@

(XEmacs Terminal Emulator Mode)

This is some notes about the term Emacs mode.

1 XEmacs Terminal Emulator Mode

The `term` package includes the major modes `term`, `shell`, and `gud` (for running gbd or another debugger). It is a replacement for the comint mode of Emacs 19, as well as shell, gdb, terminal, and telnet modes. The package works best with recent releases of Emacs 19, but will also work reasonably well with Emacs 18 as well as Lucid Emacs 19.

The file `nshell.el` is a wrapper to use unless term mode is built into Emacs. If works around some of the missing in older Emacs versions. To use it, edit the paths in `nshell.el`, appropriately, and then `M-x load-file nshell.el RET`. This will also load in replacement shell and gud modes.

1.1 Overview

The `term` mode is used to control a program (an "inferior process"). It sends most keyboard input characters to the program, and displays output from the program in the buffer. This is similar to the traditional comint mode, and modes derived from it (such as shell and gdb modes). You can do with the new term-based shell the same sort of things you could do with the old shell mode, using more or less the same interface. However, the new mode is more flexible, and works somewhat differently.

1.1.1 Output from the inferior

In typical usage, output from the inferior is added to the end of the buffer. If needed, the window will be scrolled, just like a regular terminal. (Only one line at a time will be scrolled, just like regular terminals, and in contrast to the old shell mode.) Thus the buffer becomes a log of your interaction with the inferior, just like the old shell mode.

Like a real terminal, term maintains a "cursor position." This is the `process-mark` of the inferior process. If the process-mark is not at the end of the buffer, output from the inferior will overwrite existing text in the buffer. This is like a real terminal, but unlike the old shell mode (which inserts the output, instead of overwriting).

Some programs (such as Emacs itself) need to control the appearance on the screen in detail. They do this by sending special control codes. The exact control codes needed from terminal to terminal, but nowadays most terminals and terminal emulators (including xterm) understand the so-called "ANSI escape sequences" (first popularized by the Digital's VT100 family of terminal). The term mode also understands these escape sequences, and for each control code does the appropriate thing to change the buffer so that the appearance of the window will match what it would be on a real terminal. (In contrast, the old shell mode doesn't handle terminal control codes at all.)

See <...> for the specific control codes.

1.1.2 The sub-bu er

A program that talks to terminal expects the terminal to have a fixed size. If the program is talking a terminal emulator program such as `xterm`, that size can be changed (if the xterm window is re-sized), but programs still assume a logical terminal that has a fixed size independent of the amount of output transmitted by the programs.

To programs that use it, the Emacs terminal emulator acts as if it too has a fixed size. The sub-bu er is the part of a `term`-mode buffer that corresponds to a "normal" terminal. Most of the time (unless you explicitly scroll the window displaying the buffer), the sub-buffer is the part of the buffer that is displayed in a window.

The sub-buffer is defined in terms of three buffer-local-variable:

term-height Variable

The height of the sub-buffer, in screen lines.

term-width Variable

The width of the sub-buffer, in screen columns.

term-home-marker Variable

The "home" position, that is the top left corner of the sub-buffer.

The sub-buffer is assumed to be the end part of the buffer; the `term-home-marker` should never be more than `term-height` screen lines from the end of the buffer.

1.1.3 The alternate sub-bu er

When a "graphical" program finishes, it is nice to restore the screen state to what it was before the program started. Many people are used to this behavior from `xterm`, and its also offered by the `term` emulator.

term-switch-to-alternate-sub-bu er set Function

If set is true, and we're not already using the alternate sub-buffer, switch to it. What this means is that the `term-home-marker` is saved (in the variable `term-saved-home-marker`), and the `term-home-marker` is set to the end of the buffer.

If set is false and we're using the alternate sub-buffer, switch back to the saved sub-buffer. What this means is that the (current, alternate) sub-buffer is deleted (using `(delete-region term-home-marker (point-max))`), and then the `term-home-marker` is restored (from `term-saved-home-marker`).

1.1.4 Input to the inferior

Characters typed by the user are sent to the inferior. How this is done depends on whether the `term` buffer is in "character" mode or "line" mode. (A `term` buffer can also be in "pager" mode. This is discussed <later>.) Which of these is currently active is specified in the mode line. The difference between them is the key-bindings available.

In character mode, one character (by default C-c) is special, and is a prefix for various commands. All other characters are sent directly to the inferior process, with no interpretation by Emacs. Character mode looks and feels like a real terminal, or a conventional terminal emulator such as xterm.

In line mode, key commands mostly have standard Emacs actions. Regulars characters insert themselves into the buffer. When return is typed, the entire current line of the buffer (except possibly the prompt) is sent to the inferior process. Line mode is basically the original shell mode from earlier Emacs versions.

To switch from line mode to character mode type C-c C-k. To switch from character mode to line mode type C-c C-j .

In either mode, "echoing" of user input is handled by the inferior. Therefor, in line mode after an input line at the end of the buffer is sent to the inferior, it is deleted from the buffer. This is so that the inferior can echo the input, if it wishes (which it normally does).

1.2 Connecting to remote computers

If you want to login to a remove computer, you can do that just as you would expect, using whatever commands you would normally use.

(This is worth emphasizing, because earlier versions of `shell` mode would not work properly if you tried to log in to some other computer, because of the way echoing was handled. That is why there was a separate `telnet` mode to partially compensate for these problems. The `telnet` mode is no longer needed, and is basically obsolete.)

A program that asks you for a password will normally suppress echoing of the password, so the password will not show up in the buffer. This will happen just as if you were using a real terminal, if the buffer is in char mode. If it is in line mode, the password will be temporarily visible, but will be erased when you hit return. (This happens automatically; there is no special password processing.)

When you log in to a different machine, you need to specify the type of terminal your using. If you are talking to a Bourne-compatible shell, and your system understands the `TERMCAP` variable, you can use the command M-x shell-send-termcap , which sends a string specifying the terminal type and size. (This command is also useful after the window has changed size.)

If you need to specify the terminal type manually, you can try the terminal types "ansi" or "vt100".

You can of course run gdb on that remote computer. One useful trick: If you invoke gdb with the `--fullname` option, it will send special commands to Emacs that will cause Emacs to pop up the source files you're debugging. This will work whether or not gdb is running on a different computer than Emacs, assuming can access the source files specified by gdb.

1.3 Paging

When the pager is enabled, Emacs will "pause" after each screenful of output (since the last input sent to the inferior). It will enter "pager" mode, which feels a lot like the "more" program: Typing a space requests another screenful of output. Other commands request more or less output, or scroll backwards in the `term` buffer. In pager mode, type h or ? to display a help message listing all the available pager mode commands.

In either character or line mode, type C-c p to enable paging, and C-c D to disable it.

1.4 Terminal Escape sequences

A program that does "graphics" on a terminal controls the terminal by sending strings called terminal escape sequences that the terminal (or terminal emulator) interprets as special commands. The `term` mode includes a terminal emulator that understands standard ANSI escape sequences, originally popularized by VT100 terminals, and now used by the `xterm` program and most modern terminal emulator software.

printing chars

tab

LF

1.4.1 Escape sequences to move the cursor

RETURN Moves to the beginning of the current screen line.

C-b Moves backwards one column. (Tabs are broken up if needed.)

Esc [R ; C H
Move to screen row R, screen column C, where (R=1) is the top row, and (C=1) is the leftmost column. Defaults are R=1 and C=1.

Esc [N A Move N (default 1) screen lines up.

Esc [N B Move N (default 1) screen lines down.

Esc [N C Move N (default 1) columns right.

Esc [N D Move N (default 1) columns left.

1.4.2 Escape commands for erasing text

These commands "erase" part of the sub-buffer. Erasing means replacing by white space; it is not the same as deleting. The relative screen positions of things that are not erased remain unchanged with each other, as does the relative cursor position.

E [J Erase from cursor to end of screen.

E [0 J Same as E [J.

E [1 J Erase from home position to point.

E [2 J Erase whole sub-buffer.

E [K Erase from point to end of screen line.

E [0 K Same as E [K.

E [1 K Erase from beginning of screen line to point.

E [2 K Erase whole screen line.

1.4.3 Escape sequences to insert and delete text

Esc [N L Insert N (default 1) blank lines.

Esc [N M Delete N (default 1) lines.

Esc [N P Delete N (default 1) characters.

Esc [N @ Insert N (default 1) spaces.

1.4.4 Escape sequences to scroll part of the visible window

Esc D Scroll forward one screen line.

Esc M Scroll backwards one screen line.

Esc [T ; B r
Set the scrolling region to be from lines T down to line B inclusive, where line 1 is the topmost line.

1.4.5 Command hook

If C-z is seen, any text up to a following LF is scanned. The text in between (not counting the initial C-z or the final LF) is passed to the function that is the value of `term-command-hook`.

The default value of the `term-command-hook` variable is the function `term-command-hook`, which handles the following:

C-z C-z FILENAME:LINENUMBER:IGNORED LF
Set term-pending-frame to `(cons "FILENAME" LINENUMBER)`. When the buffer is displayed in the current window, show the FILENAME in the other window, and show an arrow at LINENUMBER. Gdb emits these strings when invoked with the flag –fullname. This is used by gdb mode; you can also invoke gdb with this flag from shell mode.

C-z / DIRNAME LF
Set the directory of the term buffer to DIRNAME

C-z ! LEXPR LF
Read and evaluate LEXPR as a Lisp expression. The result is ignored.

1.4.6 Miscellaneous escapes

C-g (Bell)
Calls `(beep t)`.

Esc 7 Save cursor.

Esc 8 Restore cursor.

Esc [47 h
Switch to the alternate sub-buffer,

Esc [47 l
Switch back to the regular sub-buffer,