE and MYUNDELETE commands available every time you start Kermit, at least as long as you don't suppress execution of the initialization file. (*Exercise for the reader:* Make these macros generally useful: remove limitations, add trashcan display, browsing, emptying, etc.)

Kerbang scripts execute without the initialization file. This to keep them portable and also to make them start faster. If you want to write Kerbang scripts that depend on the initialization file, include the command

```
take \v(home).kermrc
```

at the desired spot in the script. By the way, \v(*xxx*) is a built−in variable (*xxx* is the variable name, "home" in this case). To see what built−in variables are available, type "show variables" at the C−Kermit prompt. To see what else you can show, type "show ?". \m(*xxx*) is a user defined variable (strictly speaking, it is a macro used as a variable).

Command List

C−Kermit has more than 200 top−level commands, and some of these, such as SET, branch off into hundreds of subcommands of their own, so it's not practical to describe them all here. Instead, here's a concise list of the most

commonly used top−level commands, grouped by category. To learn about each command, type "help" followed by the command name, e.g. "help set". Terms such as Command state and Connect state are explained in subsequent sections.

Optional fields are shown in *[* italicized brackets *]*. *filename* means the name of a single file. *filespec* means a file specification that is allowed to contain wildcard characters like '*' to match groups of files. *options* are (optional) switches like /PAGE, /NOPAGE, /QUIET, etc, listed in the HELP text for each command. Example:

```
send /recursive /larger:10000 /after:−1week /except:*.txt *
```

which can be read as "send all the files in this directory and all the ones underneath it that are larger than 10000 bytes, no more than one week old, and whose names don't end with ".txt".

Basic Commands

HELP	Requests top−level help.
HELP *command*	Requests help about the given command.
INTRODUCTION	Requests a brief introduction to C−Kermit.
LICENSE	Displays the C−Kermit software copyright and license.
VERSION	Displays C−Kermit's version number.
EXIT *[number]*	Exits from Kermit with the given status code. *Synonyms:* QUIT, E, Q.
TAKE *filename [parameters...]*	Executes commands from the given file.
LOG *item [filename]*	Keeps a log of the given item in the given file.
[DO *] macro [parameters...]*	Executes commands from the given macro.
SET *parameter value*	Sets the given parameter to the given value.
SHOW *category*	Shows settings in a given category.
STATUS	Tells whether previous command succeeded or failed.
DATE *[date−and/or−time]*	Shows current date−time or interprets given date−time.
RUN *[extern−command [parameters...]*	Runs the given external command. *Synonym:* !.
EXEC *[extern−command [params...]*	Kermit overlays itself with the given command.
SUSPEND	Stops Kermit and puts it in the background. *Synonym:* Z.

Local File Management

TYPE *[options] filename*	Displays the contents of the given file.
MORE *[options] filename*	Equivalent to TYPE /PAGE (pause after each screenful).
CAT *[options] filename*	Equivalent to TYPE /NOPAGE.
HEAD *[options] filename*	Displays the first few lines of a given file.
TAIL *[options] filename*	Displays the last few lines of a given file.
GREP *[options] pattern filespec*	Displays lines from files that match the pattern. *Synonym:* FIND.
DIRECTORY *[options] [filespec]*	Lists files (built−in, many options).

`LS [ options ] [ filespec ]`	Lists files (runs external "ls" command).
`DELETE [ options ] [ filespec ]`	Deletes files. *Synonym:* RM.
`PURGE [ options ] [ filespec ]`	Removes backup (`*.~n~`) files.
`COPY [ options ] [ filespecs... ]`	Copies files. *Synonym:* CP.
`RENAME [ options ] [ filespecs... ]`	Renames files. *Synonym:* MV.
`CHMOD [ options ] [ filespecs... ]`	Changes permissions of files.
`TRANSLATE filename charsets filename ]`	Converts file's character set. *Synonym:* XLATE.
`CD`	Changes your working directory to your home directory.
`CD directory`	Changes your working directory to the one given.
`CDUP`	Changes your working directory one level up.
`PWD`	Displays your working directory.
`BACK`	Returns to your previous working directory.
`MKDIR [ directory ]`	Creates a directory.
`RMDIR [ directory ]`	Removes a directory.

Making Connections

`SET LINE [ options ] devicename`	Opens the named serial port. *Synonym:* SET PORT.
`OPEN LINE [ options ] devicename`	Same as SET LINE. *Synonym:* OPEN PORT.
`SET MODEM TYPE [ name ]`	Tells Kermit what kind of modem is on the port.
`DIAL [ number ]`	Tells Kermit to dial the given phone number with the modem.
`REDIAL`	Redials the most recently dialed phone number.
`ANSWER`	Waits for and answers an incoming call on the modem.
`AUTHENTICATE [ parameters... ]`	Performs secure authentication on a TCP/IP connection.
`SET HOST [ options ] host [ port ]`	Opens a network connection to the given host and port.
`SET HOST * port`	Waits for an incoming TCP/IP connection on the given port.
`TELNET [ options ] host`	Opens a Telnet connection to the host and enters Connect state.
`RLOGIN [ options ] host`	Opens an Rlogin connection to the host and enters Connect state.
`IKSD [ options ] host`	Opens a connection to an Internet Kermit Service. state.
`SSH [ options ] host`	Opens an SSH connection to the host and enters Connect state.
`FTP OPEN host [ options ]`	Opens an FTP connection to the host.
`HTTP [ options ] OPEN host`	Opens an HTTP connection to the host.
`PTY external−command`	Runs the command on a pseudoterminal as if it were a connection.

PIPE *external−command*	Runs the command through a pipe as if it were a connection.

Using Connections

CONNECT *[options]*	Enters Connect (terminal) state. *Synonym:* C.
REDIRECT *command*	Redirects the given external command over the connection.
TELOPT *command*	Sends a Telnet protocol command (Telnet connections only).
Ctrl−\C	"Escapes back" from Connect state to Command state.
Ctrl−\B	*(In Connect state)* Sends a BREAK signal (serial or Telnet).
Ctrl−\!	*(In Connect state)* Enters inferior shell; "exit" to return.
Ctrl−\?	*(In Connect state)* Shows a menu of other escape−level options.
Ctrl−\Ctrl−\	*(In Connect state)* Type two Ctrl−Backslashes to send one of them.
SET ESCAPE *[character]*	Changes Kermit's Connect−state escape character.

Closing Connections

HANGUP	Hangs up the currently open serial−port or network connection.
CLOSE	Closes the currently open serial−port or network connection.
SET LINE *(with no devicename)*	Closes the currently open serial−port or network connection.
SET HOST *(with no hostname)*	Closes the currently open serial−port or network connection.
FTP CLOSE	Closes the currently open FTP connection.
HTTP CLOSE	Closes the currently open HTTP connection.
EXIT	Also closes all connections. *Synonym:* QUIT.
SET EXIT WARNING OFF	Suppresses warning about open connections on exit or close.

File Transfer

SEND *[options] filename [as−name]*	Sends the given file. *Synonym:* S.
SEND *[options] filespec*	Sends all files that match.
RESEND *[options] filespec*	Resumes an interupted SEND from the point of failure.
RECEIVE *[options] [as−name]*	Waits passively for files to arrive. *Synonym:* R.
LOG TRANSACTIONS *[filename]*	Keeps a record of file transfers.
FAST	Use fast file−transfer settings (default).
CAUTIOUS	Use cautious and less fast file−transfer settings.
ROBUST	

	Use ultra−conservative and slow file−transfer settings.
`STATISTICS [ options ]`	Gives statistics about the most recent file transfer.
`WHERE`	After transfer: "Where did my files go?".
`TRANSMIT [ options ] [ filename ]`	Sends file without protocol. *Synonym:* XMIT.
`LOG SESSION [ filename ]`	Captures remote text or files without protocol.
`SET PROTOCOL [ name... ]`	Tells Kermit to use an external file−transfer protocol.
`FTP { PUT, MPUT, GET, MGET, ... }`	FTP client commands.
`HTTP { PUT, GET, HEAD, POST, ... }`	HTTP client commands.

Kermit Server

`ENABLE, DISABLE`	Controls which features can be used by clients.
`SET SERVER`	Sets parameters prior to entering Server state.
`SERVER`	Enters Server state.

Client of Kermit or FTP Server

`[ REMOTE ] LOGIN [ user password ]`	Logs in to a Kermit server or IKSD that requires it.
`[ REMOTE ] LOGOUT`	Logs out from a Kermit server or IKSD.
`SEND [ options ] filename [ as−name ]`	Sends the given file to the server. *Synonyms:* S, PUT.
`SEND [ options ] filespec`	Sends all files that match.
`RESEND [ options ] filespec`	Resumes an interupted SEND from the point of failure.
`GET [ options ] remote−filespec`	Asks the server to send the given files. *Synonym:* G.
`REGET [ options ] remote−filespec`	Resumes an interrupted GET from the point of failure.
`REMOTE CD [ directory ]`	Asks server to change its working directory. *Synonym:* RCD.
`REMOTE PWD [ directory ]`	Asks server to display its working directory. *Synonym:* RPWD.
`REMOTE DIRECTORY [ filespec... ]`	Asks server to send a directory listing. *Synonym:* RDIR.
`REMOTE DELETE [ filespec... ]`	Asks server to delete files. *Synonym:* RDEL.
`REMOTE [ command... ]`	(Many other commands: "`remote ?`" for a list).
`MAIL [ options ] filespec`	Sends file(s) to be delivered as e−mail (Kermit only).
`FINISH`	Asks the server to exit server state (Kermit only).
`BYE`	Asks the server to log out and close the connection.

Script Programming

DEFINE, DECLARE, UNDEFINE, UNDECLARE, ASSIGN, EVALUATE, SEXPRESSION, ARRAY, SORT, INPUT, OUTPUT, IF, FOR, WHILE, SWITCH, GOTO, ECHO, ASK, GETC, GETOK, ASSERT, WAIT, SLEEP, FOPEN, FREAD, FWRITE, FCLOSE, STOP, END, RETURN, LEARN, SHIFT, TRACE, VOID, INCREMENT, DECREMENT, ... For these and many more you'll need to consult the manual and supplements, and/or visit the Kermit Script Library, which also includes a brief tutorial. *Hint:* HELP LEARN to find out how to get Kermit to write simple scripts for you.

Many of Kermit's commands have synonyms, variants, relatives, and so on. For example, MSEND is a version of SEND that accepts a list of file specifications to be sent, rather than just one file specification, and MPUT is a

synonym of MSEND. MOVE means to SEND and then DELETE the source file if successful. MMOVE is like MOVE, but accepts a list of filespecs, and so on. These are described in the full documentation.

Use question mark to feel your way through an unfamiliar command, as in this example (the part you type is underlined):

```
C−Kermit> remote ? One of the following:
 assign     delete     help       login      print      rename     space
 cd         directory  host       logout     pwd        rmdir      type
 copy       exit       kermit     mkdir      query      set        who
C−Kermit> remote set ? One of the following:
 attributes   file         retry        transfer
 block−check  receive      server       window
C−Kermit> remote set file ? One of the following:
 character−set  incomplete     record−length
 collision      names          type
C−Kermit> remote set file names ? One of the following:
 converted  literal
C−Kermit> remote set file names literal
C−Kermit>
```

This is called menu on demand**:** you get a menu when you want one, but menus are not forced on you even when know what you're doing. Note that you can also abbreviate most keywords, and you can complete them with the Tab or Esc key. Also note that `?` works for filenames too, and that you can use it in the middle of a keyword or filename, not just at the beginning. For example, "send x?" lists all the files in the current directory whose names start with 'x'.

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ]

INITIALIZATION FILE [Top] [Contents] [Next] [Previous]

In its default configuration, C−Kermit executes commands from a file called `.kermrc` in your home directory when it starts, unless it is given the `−Y` or `−y` command−line option. Custom configurations might substitute a shared system−wide initialization file. The SHOW FILE command tells what initialization file, if any, was used. The standard initialization file "chains" to an individual customization file, `.mykermc`, in the home directory, in which each user can establish her/his own preferences, define macros, and so on.

Since execution of the initialization file (at least the standard one) makes C−Kermit take longer to start, it might be better not to have an initialization file, especially now that Kermit's default startup configuration is well attuned to modern computing and networking −− in other words, you no longer have do anything special to make Kermit transfers go fast. So instead of having an initialization file that is executed every time Kermit starts, you might consider making one or more kerbang scripts (with names other that `.kermrc`) that do NOT include an "exit" command, and invoke those when you need the settings, macro definitions, and/or scripted actions they contain, and invoke C−Kermit directly when you don't.

To put it another way... We still distribute the standard initialization file since it's featured in the manual and backwards compatibility is important to us. But there's no harm in not using it if you don't need the stuff that's in it (services directory, dialing directory, network directory, and associated macro definitions). On the other hand, if there are settings or macros you want in effect EVERY time you use Kermit, the initialization file (or the customization file it chains to) is the place to put them, because that's the only place Kermit looks for them automatically each time you start it.

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ]

MODES OF OPERATION [Top] [Contents] [Next] [Previous]

Kermit is said to be in *Local mode* if it has made a connection to another computer, e.g. by dialing it or establishing a Telnet connection to it. The other computer is remote, so if you start another copy of Kermit on the remote computer, it is said to be in *Remote mode* (as long as it has not made any connections of its own). The local Kermit communicates over the communications device or network connection, acting as a conduit between the the remote computer and your keyboard and screen. The remote Kermit is the file−transfer partner to the local Kermit and communicates only through its standard input and output.

At any moment, a Kermit program can be in any of the following states. It's important to know what they are and how to change from one to the other.

Command state

In this state, Kermit reads commands from:

- Your keyboard; or:
- A file, or:
- A macro definition.

You can exit from Command state back to Unix with the EXIT or QUIT command (same thing). You can enter Connect state with any of various commands (CONNECT, DIAL, TELNET, etc). You can enter file transfer state with commands like SEND, RECEIVE, and GET. You can enter Server state with the SERVER command. The TAKE command tells Kermit to read and execute commands from a file. The (perhaps implied) DO command tells Kermit to read and execute commands from a macro definition. While in Command state, you can interrupt any command, macro, or command file by typing Ctrl−C (hold down the Ctrl key and press the C key); this normally brings you back to the prompt.

Shell state

You can invoke an inferior shell or external command from the Kermit command prompt by using the PUSH, RUN (!), EDIT, or BROWSE command. While the inferior shell or command is active, Kermit is suspended and does nothing. Return to Kermit Command state by exiting from the inferior shell or application.

Connect state

In this state, which can be entered only when in Local mode (i.e. when Kermit has made a connection to another computer), Kermit is acting as a terminal to the remote computer. Your keystrokes are sent to the remote computer and characters that arrive over the communication connection are displayed on your screen. This state is entered when you give a CONNECT, DIAL, TELNET, RLOGIN, or IKSD command. You can return to command state by logging out of the remote computer, or by typing:

```
Ctrl−\c
```

That is: Hold down the Ctrl key and press the backslash key, then let go of the Ctrl key and press the C key. This is called *escaping back*. Certain other escape−level commands are also provided; type Ctrl−\? for a list. For example, you can enter Shell state with:

```
Ctrl−\!
```

To send a Ctrl−\ to the host while in Connect state, type two of them in a row. See HELP CONNECT and HELP SET ESCAPE for more info.

Local file−transfer state

In this state, Kermit is sending packets back and forth with the other computer in order to transfer a file or accomplish some other file−related task. And at the same time, it is displaying its progress on your screen and watching your keyboard for interruptions. In this state, the following single−keystroke commands are accepted:

`X`	Interrupt the current file and go on to the next (if any).
`Z`	Interrupt the current file and skip all the rest.
`E`	Like Z but uses a "stronger" protocol (use if X or Z don't work).
`Ctrl-C`	Interrupt file−transfer mode (use if Z or E don't work).

Kermit returns to its previous state (Command or Connect) when the transfer is complete or when interrupted successfully by X, Z, E, or Ctrl−C (hold down the Ctrl key and press the C key).

Remote file−transfer state

In this state, Kermit is exchanging file−transfer packets with its local partner over its standard i/o. It leaves this state automatically when the transfer is complete. In case you find your local Kermit in Connect state and the remote one in File−transfer state (in which it seems to ignore your keystrokes), you can usually return it to command state by typing three Ctrl−C's in a row. If that doesn't work, return your local Kermit to Command state (Ctrl−\ C) and type "e−packet" and then press the Return or Enter key; this forces a fatal Kermit protocol error.

Remote Server state

This is like Remote File−transfer state, except it never returns automatically to Command state. Rather, it awaits further instructions from the client program; that is, from your Local Kermit program. You can return the Remote Server to its previous state by issuing a "finish" command to the client, or if you are in Connect state, by typing three Ctrl−C's in a row. You can tell the server job to log out and break the connection by issuing a "bye" command to the client.

Local Server state

Like Remote−Server state, but in local mode, and therefore with its file−transfer display showing, and listening for single−key commands, as in Local File−transfer state. Usually this state is entered automatically when a remote Kermit program gives a GET command.

C−Kermit, Kermit 95, and MS−DOS Kermit all can switch automatically from Connect state to Local File−transfer state when you initiate a file transfer from the remote computer by starting Kermit and telling it to send or get a file, in which case, Connect state is automatically resumed after the file transfer is finished.

Note that C−Kermit is not a terminal emulator. It is a communications application that you run in a terminal window (e.g. console or Xterm). The specific emulation, such as VT100, VT220, Linux Console, or Xterm, is provided by the terminal window in which you are running C−Kermit. Kermit 95 and MS−DOS Kermit, on the other hand, are true terminal emulators. *Why is C−Kermit not a terminal emulator?* CLICK HERE to read about it.

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ]

MAKING CONNECTIONS [Top] [Contents] [Next] [Previous]

Here is how to make different kinds of connections using interactive Kermit commands (as noted above, you can also make connections with command−line options). Note that you don't *have* to make connections with Kermit. It can also be used on the far end of a connection as the remote file transfer and management partner of your local communications software.

Making a Telnet Connection

At the C−Kermit command prompt, simply type:

```
telnet foo.bar.com         ; Substitute desired host name or address.
telnet xyzcorp.com 3000    ; You can also include a port number.
```

If the connection is successful, Kermit automically enters Connect state. When you logout from the remote host, Kermit automatically returns to its prompt. More info: HELP TELNET, HELP SET TELNET, HELP SET TELOPT. Also see the IKSD section below.

Making an Rlogin connection

This is just like Telnet, except you have to be root to do it because Rlogin uses a privileged TCP port:

```
rlogin foo.bar.com         ; Substitute desired host name or address.
```

More info: HELP RLOGIN.

Making an SSH Connection

Unlike Telnet and Rlogin, SSH connections are not built−in, but handled by running your external SSH client through a pseudoterminal. Using C−Kermit to control the SSH client gives you all of Kermit's features (file transfer, character−set conversion, scripting, etc) over SSH.

```
ssh foo.bar.com            ; Substitute desired host name or address.
```

More info: HELP SSH, HELP SET SSH.

Dialing with a Modem

If it's an external modem, make sure it is connected to a usable serial port on your computer with a regular (straight−through) modem cable, and to the telephone jack with a telephone cable, and that it's turned on. Then use these commands:

```
set modem type usrobotics  ; Or other supported type
set line /dev/ttyS0        ; Specify device name
set speed 57600            ; Or other desired speed
set flow rts/cts           ; Most modern modems support this
set dial method tone       ; (or pulse)
dial 7654321               ; Dial the desired number
```

Type "set modem type ?" for a list of supported modem types. If you omit the SET MODEM TYPE command, the default type is "generic−high−speed", which should work for most modern AT−command−set

modems. If the line is busy, Kermit redials automatically. If the call does not succeed, use "set dial display on" and try it again to watch what happens. If the call succeeds, Kermit enters Connect state automatically and returns to its prompt automatically when you log out from the remote computer or the connection is otherwise lost.

You can also dial from a modem that is accessible by Telnet, e.g. to a reverse terminal server. In this case the command sequence is:

```
set host ts.xxx.com 2000   ; Terminal−server and port
set modem type usrobotics  ; Or other supported type
set dial method tone       ; (or pulse)
dial 7654321               ; Dial the desired number
```

If the terminal server supports the Telnet Com Port Option, RFC 2217, you can also give serial−port related commands such as SET SPEED, SET PARITY, and so on, and Kermit relays them to the terminal server using the protocol specified in the RFC.

More info: HELP SET MODEM, HELP SET LINE, HELP SET SPEED, HELP SET FLOW, HELP DIAL, HELP SET DIAL, HELP SET MODEM, HELP SET CARRIER−WATCH, SHOW COMMUNICATIONS, SHOW MODEM, SHOW DIAL.

Direct Serial Port

Connect the two computers, A and B, with a null modem cable (or two modem cables interconnected with a null−modem adapter or modem eliminator). From Computer A:

```
set modem type none        ; There is no modem
set line /dev/ttyS0        ; Specify device name
set carrier−watch off      ; If DTR and CD are not cross−connected
set speed 57600            ; Or other desired speed
set flow rts/cts           ; If RTS and CTS are cross−connected
set flow xon/xoff          ; If you can't use RTS/CTS
connect                    ; Enter Connect (terminal) state
```

This assumes Computer B is set up to let you log in. If it isn't, you can run a copy of Kermit on Computer B and follow approximately the same directions. More info: As above plus HELP CONNECT.

With modems or direct serial connections, you might also have to "set parity even" (or "mark" or "space") if it's a 7−bit connection.

Of the connection types listed above, only one can be open at a time. However, any one of these can be open concurrently with an FTP or HTTP session. Each connection type can be customized to any desired degree, scripted, logged, you name it. See the manual.

NOTE: On selected platforms, C−Kermit also can make X.25 connections. See the manual for details.

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ]

TRANSFERRING FILES WITH KERMIT [Top] [Contents] [Next] [Previous]

- Downloading Files
- Uploading Files

There is a widespread and persistent belief that Kermit is a slow protocol. This is because, until recently, it used conservative tuning by default to make sure file transfers succeeded, rather than failing because they overloaded the connection. Some extra commands (or command−line options, like `−Q`) were needed to make it go fast, but nobody bothered to find out about them. Also, it takes two to tango: most non−Kermit−Project Kermit protocol implementations really ARE slow. The best file−transfer partners for C−Kermit are: another copy of C−Kermit (7.0 or later) and Kermit 95. These combinations work well and they work fast by default. MS−DOS Kermit is good too, but you have to tell it to go fast (by giving it the FAST command).

Furthermore, all three of these Kermit programs support **"autodownload" and "autoupload"**, meaning that when they are in Connect state and a Kermit packet comes in from the remote, they automatically switch into file transfer mode.

And plus, C−Kermit and K95 also **switch automatically between text and binary mode** for each file, so there is no need to "set file type binary" or "set file type text", or to worry about files being corrupted because they were transferred in the wrong mode.

What all of these words add up to is that now, when you use up−to−date Kermit software from the Kermit Project, file transfer is not only fast, it's ridiculously easy. You barely have to give any commands at all.

Downloading Files

Let's say you have Kermit 95, C−Kermit, or MS−DOS Kermit on your desktop computer, with a connection to a Unix computer that has C−Kermit installed as "kermit". To download a file (send it from Unix to your desktop computer), just type the following command at your Unix shell prompt:

```
kermit −s oofa.txt
```

(where `oofa.txt` is the filename). If you want to send more than one file, you can put as many filenames as you want on the command line, and they can be any combination of text and binary:

```
kermit −s oofa.txt oofa.zip oofa.html oofa.tar.gz
```

and/or you can use wildcards to send groups of files:

```
kermit −s oofa.*
```

If you want to send a file under an assumed name, use:

```
kermit −s friday.txt −a today.txt
```

This sends the file `friday.txt` but tells the receiving Kermit that its name is `today.txt`. In all cases, as noted, when the file transfer is finished, your desktop Kermit returns automatically to Connect state. No worries about escaping back, re−connecting, text/binary mode switching. Almost too easy, right?

Uploading Files

To upload files (send them from your desktop computer to the remote Unix computer) do the same thing, but use the `−g` (GET) option instead of `−s`:

```
kermit −g oofa.txt
```

This causes your local Kermit to enter server mode; then the remote Kermit program requests the named file and the local Kermit sends it and returns automatically to Connect state when done.

If you want to upload multiple files, you have have use shell quoting rules, since these aren't local files:

```
kermit −g "oofa.txt oofa.zip oofa.html oofa.tar.gz"
kermit −g "oofa.*"
```

If you want to upload a file but store it under a different name, use:

```
kermit −g friday.txt −a today.txt
```

Kermit Transfers the Old−Fashioned Way

If your desktop communications software does not support autoupload or autodownload, or it does not include Kermit server mode, the procedure requires more steps.
To download a file, type:

```
kermit −s filename
```

on the host as before, but if nothing happens automatically in response to this command, you have to switch your desktop communications software into Kermit Receive state. This might be done by escaping back using keyboard characters or hot keys (Alt−x is typical) and/or with a command (like RECEIVE) or a menu. When the file transfer is complete, you have to go back to Connect state, Terminal emulation, or whatever terminology applies to your desktop communications software.

To upload a file, type:

```
kermit −r
```

on the host (rather than "`kermit −g`"). This tells C−Kermit to wait passively for a file to start arriving. Then regain the attention of your desktop software (Alt−x or whatever) and instruct it to send the desired file(s) with Kermit protocol. When the transfer is finished, return to the Connect or Terminal screen.

If File Transfer Fails

Although every aspect of Kermit's operation can be finely tuned, there are also three short and simple "omnibus tuning" commands you can use for troubleshooting:

FAST

Use fast file−transfer settings. This has been the default since C−Kermit 7.0 now that most modern computers and connections support it. If transfers fail with fast settings, try . . .

CAUTIOUS

Use cautious but not paranoid settings. File transfers, if they work, will go at medium speed. If not, try . . .

ROBUST

Use the most robust, resilient, conservative, safe, and reliable settings. File transfers will almost certainly work, but they will be quite slow (of course this is a classic tradeoff; ROBUST was C−Kermit's default tuning in versions 6.0 and earlier, which made everybody think Kermit protocol was slow). If ROBUST doesn't do the trick, try again with SET PARITY SPACE first in case it's not an 8−bit connection.

Obviously the success and performance of a file transfer also depends on C−Kermit's file transfer partner. Up−to−date, real Kermit Project partners are recommended because they contain the best Kermit protocol implementations and because we can support them in case of trouble.

If you still have trouble, consult Chapter 10 of *Using C−Kermit*, or send email to kermit−support@columbia.edu.

Advanced Kermit File−Transfer Features

Obviously there is a lot more to Kermit file transfer, including all sorts of interactive commands, preferences, options, logging, debugging, troubleshooting, and anything else you can imagine but that's what the manual and updates are for. Here are a few topics you can explore if you're interested by Typing HELP for the listed commands:

Logging transfers:
LOG TRANSACTIONS (HELP LOG)

Automatic per−file text/binary mode switching:
SET TRANSFER MODE *{* AUTOMATIC, MANUAL *}* (HELP SET TRANSFER).

Cross−platform recursive directory tree transfer:
SEND /RECURSIVE, GET /RECURSIVE (HELP SEND, HELP GET).

File collision options:
SET FILE COLLISION *{* OVERWRITE, BACKUP, DISCARD, ... *}* (HELP SET FILE).

Update mode (only transfer files that changed since last time):
SET FILE COLLISION UPDATE (HELP SET FILE).

Filename selection patterns:
(HELP WILDCARD).

Flexible file selection:
SEND (or GET) /BEFORE /AFTER /LARGER /SMALLER /TYPE /EXCEPT, ...

Character−set conversion:
SET *{* FILE, TRANSFER *}* CHARACTER−SET, ASSOCIATE, ...

File/Pathname control:
SET *{* SEND, RECEIVE *}* PATHNAMES, SET FILE NAMES.

Atomic file movement:
SEND (or GET) /DELETE /RENAME /MOVE−TO

Transferring to/from standard i/o of other commands:

SEND (or GET) /COMMAND

Recovery of interrupted transfer from point of failure:
RESEND, REGET (HELP RESEND, HELP REGET).

Non−Kermit File Transfer

You can also use C−Kermit to transfer files with FTP or HTTP Internet protocols; see below.
On a regular serial or Telnet connection where the other computer doesn't support Kermit protocol at all, you have several options. For example, if your desktop communications software supports Zmodem, use "rz" and "sz" on the host rather than Kermit. But if Kermit *is* your desktop software, and you are using it to make calls or network connections to other computers that don't support Kermit protocol (or that don't have a good implementation of it), then if your computer also has external X, Y, or Zmodem programs that are redirectable, Kermit can use them as external protocols. HELP SET PROTOCOL for details.

You can also capture "raw" data streams from the other computer with LOG SESSION (HELP LOG and HELP SET SESSION−LOG for details), and you can upload files without any protocol at all with TRANSMIT (HELP TRANSMIT, HELP SET TRANSMIT).

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ]

Kermit's FTP client is like the regular Unix FTP client that you're used to, but with some differences:

- It has lots more commands and features.
- Each FTP command must be prefixed with "ftp", for example "ftp open", "ftp get", "ftp bye", etc (this is not strictly true, but until you're more familiar with it, it's best to follow this rule).
- Commands like "cd", "directory", etc, execute locally, not on the server. Use "ftp cd", "ftp dir", etc, to have them act on the server.
- You can have an FTP session and a regular Kermit serial or Telnet session open at the same time.
- FTP sessions can be fully automated.

Pending publication of the next edition of the manual, the Kermit FTP client is thoroughly documented at the Kermit Project website:

`http://www.columbia.edu/kermit/ftpclient.html`

You also can use HELP FTP and HELP SET FTP to get descriptions of Kermit's FTP−related commands.

The HTTP client is similar to the FTP one, except you prefix each command with HTTP instead of FTP: HTTP OPEN, HTTP GET, HTTP PUT, HTTP CLOSE, etc. Type HELP HTTP for details, or visit the to view the manual supplements. HTTP connections can be open at the same time as regular serial or Telnet connections and FTP connections. So Kermit can manage up to three types connections simultaneously.

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ] [FTP Client] [HTTP Client]

INTERNET KERMIT SERVICE [Top] [Contents] [Next] [Previous]

C−Kermit can be configured and run as an Internet service (called IKSD), similar to an FTP server (FTPD) except you can (but need not) interact with it directly, plus it does a lot more than an FTP server can do. The TCP port for IKSD is 1649. It uses Telnet protocol. C−Kermit can be an Internet Kermit Server, or it can be a client of an IKSD. You can make connections from C−Kermit to an IKSD with any of the following commands:

```
telnet foo.bar.edu 1649
telnet foo.bar.edu kermit   ; if "kermit" is listed in /etc/services
iksd foo.bar.edu
```

The IKSD command is equivalent to a TELNET command specifying port 1649. For more information about making and using connections to an IKSD, see:

http://www.columbia.edu/kermit/cuiksd.html

You can run an Internet Kermit Service on your own computer too (if you are the system administrator). For instructions, see:

http://www.columbia.edu/kermit/iksd.html

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ]

SECURITY [Top] [Contents] [Next] [Previous]

All of C−Kermit's built−in TCP/IP networking methods (Telnet, Rlogin, IKSD, FTP, and HTTP) can be secured by one or more of the following IETF−approved methods:

- MIT Kerberos IV
- MIT Kerberos V
- SSL/TLS
- Stanford SRP

For complete instructions see:

http://www.columbia.edu/kermit/security.html

And as noted previously, you can also make SSH connections with C−Kermit if you already have an SSH client installed.

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ]

ALTERNATIVE COMMAND−LINE PERSONALITIES [Top] [Contents] [Next] [Previous]

When invoked as "kermit" or any other name besides any of the special ones, C−Kermit has the command−line options described above in the OPTIONS section. However, if you invoke C−Kermit using any of the following names:

`telnet`	Telnet client
`ftp`	FTP client
`http`	HTTP client
`https`	Secure HTTP client

Kermit's command−line personality changes to match. This can be done (among other ways) with symbolic links (symlinks). For example, if you want C−Kermit to be your regular Telnet client, or the Telnet helper of your Web browser, you can create a link like the following in a directory that lies in your PATH ahead of the regular telnet program:

```
ln −s /usr/local/bin/kermit telnet
```

Now when you give a "telnet" command, you are invoking Kermit instead, but with its Telnet command−line personality so, for example:

```
telnet xyzcorp.com
```

Makes a Telnet connection to xyzcorp.com, and Kermit exits automatically when the connection is closed (just like the regular Telnet client). Type "`telnet −h`" to get a list of Kermit's Telnet−personality command−line options, which are intended to be as compatible as possible with the regular Telnet client.

Similarly for FTP:

```
ln −s /usr/local/bin/kermit ftp
```

And now type "`ftp −h`" to see its command−line options, and use command lines just like you would give your regular FTP client:

```
ftp −n xyzcorp.com
```

but with additional options allowing an entire session to be specified on the command line, as explained in the C−Kermit FTP client documentation.

And similarly for HTTP:

```
ln −s /usr/local/bin/kermit http
./http −h
./http www.columbia.edu −g kermit/index.html
```

Finally, if Kermit's first command−line option is a Telnet, FTP, IKSD, or HTTP URL, Kermit automatically makes the appropriate kind of connection and, if indicated by the URL, takes the desired action:

```
kermit telnet:xyzcorp.com                          ; Opens a Telnet session
kermit telnet://olga@xyzcorp.com                   ; Ditto for user olga
kermit ftp://olga@xyzcorp.com/public/oofa.zip      ; Downloads a file
kermit kermit://kermit.columbia.edu/kermit/f/READ.ME ; Ditto for IKSD
kermit iksd://kermit.columbia.edu/kermit/f/READ.ME   ; (This works too)
kermit http://www.columbia.edu/kermit/index.html     ; Grabs a web page
kermit https://wwws.xyzcorp.com/secret/plan.html     ; Grabs a secure web page
```

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ]

LICENSE [Top] [Contents] [Next] [Previous]

C−Kermit has an unusual license, but a fair and sensible one given that the Kermit Project must support itself out of revenue**:** it's not a BSD license, not GPL, not Artistic, not commercial, not shareware, not freeware. It can be summed up like this**:** if you want C−Kermit for your own use, you can download and use it without cost or license (but we'd appreciate it if you would purchase the manual). But if you want to *sell* C−Kermit or bundle it with a product or otherwise distribute it in a commercial setting EXCEPT WITH AN OPEN−SOURCE OPERATING SYSTEM DISTRIBUTION such as Linux, FreeBSD, NetBSD, or OpenBSD, *you must license it*. To see the complete license, give the LICENSE command at the prompt, or see the `COPYING.TXT` file distributed with C−Kermit 7.0 or later, or download it from `ftp://kermit.columbia.edu/kermit/c-kermit/COPYING.TXT`. Send licensing inquiries to `kermit@columbia.edu`.

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ]

OTHER TOPICS [Top] [Contents] [Next] [Previous]

There's way more to C−Kermit than we've touched on here −− troubleshooting, customization, character sets, dialing directories, sending pages, script writing, and on and on, all of which are covered in the manual and updates and supplements. For the most up−to−date information on documentation (or updated documentation itself) visit the Kermit Project website:

`http://www.columbia.edu/kermit/`

There you will also find Kermit software packages for other platforms: different Unix varieties, Windows, DOS, VMS, IBM mainframes, and many others**:** 20+ years' worth.

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ]

DOCUMENTATION AND UPDATES [Top] [Contents] [Next] [Previous]

The manual for C−Kermit is:

1. Frank da Cruz and Christine M. Gianone, *Using C−Kermit*, Second Edition, Digital Press / Butterworth−Heinemann, Woburn, MA, 1997, 622 pages, ISBN 1−55558−164−1. This is a printed book. It covers C−Kermit 6.0.

2. The C−Kermit 7.0 Supplement: `http://www.columbia.edu/kermit/ckermit2.html`

3. The C−Kermit 8.0 Supplement: `http://www.columbia.edu/kermit/ckermit3.html`

The C−Kermit home page is here:

`http://www.columbia.edu/kermit/ckermit.html`

Visit this page to learn about new versions, Beta tests, and other news; to read case studies and tutorials; to download source code, install packages, and prebuilt binaries for many platforms. Also visit:

`http://www.columbia.edu/kermit/scriptlib.html`

The Kermit script library and tutorial

http://www.columbia.edu/kermit/newfaq.html
The Kermit FAQ (Frequently Asked Questions about Kermit)

http://www.columbia.edu/kermit/ckfaq.html
The C−Kermit FAQ (Frequently Asked Questions about C−Kermit)

http://www.columbia.edu/kermit/telnet.html
C−Kermit Telnet client documentation.

http://www.columbia.edu/kermit/studies.html
Case studies.

http://www.columbia.edu/kermit/support.html
Technical support.

http://www.columbia.edu/kermit/k95tutorial.html
Kermit 95 tutorial.

comp.protocols.kermit.misc
The Kermit newsgroup (unmoderated).

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ]

FILES [Top] [Contents] [Next] [Previous]

COPYING.TXT
C−Kermit license.
~/.kermrc
Initialization file.
~/.mykermrc
Customization file.
~/.kdd
Kermit dialing directory (see manual).
~/.knd
Kermit network directory (see manual).
~/.ksd
Kermit services directory (see manual).
ckuins.txt
Installation instructions for Unix.
ckcbwr.txt
General C−Kermit bugs, hints, tips.
ckubwr.txt
Unix−specific C−Kermit bugs, hints, tips.
ckcplm.txt
C−Kermit program logic manual.
ckccfg.txt
C−Kermit compile−time configuration options.
ssh

(in your PATH) SSH connection helper.

rz, *sz*, *etc.*
(in your PATH) external protocols for XYZmodem.

/var/spool/locks (or whatever)
UUCP lockfile for dialing out (see installation instructions).

[Kermit Home] [C−Kermit Home] [C−Kermit FAQ]

AUTHORS [Top] [Contents] [Previous]

Frank da Cruz and Jeffrey E Altman
The Kermit Project − Columbia Univerity
612 West 115th Street
New York NY 10025−7799
USA

1985−present, with contributions from hundreds of others all over the world.
