C-KERMIT USER GUIDE

For UNIX, VMS, and Many Other Operating Systems

Version 4E(072)

C. Gianone, F. da Cruz

Columbia University Center for Computing Activities
New York, New York 10027

February 8, 1989

1. UNIX KERMIT

Program: Frank da Cruz, Bill Catchings, Jeff Damens, Columbia University; Herm Fischer, Encino CA; contributions by many others.
Language: C
Documentation: Christine Gianone, Frank da Cruz
Version: `4E(072)`
Date: February 8, 1989

C-Kermit is an implementation of Kermit, written modularly and transportably in C. The protocol state transition table is written in *wart*, a (non-proprietary) lex-like preprocessor for C. System-dependent primitive functions are isolated into separately compiled modules so that the program should be easily portable among Unix systems and also to non-Unix systems that have C compilers, such as VAX/VMS, Data General AOS/VS, Apollo Aegis, the Apple Macintosh, and the Commodore Amiga. This document applies to Unix implementations of C-Kermit, and in most ways also to the VMS, Data General, and other implementations.

Unix Kermit Capabilities At A Glance:

Local operation:	Yes
Remote operation:	Yes
Login scripts:	Yes (UUCP style)
Transfer text files:	Yes
Transfer binary files:	Yes
Wildcard send:	Yes
File transfer interruption:	Yes
Filename collision avoidance:	Yes
Can time out:	Yes
8th-bit prefixing:	Yes
Repeat count prefixing:	Yes
Alternate block checks:	Yes
Terminal emulation:	Yes
Communication settings:	Yes
Transmit BREAK:	Yes (most versions)
Support for dialout modems:	Yes
IBM mainframe communication:	Yes
Transaction logging:	Yes
Session logging:	Yes
Debug logging:	Yes
Packet logging:	Yes
Act as server:	Yes
Talk to server:	Yes
Advanced server functions:	Yes
Local file management:	Yes
Command/Init files:	Yes
UUCP and multiuser line locking:	Yes
Long packets:	Yes
Sliding Windows:	No
File attributes packets:	No
Command macros:	No
Raw file transmit:	No

All numbers in the C-Kermit documentation are decimal unless noted otherwise.

C-Kermit provides traditional Unix command line operation as well as interactive command prompting and execution. The command line options provide access to a basic subset of C-Kermit's capabilities; the interactive command set is far richer.

On systems with dialout modems, C-Kermit's command file, DIAL command, and login script facilities provide a counterpart to UUCP for file transfer with non-UNIX operating systems, including the use of scheduled (e.g. late night) unattended operation.

1.1. The Unix File System

Consult your Unix manual for details about the file system under your version of Unix. In general, Unix files have lowercase names, possibly containing one or more dots or other special characters. Unix directories are tree-structured. Directory levels are separated by slash (``/'') characters. For example,

```
/usr/foo/bar
```

denotes the file `bar` in the directory `/usr/foo`. Alphabetic case is significant in Unix file and directory names, i.e. ``a'' is a different file (or directory) from ``A''. Wildcard or "meta" characters allow groups of files to be specified. ``*'' matches any string; ``?'' matches any single character.

When C-Kermit is invoked with file arguments specified on the Unix command line, the Unix shell (Bourne Shell, C-Shell, K-Shell, etc) expands the meta characters itself, and in this case a wider variety is available. For example,

```
kermit -s ~/ck[uvm]*.{upd,bwr}]
```

is expanded by the Berkeley C-Shell into a list of all the files in the user's home directory (`~/`) that start with the characters "`ck`", followed by a single character ``u'', ``v'', or ``m'', followed by zero or more characters, followed by a dot, followed by one of the strings ``upd'' or ``bwr''. Internally, the C-Kermit program itself expands only the ``*'' and ``?'' meta characters.

Unix files are linear (sequential) streams of 8-bit bytes. Text files consist of 7-bit ASCII characters, with the high-order bit off (0), and lines separated by the Unix newline character, which is linefeed (LF, ASCII 10). This distinguishes Unix text files from those on most other ASCII systems, in which lines are separated by a carriage-return linefeed sequence (CRLF, ASCII 13, followed by linefeed, ASCII 10). Binary files are likely to contain data in the high bits of the file bytes, and have no particular line or record structure.

When transferring files, C-Kermit will convert between upper and lower case filenames and between LF and CRLF line terminators automatically, unless told to do otherwise. When binary files must be transferred, the program must be instructed not to perform LF/CRLF conversion (`-i` on the command line or "set file type binary" interactively; see below).

1.2. File Transfer

If C-Kermit is in local mode, the screen (stdout) is continously updated to show the progress of the file transer. A dot is printed for every four data packets, other packets are shown by type:

I Exchange Parameter Information
R Receive Initiate
S Send Initiate
F File Header
G Generic Server Command
C Remote Host Command
N Negative Acknowledgement (NAK)
E Fatal Error
T Indicates a timeout occurred
Q Indicates a damaged, undesired, or illegal packet was received
% Indicates a packet was retransmitted

You may type certain "interrupt" commands during file transfer:

Control-F: Interrupt the current File, and go on to the next (if any).

Control-B: Interrupt the entire Batch of files, terminate the transaction.
Control-R: Resend the current packet
Control-A: Display a status report for the current transaction.

These interrupt characters differ from the ones used in other Kermit implementations to avoid conflict with commonly used Unix shell interrupt characters. With Version 7, System III, and System V implementations of Unix, interrupt commands must be preceeded by the 'connect' escape character (e.g. normally-`\`). Ctrl-F and Ctrl-B are effective only during the transfer of data (D) packets, and cannot be used to interrupt a transfer that has not yet reached that stage.

> *CAUTION:* If Control-F or Control-B is used to cancel an incoming file, and a file of the same name previously existed, *and* the "file warning" feature is not enabled, then the previous copy of the file will disappear.

EMERGENCY EXIT: When running Unix Kermit in remote mode, if you have started a protocol operation (sending or receiving a file, server command wait, etc), you will not be able to communicate with the terminal in the normal way. In particular, you cannot stop the protocol by typing the normal Unix interrupt characters, since the terminal has been put in "raw mode". If you need to regain control quickly -- for instance, because the protocol is stuck -- you can type two Control-C's directly to the Unix Kermit program ("connect" first if necessary):

Control-C Control-C

This will cause the program to display,

```
^C^C...
```

exit, and restore the terminal to normal.

1.3. Command Line Operation

The C-Kermit command line syntax conforms to the Proposed Syntax Standards for Unix System Commands put forth by Kathy Hemenway and Helene Armitage of AT&T Bell Laboratories in *Unix/World*, Vol.1, No.3, 1984. The rules that apply are:

- Command names must be between 2 and 9 characters ("kermit" is 6).
- Command names must include lower case letters and digits only.
- An option name is a single character.
- Options are delimited by '`-`'.
- Options with no arguments may be grouped (bundled) behind one delimiter.
- Option-arguments cannot be optional.
- Arguments immediately follow options, separated by whitespace.
- The order of options does not matter.
- '`-`' preceded and followed by whitespace means standard input.

A group of bundled options may end with an option that has an argument.

The following notation is used in command descriptions:

fn A Unix file specification, possibly containing the "wildcard" characters '`*`' or '`?`' ('`*`' matches all character strings, '`?`' matches any single character).

fn1 A Unix file specification which may not contain '`*`' or '`?`'.

rfn A remote file specification in the remote system's own syntax, which may denote a single file or a group of files.

rfn1 A remote file specification which should denote only a single file.

n A decimal number between 0 and 94.

c A decimal number between 0 and 127 representing the value of an ASCII character.

cc A decimal number between 0 and 31, or else exactly 127, representing the value of an ASCII control character.

`[ ]` Any field in square braces is optional.

`{x,y,z}`
Alternatives are listed in curly braces.

C-Kermit command line options may specify any combination of actions and settings. If C-Kermit is invoked with a command line that specifies no actions, then it will issue a prompt and begin interactive dialog. Action options specify either protocol transactions or terminal connection.

`-s` *fn* Send the specified file or files. If *fn* contains wildcard (meta) characters, the Unix shell expands it into a list. If *fn* is '`-`' then kermit sends from standard input, which may come from a file:

```
kermit -s - < foo.bar
```

or a parallel process:

```
ls -l | grep christin | kermit -s -
```

You cannot use this mechanism to send terminal typein. If you want to send a file whose actual name is ''`-`'' you can precede it with a path name, as in

```
kermit -s ./-
```

`-r` Receive a file or files. Wait passively for files to arrive.

`-k` Receive (passively) a file or files, sending them to standard output. This option can be used in several ways:

`kermit -k`
Displays the incoming files on your screen; to be used only in "local mode" (see below).

`kermit -k >` *fn1*
Sends the incoming file or files to the named file, *fn1*. If more than one file arrives, all are concatenated together into the single file *fn1*.

`kermit -k | command`
Pipes the incoming data (single or multiple files) to the indicated command, as in

```
kermit -k | sort > sorted.stuff
```

`-a` *fn1* If you have specified a file transfer option, you may give an alternate name for a single file with the `-a` ("as") option. For example,

```
kermit -s foo -a bar
```

sends the file `foo` telling the receiver that its name is `bar`. If more than one file arrives or is sent, only the first file is affected by the `-a` option:

```
kermit -ra baz
```

stores the first incoming file under the name `baz`.

`-x` Begin server operation. May be used in either local or remote mode.

Before proceeding, a few words about remote and local operation are necessary. C-Kermit is "local" if it is running on PC or workstation that you are using directly, or if it is running on a multiuser system and transferring files over an external communication line -- not your job's controlling terminal or console. C-Kermit is remote if it is running on a multiuser system and transferring files over its own controlling terminal's communication line (normally `/dev/tty`), connected to your PC or workstation.

If you are running C-Kermit on a PC, it is normally used in local mode, with the "back port" designated for file transfer and terminal connection. If you are running C-Kermit on a multiuser (timesharing) system, it is in remote mode unless you explicitly point it at an external line for file transfer or terminal connection. The following command sets C-Kermit's "mode":

`-l` *dev* Line -- Specify a terminal line to use for file transfer and terminal connection, as in

```
kermit -l /dev/ttyi5
```

When an external line is being used, you will also need some additional options for successful communication with the remote system:

`-b` *n* Baud -- Specify the baud rate for the line given in the `-l` option, as in

```
kermit -l /dev/ttyi5 -b 9600
```

This option should always be included with the `-l` option, since the speed of an external line is not necessarily what you expect.

`-p` *x* Parity -- e,o,m,s,n (even, odd, mark, space, or none). If parity is other than none, then the 8th-bit prefixing mechanism will be used for transferring 8-bit binary data, provided the opposite Kermit agrees. The default parity is none.

`-t` Specifies half duplex, line turnaround with XON as the handshake character.

The following commands may be used only with a C-Kermit which is local either by default or else because the `-l` option has been specified.

`-g` *rfn* Actively request a remote server to send the named file or files; *rfn* is a file specification in the remote host's own syntax. If *fn* happens to contain any special shell characters, like space, '`*`', '`[`', etc, these must be quoted, as in

```
kermit -g x\*.\?
```

or

```
kermit -g "profile exec"
```

`-f` Send a 'finish' command to a remote server.

`-c` Establish a terminal connection over the specified or default communication line, before any protocol transaction takes place. Get back to the local system by typing the escape character (normally Control-Backslash) followed by the letter 'c'.

`-n` Like `-c`, but *after* a protocol transaction takes place; `-c` and `-n` may both be used in the same command. The use of `-n` and `-c` is illustrated below.

If the other Kermit is on a remote system, the `-l` and `-b` options should also be included with the `-r`, `-k`, or `-s` options.

Several other command-line options are provided:

`-i` Specifies that files should be sent or received exactly "as is" with no conversions. This option is necessary for transmitting binary files. It may also be used in Unix-to-Unix transfers (it must be given to *both* Unix Kermit programs), where it will improve performance by circumventing the normal text-file conversions, and will allow mixture of text and binary files in a single file group.

`-w` Write-Protect -- Avoid filename collisions for incoming files.

`-e` *n* Extended packet length -- Specify that C-Kermit is allowed to receive packets up to length *n*, where *n* may be between 10 and some large number, like 1000, depending on the system. The default maximum length for received packets is 90. Packets longer than 94 will be used only if the other Kermit supports, and agrees to use, the "long packet" protocol extension.

`-q` Quiet -- Suppress screen update during file transfer, for instance to allow a file transfer to proceed in the background.

`-d` Debug -- Record debugging information in the file `debug.log` in the current directory. Use this option if you believe the program is misbehaving, and show the resulting log to your local Kermit maintainer.

`-h` Help -- Display a brief synopsis of the command line options.

The command line may contain no more than one protocol action option.

Files are sent with their own names, except that lowercase letters are raised to upper, pathnames are stripped off, certain special characters like (‘~’) and (‘#’) are changed to ‘X’, and if the file name begins with a period, an ‘X’ is inserted before it. Incoming files are stored under their own names except that uppercase letters are lowered, and, if `-w` was specified, a "generation number" is appended to the name if it has the same name as an existing file which would otherwise be overwritten. If the `-a` option is included, then the same rules apply to its argument. The file transfer display shows any transformations performed upon filenames.

During transmission, files are encoded as follows:

- Control characters are converted to prefixed printables.
- Sequences of repeated characters are collapsed via repeat counts, if the other Kermit is also capable of repeated-character compression.
- If parity is being used on the communication line, data characters with the 8th (parity) bit on are specially prefixed, provided the other Kermit is capable of 8th-bit prefixing; if not, 8-bit binary files cannot be successfully transferred.
- Conversion is done between Unix newlines and carriage-return-linefeed sequences unless the `-i` option was specified.

Command Line Examples:

```
kermit -l /dev/ttyi5 -b 1200 -cn -r
```

This command connects you to the system on the other end of `ttyi5` at 1200 baud, where you presumably log in and run Kermit with a ’send’ command. After you escape back, C-Kermit waits for a file (or files) to arrive. When the file transfer is completed, you are reconnected to the remote system so that you can logout.

```
kermit -l /dev/ttyi4 -b 1800 -cntp m -r -a foo
```

This command is like the preceding one, except the remote system in this case uses half duplex communication with mark parity. The first file that arrives is stored under the name `foo`.

```
kermit -l /dev/ttyi6 -b 9600 -c | tek
```

This example uses Kermit to connect your terminal to the system at the other end of `ttyi6`. The C-Kermit terminal connection does not provide any particular terminal emulation, so C-Kermit’s standard i/o is piped through a (hypothetical) program called tek, which performs (say) Tektronix emulation.

```
kermit -l /dev/ttyi6 -b 9600 -nf
```

This command would be used to shut down a remote server and then connect to the remote system, in order to log out or to make further use of it. The `-n` option is invoked *after* `-f` (`-c` would have been invoked before).

```
kermit -l /dev/ttyi6 -b 9600 -qg foo.\* &
```

This command causes C-Kermit to be invoked in the background, getting a group of files from a remote server (note the quoting of the ‘`*`’ character). No display occurs on the screen, and the keyboard is not sampled for interruption commands. This allows other work to be done while file transfers proceed in the background.

```
kermit -l /dev/ttyi6 -b 9600 -g foo.\* > foo.log < /dev/null &
```

This command is like the previous one, except the file transfer display has been redirected to the file `foo.log`. Standard input is also redirected, to prevent C-Kermit from sampling it for interruption commands.

```
kermit -iwx
```

This command starts up C-Kermit as a server. Files are transmitted with no newline/carriage-return-linefeed conversion; the `-i` option is necessary for binary file transfer and recommended for Unix-to-Unix transfers.

Incoming files that have the same names as existing files are given new, unique names.

```
kermit -l /dev/ttyi6 -b 9600
```

This command sets the communication line and speed. Since no action is specified, C-Kermit issues a prompt and enters an interactive dialog with you. Any settings given on the command line remain in force during the dialog, unless explicitly changed.

```
kermit
```

This command starts up Kermit interactively with all default settings.

The next example shows how Unix Kermit might be used to send an entire directory tree from one Unix system to another, using the tar program as Kermit's standard input and output. On the orginating system, in this case the remote, type (for instance):

```
tar cf - /usr/fdc | kermit -is -
```

This causes tar to send the directory `/usr/fdc` (and all its files and all its subdirectories and all their files...) to standard output instead of to a tape; kermit receives this as standard input and sends it as a binary file. On the receiving system, in this case the local one, type (for instance):

```
kermit -il /dev/ttyi5 -b 9600 -k | tar xf -
```

Kermit receives the tar archive, and sends it via standard output to its own copy of tar, which extracts from it a replica of the original directory tree.

A final example shows how a Unix compression utility might be used to speed up Kermit file transfers:

```
compress file | kermit -is -     (sender)
kermit -ik | uncompress          (receiver)
```

Exit Status Codes:

Unix Kermit returns an exit status of zero, except when a fatal error is encountered, where the exit status is set to one. With background operation (e.g., '&' at end of invoking command line) driven by scripted interactive commands (redirected standard input and/or take files), any failed interactive command (such as failed dial or script attempt) causes the fatal error exit.

1.4. Interactive Operation

C-Kermit's interactive command prompt is "`C-Kermit>`". In response to this prompt, you may type any valid interactive C-Kermit command. C-Kermit executes the command and then prompts you for another command. The process continues until you instruct the program to terminate.

Commands begin with a keyword, normally an English verb, such as "send". You may omit trailing characters from any keyword, so long as you specify sufficient characters to distinguish it from any other keyword valid in that field. Certain commonly-used keywords (such as "send", "receive", "connect") also have special non-unique abbreviations ("s" for "send", "r" for "receive", "c" for "connect").

Certain characters have special functions during typein of interactive commands:

- `?` Question mark, typed at any point in a command, will produce a message explaining what is possible or expected at that point. Depending on the context, the message may be a brief phrase, a menu of keywords, or a list of files.
- `ESC` (The Escape or Altmode key) -- Request completion of the current keyword or filename, or insertion of a default value. The result will be a beep if the requested operation fails.

- `TAB` (The horizontal Tab key) -- Same as ESC.
- `DEL` (The Delete or Rubout key) -- Delete the previous character from the command. You may also use BS (Backspace, Control-H) for this function.
- `^W` (Control-W) -- Erase the rightmost word from the command line.
- `^U` (Control-U) -- Erase the entire command.
- `^R` (Control-R) -- Redisplay the current command.
- `SP` (Space) -- Delimits fields (keywords, filenames, numbers) within a command.
- `CR` (Carriage Return) -- Enters the command for execution. LF (Linefeed) or FF (formfeed) may also be used for this purpose.
- `\` (Backslash) -- Enter any of the above characters into the command, literally. To enter a backslash, type two backslashes in a row (`\\`). A backslash at the end of a command line causes the next line to be treated as a continuation line; this is useful for readability in command files, especially in the 'script' command.
- `^Z` (Control-Z) -- On systems (like Berkeley Unix, Ultrix) with job control, suspend Kermit, i.e. put it into the background in such a way that it can be brought back into the foreground (e.g. with an '`fg`' shell command) with all its settings intact.

You may type the editing characters (`DEL`, `^W`, etc) repeatedly, to delete all the way back to the prompt. No action will be performed until the command is entered by typing carriage return, linefeed, or formfeed. If you make any mistakes, you will receive an informative error message and a new prompt -- make liberal use of '`?`' and ESC to feel your way through the commands. One important command is "help" -- you should use it the first time you run C-Kermit.

A command line beginning with a percent sign ''`%`'' is ignored. Such lines may be used to include illustrative commentary in Kermit command dialogs.

Interactive C-Kermit accepts commands from files as well as from the keyboard. When you start C-Kermit, the program looks for the file `.kermrc` in your home or current directory (first it looks in the home directory, then in the current one) and executes any commands it finds there. These commands must be in interactive format, not Unix command-line format. A "take" command is also provided for use at any time during an interactive session, to allow interactive-format commands to be executed from a file; command files may be nested to any reasonable depth.

Here is a brief list of C-Kermit interactive commands:

%	Comment
!	Execute a Unix shell command, or start a shell.
bye	Terminate and log out a remote Kermit server.
close	Close a log file.
connect	Establish a terminal connection to a remote system.
cwd	Change Working Directory (also, cd).
dial	Dial a telephone number.
directory	Display a directory listing.
echo	Display arguments literally.
exit	Exit from the program, closing any open files.
finish	Instruct a remote Kermit server to exit, but not log out.
get	Get files from a remote Kermit server.
hangup	Hang up the phone (for use in local mode).
help	Display a help message for a given command.
log	Open a log file -- debugging, packet, session, transaction.
quit	Same as 'exit'.
receive	Passively wait for files to arrive.
remote	Issue file management commands to a remote Kermit server.
script	Execute a login script with a remote system.
send	Send files.

server	Begin server operation.
set	Set various parameters.
show	Display values of 'set' parameters.
space	Display current disk space usage.
statistics	Display statistics about most recent transaction.
take	Execute commands from a file.

The 'set' parameters are:

block-check	Level of packet error detection.
delay	How long to wait before sending first packet.
duplex	Specify which side echoes during 'connect'.
escape-character	Prefix for "escape commands" during 'connect'.
file	Set various file parameters.
flow-control	Communication line full-duplex flow control.
handshake	Communication line half-duplex turnaround character.
incomplete	Disposition for incompletely received files.
line	Communication line device name.
modem-dialer	Type of modem-dialer on communication line.
parity	Communication line character parity.
prompt	The C-Kermit program's interactive command prompt.
receive	Parameters for inbound packets.
retry	Packet retransmission limit.
send	Parameters for outbound packets.
speed	Communication line speed.
terminal	Terminal parameters.

The 'remote' commands are:

cwd	Change remote working directory.
delete	Delete remote files.
directory	Display a listing of remote file names.
help	Request help from a remote server.
host	A command to the remote host in its own command language.
space	Display current disk space usage on remote system.
type	Display a remote file on your screen.
who	Display who's logged in, or get information about a user.

Most of these commands are described adequately in the Kermit User Guide or the Kermit book. Special aspects of certain Unix Kermit commands are described below.

The 'send' command

Syntax: `send` *fn* - *or* - `send` *fn1 rfn1*

Send the file or files denoted by *fn* to the other Kermit, which should be running as a server, or which should be given the 'receive' command. Each file is sent under its own name (as described above, or as specified by the 'set file names' command). If the second form of the 'send' command is used, i.e. with *fn1* denoting a single Unix file, *rfn1* may be specified as a name to send it under. The 'send' command may be abbreviated to 's', even though 's' is not a unique abbreviation for a top-level C-Kermit command.

The wildcard (meta) characters '`*`' and '`?`' are accepted in *fn*. If '`?`' is to be included, it must be prefixed by '`\`' to override its normal function of providing help. '`*`' matches any string, '`?`' matches any single character. Other notations for file groups, like '`[a-z]og`', are not available in interactive commands (though of course they are available on the command line). When *fn* contains '`*`' or '`?`' characters, there is a limit to the number of files that can be matched, which varies from system to system. If you get the message "Too many files match" then you'll have to make a more judicious selection. If *fn* was of the form

```
usr/longname/anotherlongname/*
```

then C-Kermit's string space will fill up rapidly -- try doing a cwd (see below) to the path in question and reissuing the command.

In interactive mode, C-Kermit does not presently understand "~" notation for "home directory", as used in the C-Shell and K-Shell, nor does it understand "`[abc]`" or "`{txt,doc}`" filename metacharacter notation.

Note -- C-Kermit sends only from the current or specified directory. It does not traverse directory trees. If the source directory contains subdirectories, they will be skipped. By the same token, C-Kermit does not create directories when receiving files. If you have a need to do this, you can pipe tar through C-Kermit, as shown in the example on page 7, or under System III/V Unix you can use cpio.

Another Note -- The 'send' command does not skip over "invisible" files that match the file specification; Unix systems usually treat files whose names start with a dot (like `.login`, `.cshrc`, and `.kermrc`) as invisible. Similarly for "temporary" files whose names start with "#".

The 'receive' command

Syntax: `receive` - *or* - `receive` *fn1*

Passively wait for files to arrive from the other Kermit, which must be given the 'send' command -- the 'receive' command does not work in conjunction with a server (use 'get' for that). If *fn1* is specified, store the first incoming file under that name. The 'receive' command may be abbreviated to 'r'.

The 'get' command:

Syntax: `get` *rfn*

or: `get`
rfn
fn1

Request a remote Kermit server to send the named file or files. Since a remote file specification (or list) might contain spaces, which normally delimit fields of a C-Kermit command, an alternate form of the command is provided to allow the inbound file to be given a new name: type 'get' alone on a line, and you will be prompted separately for the remote and local file specifications, for example

```
C-Kermit>get
 Remote file specification: profile exec
 Local name to store it under: profile.exec
```

As with 'receive', if more than one file arrives as a result of the 'get' command, only the first will be stored under the alternate name given by *fn1*; the remaining files will be stored under their own names if possible. If a '?' is to be included in the remote file specification, you must prefix it with '\' to suppress its normal function of providing help.

If you have started a multiline 'get' command, you may escape from its lower-level prompts by typing a carriage return in response to the prompt, e.g.

```
C-Kermit>get
 Remote file specification: foo
 Local name to store it under: (Type a carriage return here)
(cancelled)
C-Kermit>
```

The 'server' command:

The 'server' command places C-Kermit in "server mode" on the currently selected communication line. All further commands must arrive as valid Kermit packets from the Kermit on the other end of the line. The Unix Kermit server can respond to the following commands:

Command	Server Response
get	Sends files
send	Receives files
bye	Attempts to log itself out
finish	Exits to level from which it was invoked
remote directory	Sends directory lising
remote delete	Removes files
remote cwd	Changes working directory (also, remote cd)
remote type	Sends files to your screen
remote space	Reports about its disk usage
remote who	Shows who's logged in
remote host	Executes a Unix shell command
remote help	Lists these capabilities

The Unix Kermit server cannot always respond properly to a BYE command. It will attempt to do so using "`kill()`", but this will not work on all systems or under all conditions because of the complicated process structures that can be set up under Unix.

If the Kermit server is directed at an external line (i.e. it is in "local mode") then the console may be used for other work if you have 'set file display off'; normally the program expects the console to be used to observe file transfers and enter status queries or interruption commands. The way to get C-Kermit into background operation from interactive command level varies from system to system (e.g. on Berkeley Unix you would halt the program with `^Z` and then use the C-Shell 'bg' command to continue it in the background). The more common method is to invoke the program with the desired command line arguments, including "`-q`", and with a terminating "`&`".

When the Unix Kermit server is given a 'remote host' command, it executes it using the shell invoked upon login, e.g. the Bourne shell or the Berkeley C-Shell.

The 'remote', 'bye', and 'finish' commands:

C-Kermit may itself request services from a remote Kermit server. In addition to 'send' and 'get', the following commands may also be sent from C-Kermit to a Kermit server:

remote cwd [*directory*]

> If the optional remote directory specification is included, you will be prompted on a separate line for a password, which will not echo as you type it. If the remote system does not require a password for this operation, just type a carriage return. 'remote cd' is a synomym for this command.

remote delete rfn	delete remote file or files.
remote directory [*rfn*]	directory listing of remote files.
remote host *command*	command in remote host's own command language.
remote space	disk usage report from remote host.
remote type [*rfn*]	display remote file or files on the screen.
remote who [*user*]	display information about who's logged in.
remote help	display remote server's capabilities.

bye *and* finish:

> When connected to a remote Kermit server, these commands cause the remote server to terminate; 'finish' returns it to Kermit or system command level (depending on the implementation or how the program was invoked); 'bye' also requests it to log itself out.

The 'log' and 'close' commands:

Syntax: `log {debugging, packets, session, transactions}` [*fn1*]

C-Kermit's progress may be logged in various ways. The 'log' command opens a log, the 'close' command closes it. In addition, all open logs are closed by the 'exit' and 'quit' commands. A name may be specified for a log file; if the name is omitted, the file is created with a default name as shown below.

log debugging
: This produces a voluminous log of the internal workings of C-Kermit, of use to Kermit developers or maintainers in tracking down suspected bugs in the C-Kermit program. Use of this feature dramatically slows down the Kermit protocol. Default name: `debug.log`.

log packets
: This produces a record of all the packets that go in and out of the communication port. This log is of use to Kermit maintainers who are tracking down protocol problems in either C-Kermit or any Kermit that C-Kermit is connected to. Default name: `packet.log`.

log session
: This log will contain a copy of everything you see on your screen during the 'connect' command, except for local messages or interaction with local escape commands. Default name: `session.log`.

log transactions
: The transaction log is a record of all the files that were sent or received while transaction logging was in effect. It includes time stamps and statistics, filename transformations, and records of any errors that may have occurred. The transaction log allows you to have long unattended file transfer sessions without fear of missing some vital screen message. Default name: `transact.log`.

The 'close' command explicitly closes a log, e.g. 'close debug'.

Note: Debug and Transaction logs are a compile-time option; C-Kermit may be compiled without these logs, in which case it will run faster, it will take up less space on the disk, and the commands relating to them will not be present.

Local File Management Commands:

Unix Kermit allows some degree of local file management from interactive command level:

directory [*fn*]
: Displays a listing of the names, modes, sizes, and dates of files matching *fn* (which defaults to '`*`'). Equivalent to '`ls -l`'.

cwd [directory-name]
: Changes Kermit's working directory to the one given, or to the default directory if the directory name is omitted. This command affects only the Kermit process and any processes it may subsequently create. You may also type "cd" instead of "cwd".

space
: Display information about disk space and/or quota in the current directory and device.

`!` [*command*]
: The command is executed by the Unix shell. If no command is specified, then an interactive shell is started; exiting from the shell, e.g. by typing Control-D or 'exit', will return you to C-Kermit command level. Use the '`!`' command to provide file management or other functions not explicitly provided by C-Kermit commands. The '`!`' command has certain peculiarities:

- C-Kermit attempts to use your preferred, customary (login) shell.
- At least one space must separate the '!' from the shell command.
- A 'cd' (change directory) command executed in this manner will have no effect -- use the C-Kermit 'cwd' command instead.

The 'set' and 'show' Commands:

Since Kermit is designed to allow diverse systems to communicate, it is often necessary to issue special instructions to allow the program to adapt to peculiarities of the another system or the communication path. These instructions are accomplished by the 'set' command. The 'show' command may be used to display current settings. Here is a brief synopsis of settings available in the current release of C-Kermit:

block-check {1, 2, 3}
: Determines the level of per-packet error detection. "1" is a single-character 6-bit checksum, folded to include the values of all bits from each character. "2" is a 2-character, 12-bit checksum. "3" is a 3-character, 16-bit cyclic redundancy check (CRC). The higher the block check, the better the error detection and correction and the higher the resulting overhead. Type 1 is most commonly used; it is supported by all Kermit implementations, and it has proven adequate in most circumstances. Types 2 or 3 would be used to advantage when transferring 8-bit binary files over noisy lines.

delay *n*
: How many seconds to wait before sending the first packet after a 'send' command. Used in remote mode to give you time to escape back to your local Kermit and issue a 'receive' command. Normally 5 seconds.

duplex {full, half}
: For use during 'connect'. Specifies which side is doing the echoing; 'full' means the other side, 'half' means C-Kermit must echo typein itself.

escape-character *cc*
: For use during 'connect' to get C-Kermit's attention. The escape character acts as a prefix to an 'escape command', for instance to close the connection and return to C-Kermit or Unix command level. The normal escape character is Control-Backslash (28). The escape character is also used in System III/V implementations to prefix interrupt commands during file transfers.

file {display, names, type, warning}
: Establish various file-related parameters:

  display {on, off}
  : Normally 'on'; when in local mode, display progress of file transfers on the screen (stdout), and listen to the keyboard (stdin) for interruptions. If off (-q on command line) none of this is done, and the file transfer may proceed in the background oblivious to any other work concurrently done at the console terminal.

  names {converted, literal}
  : Normally converted, which means that outbound filenames have path specifications stripped, lowercase letters raised to upper, tildes and extra periods changed to X's, and an X inserted in front of any name that starts with period. Incoming files have uppercase letters lowered. Literal means that none of these conversions are done; therefore, any directory path appearing in a received file specification must exist and be write-accessible. When literal naming is being used, the sender should not use path names in the file specification unless the same path exists on the target system and is writable.

  type {binary, text} [{7, 8}]
  : The file type is normally text, which means that conversion is done between Unix newline characters and the carriage-return/linefeed sequences required by the canonical Kermit file transmission format, and in common use on non-Unix systems. Binary means to transmit file contents without conversion. Binary ('`-i`' in command line notation) is necessary for binary files, and desirable in all Unix-to-Unix transactions to cut down on overhead.

    The optional trailing parameter tells the bytesize for file transfer. It is 8 by default. If you specify 7, the high order bit will be stripped from each byte of sent and received files. This is useful for transferring text files that may have extraneous high order bits set in their disk representation (e.g. Wordstar or similar word processor files).

  warning {on, off}
  : Normally off, which means that incoming files will silently overwrite existing files of the same name. When on ('`-w`' on command line) Kermit will check if an arriving file would overwrite an existing file; if so, it will construct a new name for the arriving file, of the form `foo~`*n*, where foo is the name they share and *n* is a "generation number"; if *foo* exists, then the new file will be called `foo~1`. If `foo` and `foo~1`

exist, the new file will be `foo~2`, and so on. If the new name would be longer than the maximum length for a filename, then characters would be deleted from the end first, for instance, `thelongestname` on a system with a limit of 14 characters would become `thelongestn~1`.

> *CAUTION:* If Control-F or Control-B is used to cancel an incoming file, and a file of the same name previously existed, *and* the "file warning" feature is not enabled, then the previous copy of the file will disappear.

flow-control {none, xon/xoff}
: Normally xon/xoff for full duplex flow control. Should be set to 'none' if the other system cannot do xon/xoff flow control, or if you have issued a 'set handshake' command. If set to xon/xoff, then handshake should be set to none. This setting applies during both terminal connection and file transfer. *Warning:* This command may have no effect on certain Unix systems, where Kermit puts the communication line into 'rawmode', and rawmode precludes flow control.

incomplete {discard, keep}
: Disposition for incompletely received files. If an incoming file is interrupted or an error occurs during transfer, the part that was received so far is normally discarded. If you "set incomplete keep" then such file fragments will be kept.

handshake {xon, xoff, cr, lf, bell, esc, none}
: Normally none. Otherwise, half-duplex communication line turnaround handshaking is done, which means Unix Kermit will not reply to a packet until it has received the indicated handshake character or has timed out waiting for it; the handshake setting applies only during file transfer. If you set handshake to other than none, then flow should be set to none.

line [device-name]
: The device name for the communication line to be used for file transfer and terminal connection, e.g. `/dev/ttyi3`. If you specify a device name, Kermit will be in local mode, and you should remember to issue any other necessary 'set' commands, such as 'set speed'. If you omit the device name, Kermit will revert to its default mode of operation. If you specify `/dev/tty`, Kermit will enter remote mode (useful when logged in through the "back port" of a system normally used as a local-mode workstation). When Unix Kermit enters local mode, it attempts to synchronize with other programs (like uucp) that use external communication lines so as to prevent two programs using the same line at once; before attempting to lock the specified line, it will close and unlock any external line that was previously in use. The method used for locking is the "uucp lock file", explained in more detail later.

modem-dialer {direct, hayes, racalvadic, ventel, ...}
: The type of modem dialer on the communication line. "Direct" indicates either there is no dialout modem, or that if the line requires carrier detection to open, then 'set line' will hang waiting for an incoming call. "Hayes", "Ventel", and the others indicate that 'set line' (or the -l argument) will prepare for a subsequent 'dial' command for the given dialer. Support for new dialers is added from time to time, so type 'set modem ?' for a list of those supported in your copy of Kermit. See the description of the 'dial' command. *NOTE:* the "set modem" command must be given *before* the "set line" command.

parity {even, odd, mark, space, none}
: Specify character parity for use in packets and terminal connection, normally none. If other than none, C-Kermit will seek to use the 8th-bit prefixing mechanism for transferring 8-bit binary data, which can be used successfully only if the other Kermit agrees; if not, 8-bit binary data cannot be successfully transferred.

prompt [string]
: The given string will be substituted for "`C-Kermit>`" as this program's prompt. If the string is omitted, the prompt will revert to "`C-Kermit>`". If the string is enclosed in doublequotes, the quotes will be stripped and any leading and trailing blanks will be retained.

send *parameter*
: Establish parameters to use when sending packets. These will be in effect only for the initial packet sent, since the other Kermit may override these parameters during the protocol parameter exchange (unless noted below).

    end-of-packet *cc*
    : Specifies the control character needed by the other Kermit to recognize the end of a packet. C-Kermit sends this character at the end of each packet. Normally 13 (carriage return), which most Kermit implementations require. Other Kermits require no terminator at all, still others may require a different terminator, like linefeed (10).

packet-length *n*
: Specify the maximum packet length to send. Normally 90. Shorter packet lengths can be useful on noisy lines, or with systems or front ends or networks that have small buffers. The shorter the packet, the higher the overhead, but the lower the chance of a packet being corrupted by noise, and the less time to retransmit corrupted packets. This command overrides the value requested by the other Kermit during protocol initiation unless the other Kermit requests a shorter length.

pad-character *cc*
: Designate a character to send before each packet. Normally, none is sent. Outbound padding is sometimes necessary for communicating with slow half duplex systems that provide no other means of line turnaround control. It can also be used to send special characters to communications equipment that needs to be put in "transparent" or "no echo" mode, when this can be accomplished in by feeding it a certain control character.

padding *n*
: How many pad characters to send, normally 0.

start-of-packet *cc*
: The normal Kermit packet prefix is Control-A (1); this command changes the prefix C-Kermit puts on outbound packets. The only reasons this should ever be changed would be: Some piece of equipment somewhere between the two Kermit programs will not pass through a Control-A; or, some piece of of equipment similarly placed is echoing its input. In the latter case, the recipient of such an echo can change the packet prefix for outbound packets to be different from that of arriving packets, so that the echoed packets will be ignored. The opposite Kermit must also be told to change the prefix for its inbound packets.

timeout *n*
: Specifies the number of seconds you want the other Kermit to wait for a packet before timing it out and requesting retransmission.

receive *parameter*
: Establish parameters to request the other Kermit to use when sending packets.

end-of-packet *cc*
: Requests the other Kermit to terminate its packets with the specified character.

packet-length *n*
: Specify the maximum packet length to that you want the other Kermit to send, normally 90. If you specify a length of 95 or greater, then it will be used if the other Kermit supports, and agrees to use, the Kermit protocol extension for long packets. In this case, the maximum length depends upon the systems involved, but there would normally be no reason for packets to be more than about 1000 characters in length. The 'show parameters' command displays C-Kermit's current and maximum packet lengths.

pad-character *cc*
: C-Kermit normally does not need to have incoming packets preceded with pad characters. This command allows C-Kermit to request the other Kermit to use *cc* as a pad character. Default *cc* is NUL, ASCII 0.

padding *n*
: How many pad characters to ask for, normally 0.

start-of-packet *cc*
: Change the prefix C-Kermit looks for on inbound packets to correspond with what the other Kermit is sending.

timeout *n*
: Normally, each Kermit partner sets its packet timeout interval based on what the opposite Kermit requests. This command allows you to override the normal procedure and specify a timeout interval for Unix Kermit to use when waiting for packets from the other Kermit. If you specify 0, then no timeouts will occur, and Unix Kermit will wait forever for expected packets to arrive.

speed {0, 110, 150, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200}
: The transmission speed ("baud rate") for the external communication line. This command cannot be used to change the speed of your own console terminal. Many Unix systems are set up in such a way that you must give this command after a 'set line' command before you can use the line. 'set baud' is a synomym for 'set speed'. Use 19200 with caution -- it may not work on all systems.

terminal
Used for specifying terminal parameters. Currently, ’bytesize’ is the only parameter provided, and it can be set to 7 or 8. It’s 7 by default.

The ’show’ Command:

Syntax: `show {parameters, versions}`

The "show" command with the default argument of "parameters" displays the values of all the ’set’ parameters described above. If you type "show versions", then C-Kermit will display the version numbers and dates of all its internal modules. You should use the "show versions" command to ascertain the vintage of your Kermit program before reporting problems to Kermit maintainers.

The ’statistics’ Command:

The statistics command displays information about the most recent Kermit protocol transaction, including file and communication line i/o, timing and efficiency, as well as what encoding options were in effect (such as 8th-bit prefixing, repeat-count compression).

The ’take’ and ’echo’ Commands:

Syntax: `take` *fn1*
`echo` *[text to be echoed]*

The ’take’ command instructs C-Kermit to execute commands from the named file. The file may contain any interactive C-Kermit commands, including ’take’; command files may be nested to any reasonable depth, but it may not contain text to be sent to a remote system during the ’connect’ command. This means that a command file like this:

```
set line /dev/tty17
set speed 9600
connect
login myuserid
mypassword
etc
```

will not send "login myserid" or any of the following text to the remote system. To carry on a canned dialog, use the ’script’ command, described later.

The ’`%`’ command is useful for including comments in take-command files. It may only be used at the beginning of a line.

The ’echo’ command may be used within command files to issue greetings, announce progress, ring the terminal bell, etc. The ’echo’ command should not be confused with the Unix ’echo’ command, which can be used to show how meta characters would be expanded. The Kermit echo command simply displays its text argument (almost) literally at the terminal; the argument may contain octal escapes of the form ‘‘`\ooo`’’, where `o` is an octal digit (0-7), and there may be 1, 2, or 3 such digits, whose value specify an ASCII character, such as ‘‘`\007`’’ (or ‘‘`\07`’’ or just ‘‘`\7`’’) for beep, ‘‘`\012`’’ for newline, etc. Of course, each backslash must be must be entered twice in order for it to be passed along to the echo command by the Kermit command parser.

Take-command files are in exactly the same syntax as interactive commands. Note that this implies that if you want to include special characters like question mark or backslash that you would have to quote with backslash when typing interactive commands, you must quote these characters the same way in command files. Long lines may be continued by ending them with a single backslash.

Command files may be used in lieu of command macros, which have not been implemented in this version of C-Kermit. For instance, if you commonly connect to a system called 'B' that is connected to ttyh7 at 4800 baud, you could create a file called `b` containing the commands

```
% C-Kermit command file to connect to System B thru /dev/ttyh7
set line /dev/ttyh7
set speed 4800
% Beep and give message
echo \\007Connecting to System B...
connect
```

and then simply type 'take b' (or 't b' since no other commands begin with the letter 't') whenever you wish to connect to system B. Note the comment lines and the beep inserted into the 'echo' command.

For connecting to IBM mainframes, a number of 'set' commands are required; these, too, can be conveniently collected into a 'take' file like this one:

```
% Sample C-Kermit command file to set up current line
% for IBM mainframe communication
%
set parity mark
set handshake xon
set flow-control none
set duplex half
```

Note that no single command is available to wipe out all of these settings and return C-Kermit to its default startup state; to do that, you can either restart the program, or else make a command file that executes the necessary 'set' commands:

```
% Sample C-Kermit command file to restore normal settings
%
set parity none
set handshake none
set flow-control xon/xoff
set duplex full
```

An implicit 'take' command is executed upon your `.kermrc` file when C-Kermit starts up, upon either interactive or command-line invocation. The `.kermrc` file should contain 'set' or other commands you want to be in effect at all times. For instance, you might want override the default action when incoming files have the same names as existing files -- in that case, put the command

```
set file warning on
```

in your `.kermrc` file. On some non-Unix systems that run C-Kermit, the initialization file might have a different name, such as `kermit.ini`.

Errors encountered during execution of take files (such as failure to complete dial or script operations) cause termination of the current take file, popping to the level that invoked it (take file, interactive level, or the shell). When kermit is executed in the background, errors during execution of a take file are fatal.

Under Unix, you may also use the shell's redirection mechanism to cause C-Kermit to execute commands from a file:

```
kermit < cmdfile
```

or you can even pipe commands in from another process:

```
command | kermit
```

The ’connect’ Command:

The ’connect’ command (’c’ is an acceptable non-unique abbreviation for ’connect’) links your terminal to another computer as if it were a local terminal to that computer, through the device specified in the most recent ’set line’ command, or through the default device if your system is a PC or workstation. All characters you type at your keyboard are sent out the communication line (and if you have ’set duplex half’, also displayed on your screen), and all characters arriving at the communication port are displayed on the screen. Current settings of speed, parity, duplex, and flow-control are honored, and the data connection is 7 bits wide unless you have given the command ’set terminal bytesize 8’. If you have issued a ’log session’ command, everything you see on your screen will also be recorded to your session log. This provides a way to "capture" files from remote systems that don’t have Kermit programs available.

To get back to your own system, you must type the escape character, which is Control-Backslash (`^\`) unless you have changed it with the ’set escape’ command, followed by a single-character command, such as ’c’ for "close connection". Single-character commands include:

c	Close the connection
b	Send a BREAK signal
0	(zero) send a null
s	Give a status report about the connection
h	Hangup the phone
`^\`	Send Control-Backslash itself (whatever you have defined the escape character to be, typed twice in a row sends one copy of it).

Uppercase and control equivalents for (most of) these letters are also accepted. A space typed after the escape character is ignored. Any other character will produce a beep.

The connect command simply displays incoming characters on the screen. It is assumed any screen control sequences sent by the host will be handled by the firmware or emulation software in your terminal or PC. If special terminal emulation is desired, then the ’connect’ command can invoked from the Unix command line (`-c` or `-n`), piped through a terminal emulation filter, e.g.

```
kermit -l /dev/acu -b 1200 -c | tek
```

The ’hangup’ command:

The ’hangup’ command attempts to hang up the modem on a local-mode dialout connection.

The ’dial’ command:

Syntax: `dial` *telephone-number-string*

This command controls dialout modems; you should have already issued a "set line" and "set speed" command to identify the terminal device, and a "set modem" command to identify the type of modem to be used for dialing. In the "dial" command, you supply the phone number and the Kermit program feeds it to the modem in the appropriate format and then interprets dialer return codes and modem signals to inform you whether the call was completed. The telephone-number-string may contain imbedded modem-dialer commands, such as comma for Hayes pause, or ‘&’ for Ventel dialtone wait and ‘%’ for Ventel pause (consult your modem manual for details).

At the time of this writing, support is included for the following modems:

- AT&T 7300 Internal Modem
- Cermetek Info-Mate 212A

- Concord Condor CDS 220
- DEC DF03-AC
- DEC DF100 Series
- DEC DF200 Series
- General DataComm 212A/ED
- Hayes Smartmodem and compatibles
- Penril
- Racal Vadic
- Rolm CBX
- US Robotics 212A
- Ventel

Support for new modems is added to the program from time to time; you can check the current list by typing "`set modem ?`".

There are also two "generic" modem types -- "direct" (i.e. no modem at all, so that no attempt is made to deal with modem signals), and "unknown" (which tells C-Kermit to attempt to honor modem signals, but leaves the dialing mechanism unspecified).

The device used for dialing out is the one selected in the most recent "set line" command (or on a workstation, the default line if no "set line" command was given). The "dial" command calls attempts to lock the terminal device's path (see the section on line locking below) and to establish a call on an exclusive basis. If it is desired to dial a call and then return to the shell (such as to do kermit activities depending on standard in/out redirection), it is necessary to place the dialed call under one device name (say, "`/dev/cua0`") and then escape to the shell *within Kermit* on a linked device which is separate from the dialed line (say, "`/dev/cul0`"). This is the same technique used by uucp (to allow locks to be placed separately for dialing and conversing).

Because modem dialers have strict requirements to override the carrier-detect signal most Unix implementations expect, the sequence for dialing is more rigid than most other C-Kermit procedures.

Example one:

```
kermit -l /dev/cul0 -b 1200
C-Kermit>set modem-dialer hayes    hint: abbreviate set m h
C-Kermit>dial 9,5551212
Connected!
C-Kermit>connect                   hint: abbreviate c
logon, request remote server, etc.
^\c                                escape back
C-Kermit> ...
C-Kermit>quit                      hint: abbreviate q
```

this disconnects modem, and unlocks line.

Example two:

```
kermit
C-Kermit>set modem-dialer ventel
C-Kermit>set line /dev/cul0
C-Kermit>dial 9&5551212%
Connected!
C-Kermit> ...
```

Example three:

```
kermit
C-Kermit>take my-dial-procedure
Connected!
```

```
file my-dial-procedure:
set modem hayes
```

```
set line /dev/tty99
dial 5551212
connect
```

In general, C-Kermit requires that the modem provide the "carrier detect" (CD) signal when a call is in progress, and remove that signal when the call completes or the line drops. If a modem switch setting is available to force CD, it should normally not be in that setting. C-Kermit also requires (on most systems) that the modem track the computer's "data terminal ready" (DTR) signal. If a switch setting is available to simulate DTR asserted within the modem, then it should normally not be in that setting. Otherwise the modem will be unable to hang up at the end of a call or when interrupts are received by Kermit.

For Hayes 1200 dialers, two important switch settings are #1 and #6. Switch #1 should be normally be UP so that the modem can act according to your computer's DTR signal. But if your computer, or particular implementation of Kermit, cannot control DTR, then switch 1 should be DOWN. Switch #6 should normally be UP so carrier-detect functions properly (but put it DOWN if you have trouble with the UP position). Switches #2 (English versus digit result codes) and #4 (Hayes echoes modem commands) may be in either position. Hayes 2400 modems have equivalent "software" switches.

If you want to interrupt a dial command in progress (for instance, because you just realize that you gave it the wrong number), type a Control-C to get back to command level.

The 'script' Command:

Syntax: `script` *expect send [expect send] . . .*

"expect" has the syntax: *expect[-send-expect[-send-expect[...]]]*

The 'script' command carries on a "canned dialog" with a remote system, in which data is sent according to the remote system's responses. The typical use is for logging in to a remote system automatically.

C-Kermit's script facility operates in a manner similar to that commonly used by the Unix UUCP system's "`L.sys`" file entries. A login script is a sequence of the form:

`expect send [expect send] . . .`

where *expect* is a prompt or message to be issued by the remote site, and *send* is the string (names, numbers, etc) to return, and expects are separated from sends by spaces. The send may also be the keyword EOT, to send Control-D, or BREAK, to send a break signal. Letters in sends may be prefixed by '~' to send special characters, including:

`~b` backspace
`~s` space
`~q` '`?`'(trapped by Kermit's command interpreter)
`~n` linefeed
`~r` carriage return
`~t` tab
`~'` single quote
`~~` tilde
`~"` double quote
`~x` XON (Control-Q)
`~c` don't append a carriage return
~*o[o[o]]* an octal character
`~d` delay approx 1/3 second during send
`~w`*[d[d]]* wait specified interval during expect, then time out

As with some UUCP systems, sent strings are followed by `~r` unless they have a `~c`.

Only the last 7 characters in each expect are matched. A null *expect*, e.g. `~0` or two adjacent dashes, causes a short delay before proceeding to the next send sequence. A null expect always succeeds.

As with UUCP, if the expect string does not arrive, the script attempt fails. If you expect that a sequence might not arrive, as with UUCP, conditional sequences may be expressed in the form:

```
-send-expect[-send-expect[...]]
```

where dashed sequences are followed as long as previous expects fail. Timeouts for expects can be specified using `~w`; `~w` with no arguments waits 15 seconds.

Expect/send transactions can be easily be debugged by logging transactions. This records all exchanges, both expected and actual. The script execution will also be logged in the session log, if that is activated.

Note that '\' characters in login scripts, as in any other C-Kermit interactive commands, must be doubled up. A line may be ended with a single '\' for continuation.

Example one:

Using a modem, dial a UNIX host site. Expect "login" (...gin), and if it doesn't come, simply send a null string with a `~r`. (Some Unixes require either an EOT or a BREAK instead of the null sequence, depending on the particular site's "logger" program.) After providing user id and password, respond "x" to a question-mark prompt, expect the Bourne shell "`$`" prompt (and send return if it doesn't arrive). Then cd to directory kermit, and run the program called "wermit", entering the interactive connect state after wermit is loaded.

```
set modem ventel
set line /dev/tty77
set baud 1200
dial 9&5551212
script gin:--gin:--gin: smith ssword: mysecret ~q x $--$ \
 cd~skermit $ wermit
connect
```

Note that 'set line' is issued *after* 'set modem', but *before* 'set baud' or other line-related parameters.

Example two:

Using a modem, dial the Telenet network. This network expects three returns with slight delays between them. These are sent following null expects. The single return is here sent as a null string, with a return appended by default. Four returns are sent to be safe before looking for the prompt. Then the Telenet id and password are entered. Then Telenet is instructed to connect to a host site (c 12345). The host has a data switch that asks "which system"; the script responds "myhost" (if the "which system" prompt doesn't appear, the Telenet connect command is reissued). The script waits for an ''`@`'' prompt from the host, then sends the user ID ("joe") and password ("secret"), looks for another ''`@`'' prompt, runs Kermit, and in response to the Kermit's prompt (which ends in ''>''), gives the commands "set parity even" and "server". Files are then exchanged. The commands are in a take file; note the continuation of the 'script' command onto several lines using the '\' terminator.

```
set modem hayes
set line /dev/acu
set speed 1200
set parity mark
dial 9,5551212
script ~0 ~0 ~0 ~0 ~0 ~0 ~0 ~0 @--@--@ id~saa001122 = 002211 @ \
    c~s12345 ystem-c~s12345-ystem myhost @ joe~ssecret @ kermit \
    > set~sparity~seven > server
send some.stuff
get some.otherstuff
bye
quit
```

Since these commands may be executed totally in the background, they can also be scheduled. A typical shell script, which might be scheduled by cron, would be as follows (csh used for this example):

```
#
#keep trying to dial and log onto remote host and exchange files
#wait 10 minutes before retrying if dial or script fail.
#
cd someplace
while ( 1 )
        kermit < /tonight.cmd >> nightly.log &
        if ( ! $status ) break
        sleep 600
end
```

File `tonight.cmd` might have two takes in it, for example, one to take a file with the set modem, set line, set baud, dial, and script, and a second take of a file with send/get commands for the remote server. The last lines of `tonight.cmd` should be a bye and a quit.

The 'help' Command:

Syntax: `help`
or: `help` *keyword*
or: `help {set, remote}` *keyword*

Brief help messages or menus are always available at interactive command level by typing a question mark at any point. A slightly more verbose form of help is available through the 'help' command. The 'help' command with no arguments prints a brief summary of how to enter commands and how to get further help. 'help' may be followed by one of the top-level C-Kermit command keywords, such as 'send', to request information about a command. Commands such as 'set' and 'remote' have a further level of help. Thus you may type 'help', 'help set', or 'help set parity'; each will provide a successively more detailed level of help.

The 'exit' and 'quit' Commands:

These two commands are identical. Both of them do the following:

- Attempt to insure that the terminal is returned to normal.
- Relinquish access to any communication line assigned via 'set line'.
- Relinquish any uucp and multiuser locks on the communications line.
- Hang up the modem, if the communications line supports data terminal ready.
- Close any open logs or other files.

After exit from C-Kermit, your default directory will be the same as when you started the program. The 'exit' command is issued implicitly whenever C-Kermit halts normally, e.g. after a command line invocation, or after certain kinds of interruptions.

1.5. UUCP Lock Files

Unix has no standard way of obtaining exclusive access to an external communication line. When you issue the 'set line' command to Unix Kermit, Unix would normally grant you access to the line even if some other process is making use of it. The method adopted by most Unix systems to handle this situation is the "UUCP lock file". UUCP, the Unix-to-Unix Copy program, creates a file in its directory (usually `/usr/spool/uucp`, on some systems `/etc/locks`) with a name like `LCK..`*name*, where *name* is the device name, for instance `tty07`.

Unix Kermit uses UUCP lock files in order to avoid conflicts with UUCP, tip, or other programs that follow this convention. Whenever you attempt to access an external line using the 'set line' command or '`-l`' on the command line, Kermit looks in the UUCP directory for a lock file corresponding to that device. For instance, if you 'set line /dev/ttyi6' then Kermit looks for the file

```
/usr/spool/uucp/LCK..ttyi6
```

If it finds this file, it gives you an error message and a directory listing of the file so that you can see who is using it, e.g.

```
-r--r--r--  1 fdc        4 May  7 13:02 /usr/spool/uucp/LCK..ttyi6
```

In this case, you would look up user fdc to find out how soon the line will become free.

This convention requires that the uucp directory be publicly readable and writable. If it is not, the program will issue an appropriate warning message, but will allow you to proceed at your own risk (and the risk of anyone else who might also be using the same line).

If no lock file is found, Unix Kermit will attempt create one, thus preventing anyone who subsequently tries to run Kermit, UUCP, tip, or similar programs on the same line from gaining access until you release the line. If Kermit could not create the lock file (for instance because the uucp directory is write-protected), then you will receive a warning message but will be allowed to proceed at your -- and everyone else's -- risk. When Kermit terminates normally, your lock file is removed.

Even when the lock directory is writable and readable, the locking mechanism depends upon all users using the same name for the same device. If a device has more than one path associated with it, then a lock can be circumvented by using an alias.

When a lock-creating program abruptly terminates, e.g. because it crashes or is killed via shell command, the lock file remains in the uucp directory, spuriously indicating that the line is in use. If the lock file is owned by yourself, you may remove it. Otherwise, you'll have to get the owner or the system manager to remove it, or else wait for a system task to do so; uucp supports a function (uuclean) which removes these files after a predetermined age -- uucp sites tend to run this function periodically via crontab.

Locking is not needed, or used, if communications occur over the user's login terminal line (normally `/dev/tty`).

It may be seen that line locking is fraught with peril. It is included in Unix Kermit only because other Unix communication programs rely on it. While it is naturally desirable to assure exclusive access to a line, it is also undesirable to refuse access to a vacant line only because of a spurious lock file, or because the uucp directory is not appropriately protected.

1.6. C-Kermit under Berkeley or System III/V Unix:

C-Kermit may be interrupted at command level or during file transfer by typing Control-C. The program will perform its normal exit function, restoring the terminal and releasing any lock. If a protocol transaction was in progress, an error packet will be sent to the opposite Kermit so that it can terminate cleanly.

C-Kermit may be invoked in the background ("&" on shell commmand line). If a background process is "killed", the user will have to manually remove any lock file and may need to restore the modem. This is because the kill signal (`kill(`*x*`,9)`) cannot be trapped by Kermit.

During execution of a system command ('directory', 'cwd', or '`!`'), C-Kermit can often be returned to command level by typing a single Control-C. (With System III/V, the usual interrupt function (often the DEL key) is replaced by Control-C.)

Under Berkeley Unix only: C-Kermit may also be interrupted by `^Z` to put the process in the background.

Control-C, Control-Z, and Control-\ lose their normal functions during terminal connection and also during file transfer when the controlling tty line is being used for packet i/o.

If you are running C-Kermit in "quiet mode" in the foreground, then interrupting the program with a console interrupt like Control-C will not restore the terminal to normal conversational operation. This is because the system call to enable console interrupt traps will cause the program to block if it's running in the background, and the primary reason for quiet mode is to allow the program to run in the background without blocking, so that you can do other work in the foreground.

If C-Kermit is run in the background ("&" on shell commmand line), then the interrupt signal (Control-C) (and System III/V quit signal) are ignored. This prevents an interrupt signal intended for a foreground job (say a compilation) from being trapped by a background Kermit session.

1.7. C-Kermit on the DEC Pro-3xx with Pro/Venix Version 1

The DEC Professional 300 series are PDP-11/23 based personal computers. Venix Version 1 is a Unix v7 derivative. It should not be confused with Venix Version 2, which is based on ATT System V; these comments apply to Venix Version 1 only. C-Kermit runs in local mode on the Pro-3*xx* when invoked from the console; the default device is `/dev/com1.dout`. When connected to a remote system (using C-Kermit's 'connect' command), Pro/Venix itself (not Kermit) provides VT52 terminal emulation. Terminal operation at high speeds (like 9600 baud) requires xon/xoff flow control, which unfortunately interferes with applications such as the EMACS that use Control-Q and Control-S as commands.

When logging in to a Pro-3xx (or any workstation) through the "back port", it may be necessary to give the command "set line /dev/tty" in order to get C-Kermit to function correctly in remote mode (on a system in which it normally expects to be operating in local mode).

1.8. C-Kermit under VAX/VMS

C-Kermit can be built using VAX-11 C to run under VMS. Most of the descriptions in this manual hold true, but it should be noted that as of this writing the VMS support is not thoroughly tested, and no explicit support exists for the various types of VMS files and their attributes.

The C-Kermit init file for VMS is called `KERMIT.INI`.

1.9. C-Kermit on the Macintosh and other Systems

The "protocol kernel" of C-Kermit is also used by Columbia's Macintosh Kermit. The user and system interface is entirely different, and is covered in a separate document.

There is also a Kermit for the Commodore Amiga based on C-Kermit, as well as versions for MS-DOS, Data General operating systems, etc.

1.10. C-Kermit Restrictions and Known Bugs

1. Editing characters: The program's interactive command interrupt, delete, and kill characters are Control-C, Delete (or Backspace), and Control-U, respectively. There is currently no way to change them to suit your taste or match those used by your shell, in case those are different.

2. Flow control: C-Kermit attempts to use XON/XOFF flow control during protocol operations, but it also puts the communication line into "rawmode". On many systems, rawmode disables flow control, so even though you may have "set flow xon/xoff", no flow control will be done. This is highly system and Unix-version dependent.

3. Modem controls: If a connection is made over a communication line (rather than on the controlling

terminal line), and that line has modem controls, (e.g. data terminal ready and carrier detection implementation), returning to the shell level will disconnect the conversation. In that case, one should use interactive mode commands, and avoid use of piped shell-level operation (also see ’set modem-dialer’ and ’dial’ commands.)

4. Login Scripts: The present login scripts implementation follows the Unix conventions of uucp’s "`L.sys`" file, rather than the normal Kermit "INPUT/OUTPUT" style.

5. Dial-out vs dial-in communications lines: C-Kermit requires a dial-out or dedicated line for the "set line" or "-l" options. Most systems have some lines dedicated to dial-in, which they enable "loggers" on, and some lines available for dial-out. Recent releases of Unix (ATT & Berkeley) have mechanisms for changing the directionality of a line.

6. Using C-Kermit on Local Area Networks: C-Kermit can successfully operate at speeds up to 19200 baud over LANs, provided the network buffers are big enough to accommodate Kermit packets.

   When computers are connected to LANs through asynchronous terminal interfaces, then the connection should be configured to do XON/XOFF flow control between the network interface and the computer, rather than passing these signals through transparently. This can help prevent Kermit from overrunning the LAN’s buffers if they are small (or if the LAN is congested), and will can also prevent the LAN from overrunning a slow Kermit’s buffers.

   If the network hardware cannot accept 100 characters at a time, and flow control cannot be done between the network and the computer, then Kermit’s "set send/receive packet-length" command can be used to shorten the packets.

7. Resetting terminal after abnormal termination or kill: When C-Kermit terminates abnormally (say, for example, by a kill command issued by the operator) the user may need to reset the terminal state. If commands do not seem to be accepted at the shell prompt, try Control-J "stty sane" Control-J (use "reset" on Berkeley Unix). That should take the terminal out of "raw mode" if it was stuck there.

8. Remote host commands may time-out on lengthy activity: Using "remote host" to instruct the C-Kermit server to invoke Unix functions (like "make") that might take a long time to produce output can cause timeout conditions.

9. XOFF deadlocks: When connecting back to C-Kermit after a transaction, or after finishing the server, it may be necessary to type a Control-Q to clear up an XOFF deadlock. There’s not much the program can do about this...

1.11. How to Build C-Kermit for a Unix System

The C-Kermit files, as distributed from Columbia, all begin with the prefix "ck". You should make a directory for these files and then cd to it. A makefile is provided to build C-Kermit for various Unix systems (there are separate makefiles for VMS and the Macintosh). As distributed, the makefile has the name "`ckuker.mak`". You should rename it to "`makefile`" and then type "make xxx", where xxx is the symbol for your system, for instance "make bsd" to make C-Kermit for 4.x BSD Unix. The result will be a program called "wermit". You should test this to make sure it works; if it does, then you can rename it to "kermit" and install it for general use. See the makefile for a list of the systems supported and the corresponding "make" arguments.

1.12. Adapting C-Kermit to Other Systems

C-Kermit is designed for portability. The level of portability is indicated in parentheses after the module name: "C" means any system that has a C compiler that conforms to the description in "The C Programming Language" by Kernighan & Ritchie (Prentice-Hall, 1978). "Cf" is like "C", but also requires "standard" features like printf and fprintf, argument passing via argv/argc, and so on, as described in Kernighan & Ritchie. "Unix" means the module should be useful under any Unix implementation; it requires features such as fork() and pipes. Anything else means that the module is particular to the indicated system. C-Kermit file names are of the form:

`ck`<*system*><*what*>`.`<*type*>

where the part before the dot is no more than 6 characters long, the part after the dot no more than 3 characters long, and:

<*type*> is the file type:

c:	C language source
h:	Header file for C language source
w:	Wart preprocessor source, converted by Wart (or Lex) to a C program
nr:	Nroff/Troff text formatter source
mss:	Scribe text formatter source
doc:	Documentation
ps:	Documentation, Postscript format
hlp:	Help text
bld:	Instructions for building the program
bwr:	A "beware" file - list of known bugs
upd:	Program update log
mak:	Makefile

<*system*> is a single character to tell what system the file applies to:

9:	OS-9
a:	Descriptive material, documentation
c:	All systems with C compilers
d:	Data General
h:	Harris computers (reserved)
i:	Commodore Amiga (Intuition)
m:	Macintosh
o:	OS/2
p:	IBM PC, PC-DOS (reserved)
u:	Unix and Unix-like systems
v:	VAX/VMS
w:	Wart

<*what*> is mnemonic (up to 3 characters) for what’s in the file:

aaa:	A "read-me" file, like this one
cmd:	Command parsing
con:	Connect command
deb:	Debug/Transaction Log formats, Typedefs
dia:	Modem/Dialer control
fio:	System-depdendent File I/O
fns:	Protocol support functions
fn2:	More protocol support functions
ker:	General C-Kermit definitions, information, documentation
mai:	Main program
pro:	Protocol
scr:	Script command

tio:	System-dependent terminal i/o & control and interrupt handing
usr:	User interface
us2:	More user interface
us3:	Still more user interface

Examples:

`ckufio.c`	File i/o for Unix
`ckmtio.c`	Terminal i/o for Macintosh
`ckuker.mss`	Scribe source for for Kermit User Guide chapter
`ckuker.nr`	Nroff source file for Unix C-Kermit man page

The following material discusses each of the C-Kermit modules briefly.

`ckcmai.c, ckcker.h, ckcdeb.h (Cf)`:
This is the main program. It contains declarations for global variables and a small amount of code to initialize some variables and invoke the command parser. In its distributed form, it assumes that command line arguments are passed to it via argc and argv. Since this portion of code is only several lines long, it should be easy to replace for systems that have different styles of user interaction. The header files define symbols and macros used by the various modules of C-Kermit. `ckcdeb.h` is the only header file that is included by all the C-Kermit modules, so it contains not only the debug format definitions, but also any compiler-dependent typedefs.

`ckwart.c (Cf), ckcpro.w (C)`:
The ckcpro module embodies the Kermit protocol state table and the code to accomplish state switching. It is written in "wart", a language which may be regarded as a subset of the Unix "lex" lexical analyzer generator. Wart implements enough of lex to allow the ckprot module to function. Lex itself was not used because it is proprietary. The protocol module `ckcpro.w` is read by wart, and a system-independent C program is produced. The syntax of a Wart program is illustrated by `ckcpro.w`, and is described in `ckwart.doc`.

`ckcfns.c (C)`:
The module contains all the Kermit protocol support functions -- packet formation, encoding, decoding, block check calculation, filename and data conversion, protocol parameter negotiation, and high-level interaction with the communication line and file system. To accommodate small systems, this module has been split into two -- `ckcfns.c` and `ckcfn2.c`.

`ckutio.c`:
This module contains the system-dependent primitives for communication line i/o, timers, and interrupts for the various versions of Unix. Certain important variables are defined in this module, which determine whether C-Kermit is by default remote or local, what the default communication device is, and so forth. The tio module maintains its own private database of file descriptors and modes for the console terminal and the file transfer communication line so that other modules (like ckcfns or the terminal connect module) need not be concerned with them. The variations among Unix implementations with respect to terminal control and timers are accommodated via conditional compilation.

`ckufio.c`:
This module contains system-dependent primitives for file i/o, wildcard (meta character) expansion, file existence and access checking, and system command execution for the various versions of Unix. It maintains an internal database of i/o "channels" (file pointers in this case) for the files C-Kermit cares about -- the input file (the file which is being sent), the output file (the file being received), the various logs, the screen, and so forth. This module varies little among Unix implementations except for the wildcard expansion code; the directory structure of 4.2bsd Unix is different from that of other Unix systems. Again, variation among Unix systems is selected using conditional compilation.

`ckuusr.h, ckuusr.c, ckuus2.c, ckuus3.c` (Unix):
This is the "user interface" for C-Kermit. It includes the command parser, the screen output functions, and console input functions. The command parser comes in two pieces -- the traditional Unix command line decoder (which is quite small and compact), and the interactive keyword parser (which is rather large). This module is fully replacable; its interface to the other modules is very simple, and is explained at the beginning of the source file. The ckuusr module also includes code to execute any commands directly which don't require the Kermit protocol -- local file management, etc. The module is rated "Unix" because it makes occasional use of the `system()` function.

Note that while `ckuusr` is logically one module, it has been split up into three C source files, plus a header file for the symbols they share in common. This is to accommodate small systems that cannot handle big modules. `ckuusr.c` has the command line and top-level interactive command parser; `ckuus2.c` has the help command and strings; `ckuus3` has the set and remote commands along with the logging, screen, and "interrupt" functions.

`ckucmd.c, ckucmd.h` (Cf):
This is an interactive command parsing package developed for C-Kermit. It is written portably enough to be usable on any system that has a C compiler that supports functions like printf. The file name parsing functions depend upon primitives defined in the fio module; if these primitives cannot be supplied for a certain system, then the filename parsing functions can be deleted, and the package will still be useful for parsing keywords, numbers, arbitrary text strings, and so forth. The style of interaction is the same as that found on the DECSYSTEM-20.

`ckucon.c` (Unix):
This is the connect module. As supplied, it should operate in any Unix environment, or any C-based environment that provides the fork() function. The module requires access to global variables that specify line speed, parity, duplex, flow control, etc, and invokes functions from the tio module to accomplish the desired settings and input/output, and functions from the fio module to perform session logging. No terminal emulation is performed, but since standard i/o is used for the console, this may be piped through a terminal emulation filter. The ckucon function may be entirely replaced, so long as the global settings are honored by its replacement. PC implementations of C-Kermit may require the ck?con module to do screen control, escape sequence interpretation, etc, and may also wish to write special code to get the best possible performance.

`ckudia.c` (Unix):
This is the dialer module. As supplied, it handles Hayes, Ventel, Penril, Racal-Vadic, and several other modems.

`ckuscr.c` (Unix):
This is the login script module. As supplied, it handles uucp-style scripts.

Moving C-Kermit to a new system entails:

1. Creating a new `ck?tio` module in C, assembler, or whatever language is most appropriate for system programming on the new system. If the system is Unix-like, then support may be added within the `ckutio.c` module itself using conditional compilation.

2. Creating a new `ck?fio` module, as above.

3. If the system is not Unix-like, then a new `ckuusr` module may be required, as well as a different invocation of it from `ckcmai`.

4. If the distributed connect module doesn't work or performs poorly, then it may be replaced. For instance, interrupt-driven i/o may be required, especially if the system doesn't have forks.

Those who favor a different style of user/program interaction from that provided in `ckuusr.c` may replace the entire module, for instance with one that provides a mouse/window/icon environment, a menu/function-key environment, etc.

A few guidelines should be followed to maintain portability:

- Keep variable and function names to 6 characters or less. Don't use identifiers that are distinguished from one another only by alphabetic case.

- Keep modules small. For instance, on a PDP-11 it is necessary to keep the code segment of each module below 8K in order to allow the segment mapping to occur which is necessary to run programs larger than 64K on a non-I-and-D-space machine.

- Keep strings short; many compilers have restrictive maximum lengths; 128 is the smallest maximum string constant length we've encountered so far.

- Keep (f,s)printf formats short. If these exceed some compiler dependent maximum (say, 128) memory will be overwritten and the program will probably core dump.

- Do not introduce system dependencies into `ckcpro.w` or `ckcfn*.c`.
- If a variable is a character, declare as CHAR, not int, to prevent the various sign extension and byte swapping foulups that occur when characters are placed in integer variables.
- Remember that different systems may use different length words for different things. Don't assume an integer can be used as a pointer, etc.
- Don't declare static functions; these can wreak havoc with systems that do segment mapping.
- In conditional compilations expressions, use `#ifdef` and `#ifndef` and not `#if`, which is not supported by some compilers. Also, don't use any operators in these expressions; many compilers will fail to understand expressions like `#ifdef FOO | BAR`. Also, don't put trailing tokens on `#else`'s or `#endif`'s (use `/*` comments `*/`).
- Don't define multiline macros.

In general, remember that this program will have to be compilable by old compilers and runnable on small systems.

Index

Table of Contents