NAME

termcap - terminal capability data base

SYNOPSIS

termcap

DESCRIPTION

The **termcap** file is a data base describing terminals, used, for example, by vi(1) and ncurses(3). Terminals are described in **termcap** by giving a set of capabilities that they have and by describing how operations are performed. Padding requirements and initialization sequences are included in **termcap**.

Entries in **termcap** consist of a number of ‘:’-separated fields. The first entry for each terminal gives the names that are known for the terminal, separated by ‘|’ characters. The first name given is the most common abbreviation for the terminal. The last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should be in lower case and contain no blanks; the last name may well contain upper case characters and blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, thus "hp2621" This name should not contain hyphens. Modes that the hardware can be in or user preferences should be indicated by appending a hyphen and an indicator of the mode. Therefore, a "vt100" in 132-column mode would be "vt100-w". The following suffixes should be used where possible:

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	vt100-w
-am	With automatic margins (usually default)	vt100-am
-nam	Without automatic margins	vt100-nam
-*n*	Number of lines on screen	aaa-60
-na	No arrow keys (leave them in local)	concept100-na
-*np*	Number of pages of memory	concept100-4p
-rv	Reverse video	concept100-rv

CAPABILITIES

The description field attempts to convey the semantics of the capability. You may find some codes in the description field:

(P) indicates that padding may be specified.

#[1-9] in the description field indicates that the string is passed through tparm(3) or tgoto(3) with parms as given (#*i*).

(P*) indicates that padding may vary in proportion to the number of lines affected.

(#*i*) indicates the *i*th parameter.

These are the boolean capabilities:

Boolean	**TCap**	**Description**
Variables	**Code**	
auto_left_margin	bw	cursor_left wraps from column 0 to last column
auto_right_margin	am	terminal has automatic margins
no_esc_ctlc	xb	beehive (f1=escape, f2=ctrl C)
ceol_standout_glitch	xs	standout not erased by overwriting (hp)
eat_newline_glitch	xn	newline ignored after 80 cols (concept)
erase_overstrike	eo	can erase overstrikes with a blank
generic_type	gn	generic line type
hard_copy	hc	hardcopy terminal
has_meta_key	km	Has a meta key, sets msb high
has_status_line	hs	has extra status line
insert_null_glitch	in	insert mode distinguishes nulls
memory_above	da	display may be retained above the screen
memory_below	db	display may be retained below the screen
move_insert_mode	mi	safe to move while in insert mode
move_standout_mode	ms	safe to move while in standout mode
over_strike	os	terminal can overstrike
status_line_esc_ok	es	escape can be used on the status line
dest_tabs_magic_smso	xt	tabs destructive, magic so char (t1061)
tilde_glitch	hz	cannot print ~'s (hazeltine)
transparent_underline	ul	underline character overstrikes
xon_xoff	xo	terminal uses xon/xoff handshaking
needs_xon_xoff	nx	padding will not work, xon/xoff required
prtr_silent	5i	printer will not echo on screen
hard_cursor	HC	cursor is hard to see
non_rev_rmcup	NR	enter_ca_mode does not reverse exit_ca_mode
no_pad_char	NP	pad character does not exist
non_dest_scroll_region	ND	scrolling region is non-destructive
can_change	cc	terminal can re-define existing colors
back_color_erase	ut	screen erased with background color

hue_lightness_saturation	hl	terminal uses only HLS color notation (tektronix)
col_addr_glitch	YA	only positive motion for column address and micro_column_address caps
cr_cancels_micro_mode	YB	using cr turns off micro mode
has_print_wheel	YC	printer needs operator to change character set
row_addr_glitch	YD	only positive motion for row_address and micro_row_address caps
semi_auto_right_margin	YE	printing in last column causes cr
cpi_changes_res	YF	changing character pitch changes resolution
lpi_changes_res	YG	changing line pitch changes resolution

These are the numeric capabilities:

Numeric	**TCap**	**Description**
Variables	**Code**	
columns	co	number of columns in aline
init_tabs	it	tabs initially every # spaces
lines	li	number of lines on screen or page
lines_of_memory	lm	lines of memory if > line. 0 => varies
magic_cookie_glitch	sg	number of blank chars left by enter_standout_mode or exit_standout_mode
padding_baud_rate	pb	lowest baud rate where padding needed
virtual_terminal	vt	virtual terminal number (CB/unix)
width_status_line	ws	columns in status line
num_labels	Nl	number of labels on screen
label_height	lh	rows in each label
label_width	lw	columns in each label
max_attributes	ma	maximum combined attributes terminal can handle
maximum_windows	MW	maximum number of definable windows
magic_cookie_glitch_ul	ug	number of blanks left by underline
#		
# These came in with SVr4's color support		
#		
max_colors	Co	maximum numbers of colors on screen
max_pairs	pa	maximum number of color-pairs on the screen
no_color_video	NC	video attributes that cannot be used with colors

```
#
# The following numeric capabilities are present in the SVr4.0 term
# structure, but are not yet documented in the man page.
# They came in with SVr4's printer support.
#
```

buffer_capacity	Ya	numbers of bytes buffered before printing
dot_vert_spacing	Yb	spacing of pins vertically in pins per inch
dot_horz_spacing	Yc	spacing of dots horizontally in dots per inch
max_micro_address	Yd	maximum value in micro_..._address
max_micro_jump	Ye	maximum value in parm_..._micro
micro_char_size	Yf	character size when in micro mode
micro_line_size	Yg	line size when in micro mode
number_of_pins	Yh	numbers of pins in print-head
output_res_char	Yi	horizontal resolution in units per line
output_res_line	Yj	vertical resolution in units per line
output_res_horz_inch	Yk	horizontal resolution in units per inch
output_res_vert_inch	Yl	vertical resolution in units per inch
print_rate	Ym	print rate in chars per second
wide_char_size	Yn	character step size when in double wide mode
buttons	BT	number of buttons on mouse
bit_image_entwining	Yo	number of passed for each bit-image row
bit_image_type	Yp	type of bit-image device

These are the string capabilities:

String Variables	**TCap Code**	**Description**
back_tab	bt	back tab (P)
bell	bl	audible signal (bell) (P)
carriage_return	cr	carriage return (P*)
change_scroll_region	cs	change region to line #1 to line #2 (P)
clear_all_tabs	ct	clear all tab stops (P)
clear_screen	cl	clear screen and home cursor (P*)
clr_eol	ce	clear to end of line (P)
clr_eos	cd	clear to end of screen (P*)
column_address	ch	horizontal position #1, absolute (P)
command_character	CC	terminal settable cmd character in prototype

cursor_address	cm	move to row #1 columns #2
cursor_down	do	down one line
cursor_home	ho	home cursor
cursor_invisible	vi	make cursor invisible
cursor_left	le	move left one space
cursor_mem_address	CM	memory relative cursor addressing
cursor_normal	ve	make cursor appear normal (undo cursor_invisible/cursor_visible)
cursor_right	nd	move right one space
cursor_to_ll	ll	last line, first column
cursor_up	up	up one line
cursor_visible	vs	make cursor very visible
delete_character	dc	delete character (P*)
delete_line	dl	delete line (P*)
dis_status_line	ds	disable status line
down_half_line	hd	half a line down
enter_alt_charset_mode	as	start alternate character set (P)
enter_blink_mode	mb	turn on blinking
enter_bold_mode	md	turn on bold (extra bright) mode
enter_ca_mode	ti	string to start programs using cursor_address
enter_delete_mode	dm	enter delete mode
enter_dim_mode	mh	turn on half-bright mode
enter_insert_mode	im	enter insert mode
enter_secure_mode	mk	turn on blank mode (characters invisible)
enter_protected_mode	mp	turn on protected mode
enter_reverse_mode	mr	turn on reverse video mode
enter_standout_mode	so	begin standout mode
enter_underline_mode	us	begin underline mode
erase_chars	ec	erase #1 characters (P)
exit_alt_charset_mode	ae	end alternate character set (P)
exit_attribute_mode	me	turn off all attributes
exit_ca_mode	te	strings to end programs using cup
exit_delete_mode	ed	end delete mode
exit_insert_mode	ei	exit insert mode
exit_standout_mode	se	exit standout mode
exit_underline_mode	ue	exit underline mode
flash_screen	vb	visible bell (may not move cursor)
form_feed	ff	hardcopy terminal page eject (P*)
from_status_line	fs	return from status line

init_1string		i1	initialization string
init_2string		is	initialization string
init_3string		i3	initialization string
init_file	if	name of	initialization file
insert_character	ic	insert character (P)	
insert_line		al	insert line (P*)
insert_padding		ip	insert padding after inserted character
key_backspace		kb	backspace key
key_catab		ka	clear-all-tabs key
key_clear	kC	clear-screen or erase key	
key_ctab	kt	clear-tab key	
key_dc		kD	delete-character key
key_dl		kL	delete-line key
key_down		kd	down-arrow key
key_eic		kM	sent by rmir or smir in insert mode
key_eol		kE	clear-to-end-of-line key
key_eos		kS	clear-to-end-of-screen key
key_f0		k0	F0 function key
key_f1		k1	F1 function key
key_f10		k;	F10 function key
key_f2		k2	F2 function key
key_f3		k3	F3 function key
key_f4		k4	F4 function key
key_f5		k5	F5 function key
key_f6		k6	F6 function key
key_f7		k7	F7 function key
key_f8		k8	F8 function key
key_f9		k9	F9 function key
key_home		kh	home key
key_ic		kI	insert-character key
key_il		kA	insert-line key
key_left	kl	left-arrow key	
key_ll		kH	last-line key
key_npage		kN	next-page key
key_ppage		kP	prev-page key
key_right	kr	right-arrow key	
key_sf		kF	scroll-forward key
key_sr		kR	scroll-backward key
key_stab	kT	set-tab	key
key_up		ku	up-arrow key

keypad_local		ke	leave ’keyboard_transmit’ mode			
keypad_xmit		ks	enter ’keyboard_transmit’ mode			
lab_f0		l0	label on function key f0 if not			f0
lab_f1		l1	label on function key f1 if not			f1
lab_f10		la	label on function key f10 if not f10			
lab_f2		l2	label on function key f2 if not			f2
lab_f3		l3	label on function key f3 if not			f3
lab_f4		l4	label on function key f4 if not			f4
lab_f5		l5	label on function key f5 if not			f5
lab_f6		l6	label on function key f6 if not			f6
lab_f7		l7	label on function key f7 if not			f7
lab_f8		l8	label on function key f8 if not			f8
lab_f9		l9	label on function key f9 if not			f9
meta_off	mo	turn off meta mode				
meta_on		mm	turn on	meta mode (8th-bit on)		
newline		nw	newline	(behave	like cr	followed by lf)
pad_char	pc	padding	char (instead of null)			
parm_dch	DC	delete #1 chars		(P*)		
parm_delete_line	DL	delete #1 lines		(P*)		
parm_down_cursor	DO	down #1	lines (P*)			
parm_ich	IC	insert #1 chars		(P*)		
parm_index		SF	scroll forward #1 lines			(P)
parm_insert_line	AL	insert #1 lines		(P*)		
parm_left_cursor	LE	move #1	chars to the left (P)			
parm_right_cursor	RI	move #1	chars to the right (P*)			
parm_rindex		SR	scroll back #1 lines (P)			
parm_up_cursor		UP	up #1 lines (P*)			
pkey_key	pk	program	function key #1		to type	string #2
pkey_local		pl	program	function key #1		to execute
			string #2			
pkey_xmit		px	program	function key #1		to transmit
			string #2			
print_screen		ps	print contents of screen			
prtr_off	pf	turn off printer				
prtr_on		po	turn on	printer		
repeat_char		rp	repeat char #1 #2 times			(P*)
reset_1string		r1	reset string			
reset_2string		r2	reset string			
reset_3string		r3	reset string			
reset_file	rf	name of	reset file			

```
restore_cursor          rc      restore cursor to last position         of
                                save_cursor
row_address             cv      vertical position #1 absolute (P)
save_cursor             sc      save current cursor position (P)
scroll_forward          sf      scroll text up (P)
scroll_reverse          sr      scroll text down (P)
set_attributes          sa      define video attributes #1-#9 (PG9)
set_tab                 st      set a tab in every row, current columns
set_window              wi      current window is lines #1-#2 cols #3-#4
tab                     ta      tab to next 8-space hardware tab stop
to_status_line          ts      move to  status line
underline_char          uc      underline char and move past it
up_half_line            hu      half a line up
init_prog       iP      path name of program for initialization
key_a1                  K1      upper left of keypad
key_a3                  K3      upper right of keypad
key_b2                  K2      center of keypad
key_c1                  K4      lower left of keypad
key_c3                  K5      lower right of keypad
prtr_non        pO      turn on  printer  for #1 bytes
termcap_init2           i2      secondary initialization string
termcap_reset           rs      terminal reset string
#
# SVr1 capabilities stop here. IBM's version of terminfo is the same as
# SVr4 up to this point, but has a different set afterwards.
#
char_padding            rP      like insert_padding but when in  insert mode
acs_chars       ac      graphics charset pairs - def=vt100
plab_norm               pn      program label #1 to show string  #2
key_btab        kB      back-tab key
enter_xon_mode          SX      turn on  xon/xoff handshaking
exit_xon_mode           RX      turn off xon/xoff handshaking
enter_am_mode           SA      turn on  automatic margins
exit_am_mode            RA      turn off automatic margins
xon_character           XN      XON character
xoff_character          XF      XOFF character
ena_acs                 eA      enable alternate char set
label_on        LO      turn on  soft labels
label_off       LF      turn off soft labels
key_beg                 @1      begin key
```

key_cancel		@2	cancel key	
key_close	@3	close key		
key_command		@4	command	key
key_copy	@5	copy key		
key_create		@6	create key	
key_end		@7	end key	
key_enter	@8	enter/send key		
key_exit	@9	exit key		
key_find	@0	find key		
key_help	%1	help key		
key_mark	%2	mark key		
key_message		%3	message	key
key_move		%4	move key	
key_next	%5	next key		
key_open	%6	open key		
key_options		%7	options	key
key_previous		%8	previous key	
key_print	%9	print key		
key_redo	%0	redo key		
key_reference		&1	reference key	
key_refresh		&2	refresh	key
key_replace		&3	replace	key
key_restart		&4	restart	key
key_resume		&5	resume key	
key_save	&6	save key		
key_suspend		&7	suspend	key
key_undo	&8	undo key		
key_sbeg	&9	shifted	key	
key_scancel		&0	shifted	key
key_scommand		*1	shifted	key
key_scopy		*2	shifted	key
key_screate		*3	shifted	key
key_sdc		*4	shifted	key
key_sdl		*5	shifted	key
key_select		*6	select key	
key_send	*7	shifted	key	
key_seol	*8	shifted	key	
key_sexit	*9	shifted	key	
key_sfind	*0	shifted	key	
key_shelp		#1	shifted	key

key_shome	#2	shifted key
key_sic	#3	shifted key
key_sleft	#4	shifted key
key_smessage	%a	shifted key
key_smove	%b	shifted key
key_snext	%c	shifted key
key_soptions	%d	shifted key
key_sprevious	%e	shifted key
key_sprint	%f	shifted key
key_sredo	%g	shifted key
key_sreplace	%h	shifted key
key_sright	%i	shifted key
key_srsume	%j	shifted key
key_ssave	!1	shifted key
key_ssuspend	!2	shifted key
key_sundo	!3	shifted key
req_for_input	RF	send next input char (for ptys)
key_f11	F1	F11 function key
key_f12	F2	F12 function key
key_f13	F3	F13 function key
key_f14	F4	F14 function key
key_f15	F5	F15 function key
key_f16	F6	F16 function key
key_f17	F7	F17 function key
key_f18	F8	F18 function key
key_f19	F9	F19 function key
key_f20	FA	F20 function key
key_f21	FB	F21 function key
key_f22	FC	F22 function key
key_f23	FD	F23 function key
key_f24	FE	F24 function key
key_f25	FF	F25 function key
key_f26	FG	F26 function key
key_f27	FH	F27 function key
key_f28	FI	F28 function key
key_f29	FJ	F29 function key
key_f30	FK	F30 function key
key_f31	FL	F31 function key
key_f32	FM	F32 function key
key_f33	FN	F33 function key

key_f34	FO	F34 function key
key_f35	FP	F35 function key
key_f36	FQ	F36 function key
key_f37	FR	F37 function key
key_f38	FS	F38 function key
key_f39	FT	F39 function key
key_f40	FU	F40 function key
key_f41	FV	F41 function key
key_f42	FW	F42 function key
key_f43	FX	F43 function key
key_f44	FY	F44 function key
key_f45	FZ	F45 function key
key_f46	Fa	F46 function key
key_f47	Fb	F47 function key
key_f48	Fc	F48 function key
key_f49	Fd	F49 function key
key_f50	Fe	F50 function key
key_f51	Ff	F51 function key
key_f52	Fg	F52 function key
key_f53	Fh	F53 function key
key_f54	Fi	F54 function key
key_f55	Fj	F55 function key
key_f56	Fk	F56 function key
key_f57	Fl	F57 function key
key_f58	Fm	F58 function key
key_f59	Fn	F59 function key
key_f60	Fo	F60 function key
key_f61	Fp	F61 function key
key_f62	Fq	F62 function key
key_f63	Fr	F63 function key
clr_bol	cb	Clear to beginning of line
clear_margins	MC	clear right and left soft margins
set_left_margin	ML	set left soft margin
set_right_margin	MR	set right soft margin
label_format	Lf	label format
set_clock	SC	set clock, #1 hrs #2 mins #3 secs
display_clock	DK	display clock at (#1,#2)
remove_clock	RC	remove clock
create_window	CW	define a window #1 from #2, #3 to #4, #5
goto_window	WG	go to window #1

```
hangup                        HU        hang-up   phone
dial_phone                    DI        dial number #1
quick_dial                    QD        dial number #1 without checking
tone                          TO        select touch tone dialing
pulse                         PU        select pulse dialling
flash_hook                    fh        flash switch hook
fixed_pause                   PA        pause for 2-3 seconds
wait_tone           WA        wait for dial-tone
user0                         u0        User string #0
user1                         u1        User string #1
user2                         u2        User string #2
user3                         u3        User string #3
user4                         u4        User string #4
user5                         u5        User string #5
user6                         u6        User string #6
user7                         u7        User string #7
user8                         u8        User string #8
user9                         u9        User string #9
#
# SVr4 added these capabilities to support color
#
orig_pair           op        Set default pair to its       original value
orig_colors                   oc        Set all   color pairs to the original ones
initialize_color    Ic        initialize color #1 to (#2,#3,#4)
initialize_pair               Ip        Initialize color pair #1 to fg=(#2,#3,#4),
                                        bg=(#5,#6,#7)
set_color_pair                sp        Set current color pair to #1
set_foreground                Sf        Set foreground color #1
set_background                Sb        Set background color #1
#
# SVr4 added these capabilities to support printers
#
change_char_pitch   ZA        Change number of characters per         inch
change_line_pitch   ZB        Change number of lines per inch
change_res_horz               ZC        Change horizontal resolution
change_res_vert               ZD        Change vertical     resolution
define_char                   ZE        Define a character
enter_doublewide_mode         ZF        Enter double-wide mode
enter_draft_quality ZG        Enter draft-quality mode
enter_italics_mode  ZH        Enter italic mode
```

enter_leftward_mode	ZI	Start leftward carriage motion
enter_micro_mode	ZJ	Start micro-motion mode
enter_near_letter_quality	ZK	Enter NLQ mode
enter_normal_quality	ZL	Enter normal-quality mode
enter_shadow_mode	ZM	Enter shadow-print mode
enter_subscript_mode	ZN	Enter subscript mode
enter_superscript_mode	ZO	Enter superscript mode
enter_upward_mode	ZP	Start upward carriage motion
exit_doublewide_mode	ZQ	End double-wide mode
exit_italics_mode	ZR	End italic mode
exit_leftward_mode	ZS	End left-motion mode
exit_micro_mode	ZT	End micro-motion mode
exit_shadow_mode	ZU	End shadow-print mode
exit_subscript_mode	ZV	End subscript mode
exit_superscript_mode	ZW	End superscript mode
exit_upward_mode	ZX	End reverse character motion
micro_column_address	ZY	Like column_address in micro mode
micro_down	ZZ	Like cursor_down in micro mode
micro_left	Za	Like cursor_left in micro mode
micro_right	Zb	Like cursor_right in micro mode
micro_row_address	Zc	Like row_address in micro mode
micro_up	Zd	Like cursor_up in micro mode
order_of_pins	Ze	Match software bits to print-head pins
parm_down_micro	Zf	Like parm_down_cursor in micro mode
parm_left_micro	Zg	Like parm_left_cursor in micro mode
parm_right_micro	Zh	Like parm_right_cursor in micro mode
parm_up_micro	Zi	Like parm_up_cursor in micro mode
select_char_set	Zj	Select character set
set_bottom_margin	Zk	Set bottom margin at current line
set_bottom_margin_parm	Zl	Set bottom margin at line #1 or #2 lines from bottom
set_left_margin_parm	Zm	Set left (right) margin at column #1 (#2)
set_right_margin_parm	Zn	Set right margin at column #1
set_top_margin	Zo	Set top margin at current line
set_top_margin_parm	Zp	Set top (bottom) margin at row #1 (#2)
start_bit_image	Zq	Start printing bit image graphics
start_char_set_def	Zr	Start character set definition
stop_bit_image	Zs	Stop printing bit image graphics
stop_char_set_def	Zt	End definition of character aet
subscript_characters	Zu	List of subscriptible characters

```
superscript_characters      Zv     List of   superscriptible    characters
these_cause_cr              Zw     Printing any of    these chars causes CR
zero_motion                 Zx     No motion for subsequent character
#
# The following    string capabilities are        present   in the SVr4.0 term
# structure, but are not documented in the man page.
#
char_set_names              Zy     List of   character set names
key_mouse                   Km     Mouse event has   occurred
mouse_info                  Mi     Mouse status information
req_mouse_pos               RQ     Request   mouse position
get_mouse                   Gm     Curses should get button events
set_a_foreground   AF       Set ANSI foreground color
set_a_background   AB       Set ANSI background color
pkey_plab                   xl     Program   function key #1    to type    string #2
                                   and show string    #3
device_type                 dv     Indicate language/codeset support
code_set_init               ci     Init sequence for multiple codesets
set0_des_seq                s0     Shift to code set 0 (EUC set 0,          ASCII)
set1_des_seq                s1     Shift to code set 1
set2_des_seq                s2     Shift to code set 2
set3_des_seq                s3     Shift to code set 3
set_lr_margin               ML     Set both left and right        margins   to #1, #2
set_tb_margin               MT     Sets both top and bottom margins to #1,           #2
bit_image_repeat   Xy       Repeat bit image cell #1 #2 times
bit_image_newline  Zz       Move to   next row of the    bit image
bit_image_carriage_return   Yv     Move to   beginning of same row
color_names                 Yw     Give name for color #1
define_bit_image_region     Yx     Define rectangular bit image region
end_bit_image_region        Yy     End a bit-image    region
set_color_band              Yz     Change to ribbon color #1
set_page_length             YZ     Set page length    to #1 lines
#
# SVr4 added these capabilities for direct PC-clone support
#
display_pc_char             S1     Display   PC character
enter_pc_charset_mode       S2     Enter PC character display mode
exit_pc_charset_mode        S3     Exit PC   character display mode
enter_scancode_mode         S4     Enter PC scancode mode
exit_scancode_mode          S5     Exit PC   scancode mode
```

pc_term_options	S6	PC terminal options
scancode_escape	S7	Escape for scancode emulation
alt_scancode_esc	S8	Alternate escape for scancode emulation
#		
# The XSI Curses standard added		these.
#		
enter_horizontal_hl_mode	Xh	Enter horizontal highlight mode
enter_left_hl_mode	Xl	Enter left highlight mode
enter_low_hl_mode	Xo	Enter low highlight mode
enter_right_hl_mode	Xr	Enter right highlight mode
enter_top_hl_mode	Xt	Enter top highlight mode
enter_vertical_hl_mode	Xv	Enter vertical highlight mode

Obsolete termcap capabilities. New software should not rely on them at all.

Boolean Variables	**TCap Code**	**Description**
linefeed_is_newline	NL	move down with ^J
even_parity	EP	terminal requires even parity
odd_parity	OP	terminal requires odd parity
half_duplex	HD	terminal is half-duplex
lower_case_only	LC	terminal has only lower case
upper_case_only	UC	terminal has only upper case
has_hardware_tabs	pt	has 8-char tabs invoked with ^I
return_does_clr_eol	xr	return clears the line
tek_4025_insert_line	xx	Tektronix 4025 insert-line glitch
backspaces_with_bs	bs	uses ^H to move left
crt_no_scrolling	ns	crt cannot scroll
no_correctly_working_cr	nc	no way to go to start of line

Number Variables	**TCap Code**	**Description**
backspace_delay	dB	padding required for ^H
form_feed_delay	dF	padding required for ^L
horizontal_tab_delay	dT	padding required for ^I
vertical_tab_delay	dV	padding required for ^V
number_of_function_keys	kn	count of function keys
carriage_return_delay	dC	pad needed for CR
new_line_delay	dN	pad needed for LF

String	**TCap**	**Description**
Variables	**Code**	
other_non_function_keys	ko	list of self-mapped keycaps
arrow_key_map	ma	map arrow keys
memory_lock_above	ml	lock visible screen memory above the current line
memory_unlock	mu	unlock visible screen memory above the current line
linefeed_if_not_lf	nl	use to move down
backspace_if_not_bs	bc	move left, if not ^H

A Sample Entry

The following entry, which describes the Concept-100, is among the more complex entries in the **termcap** file as of this writing.

```
ca|concept100|c100|concept|c104|concept100-4p|HDS Concept-100:\
        :al=3*\E^R:am:bl=^G:cd=16*\E^C:ce=16\E^U:cl=2*^L:cm=\Ea%+ %+ :\
        :co#80:.cr=9^M:db:dc=16\E^A:dl=3*\E^B:do=^J:ei=\E\200:eo:im=\E^P:in:\
        :ip=16*:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200\Eo\47\E:k1=\E5:\
        :k2=\E6:k3=\E7:kb=^h:kd=\E<:ke=\Ex:kh=\E?:kl=\E>:kr=\E=:ks=\EX:\
        :ku=\E;:le=^H:li#24:mb=\EC:me=\EN\200:mh=\EE:mi:mk=\EH:mp=\EI:\
        :mr=\ED:nd=\E=:pb#9600:rp=0.2*\Er%.%+ :se=\Ed\Ee:sf=^J:so=\EE\ED:\
        :.ta=8\t:te=\Ev    \200\200\200\200\200\200\Ep\r\n:\
        :ti=\EU\Ev  8p\Ep\r:ue=\Eg:ul:up=\E;:us=\EG:\
        :vb=\Ek\200\200\200\200\200\200\200\200\200\200\200\200\200\200\EK:\
        :ve=\Ew:vs=\EW:vt#8:xn:\
        :bs:cr=^M:dC#9:dT#8:nl=^J:ta=^I:pt:
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and empty fields may be included for readability (here between the last field on a line and the first field on the next). Comments may be included on lines beginning with "#".

Types of Capabilities

Capabilities in **termcap** are of three types: Boolean capabilities, which indicate particular features that the terminal has; numeric capabilities, giving the size of the display or the size of other attributes; and string capabilities, which give character sequences that can be used to perform particular terminal operations. All capabilities have two-letter codes. For instance, the fact that the Concept has *automatic margins* (an automatic return and linefeed when the end of a line is reached) is indicated by the Boolean capability **am**. Hence the description of the Concept includes **am**.

Numeric capabilities are followed by the character ‘#’ then the value. In the example above **co**, which indicates the number of columns the display has, gives the value ‘80’ for the Concept.

Finally, string-valued capabilities, such as **ce** (clear-to-end-of-line sequence) are given by the two-letter code, an ‘=’, then a string ending at the next following ‘:’. A delay in milliseconds may appear after the ‘=’ in such a capability, which causes padding characters to be supplied by tputs(3) after the remainder of the string is sent to provide this delay. The delay can be either a number, such as ‘20’, or a number followed by an ‘*’, such as ‘3*’. An ‘*’ indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-line padding required. (In the case of insert-character, the factor is still the number of *lines* affected; this is always 1 unless the terminal has **in** and the software uses it.) When an ‘*’ is specified, it is sometimes useful to give a delay of the form ‘3.5’ to specify a delay per line to tenths of milliseconds. (Only one decimal place is allowed.)

A number of escape sequences are provided in the string-valued capabilities for easy encoding of control characters there. **\E** maps to an ESC character, **^X** maps to a control-X for any appropriate X, and the sequences **\n \r \t \b \f** map to linefeed, return, tab, backspace, and formfeed, respectively. Finally, characters may be given as three octal digits after a \, and the characters ^ and \ may be given as \^ and \\. If it is necessary to place a **:** in a capability it must be escaped as **\:** or be encoded as **\072**. If it is necessary to place a NUL character in a string capability it must be encoded as **\200**. (The routines that deal with **termcap** use C strings and strip the high bits of the output very late, so that a **\200** comes out as a **\000** would.)

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the first **cr** and **ta** in the example above.

Preparing Descriptions

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in **termcap** and to build up a description gradually, using partial descriptions with vi(1) to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the **termcap** file to describe it or bugs in vi(1). To easily test a new terminal description you are working on you can put it in your home directory in a file called *.termcap* and programs will look there before looking in */usr/share/misc/termcap*. You can also set the environment variable TERMPATH to a list of absolute file pathnames (separated by spaces or colons), one of which contains the description you are working on, and programs will search them in the order listed, and nowhere else. See termcap(3). The TERMCAP environment variable is usually set to the **termcap** entry itself to avoid reading files when starting up a program.

To get the padding for insert-line right (if the terminal manufacturer did not document it), a severe test is to use vi(1) to edit */etc/passwd* at 9600 baud, delete roughly 16 lines from the middle of the screen, then hit the ‘u’ key several times quickly. If the display messes up, more padding is usually needed. A similar test can be used for insert-character.

Basic Capabilities

The number of columns on each line of the display is given by the **co** numeric capability. If the display is a CRT, then the number of lines on the screen is given by the **li** capability. If the display wraps around to the beginning of the next line when the cursor reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, the code to do this is given by the **cl** string capability. If the terminal overstrikes (rather than clearing the position when a character is overwritten), it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as the Tektronix 4010 series, as well as to hard copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be carriage-return, ^**M**.) If there is a code to produce an audible signal (bell, beep, etc.), give this as **bl**.

If there is a code (such as backspace) to move the cursor one position to the left, that capability should be given as **le**. Similarly, codes to move to the right, up, and down should be given as **nd**, **up**, and **do**, respectively. These *local cursor motions* should not alter the text they pass over; for example, you would not normally use "nd= " unless the terminal has the **os** capability, because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in **termcap** have undefined behavior at the left and top edges of a CRT display. Programs should never attempt to backspace around the left edge, unless **bw** is given, and never attempt to go up off the top using local cursor motions.

In order to scroll text up, a program goes to the bottom left corner of the screen and sends the **sf** (index) string. To scroll text down, a program goes to the top left corner of the screen and sends the **sr** (reverse index) string. The strings **sf** and **sr** have undefined behavior when not on their respective corners of the screen. Parameterized versions of the scrolling sequences are **SF** and **SR**, which have the same semantics as **sf** and **sr** except that they take one parameter and scroll that many lines. They also have undefined behavior except at the appropriate corner of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output there, but this does not necessarily apply to **nd** from the last column. Leftward local motion is defined from the left edge only when **bw** is given; then an **le** from the left edge will move to the right edge of the previous row. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch-selectable automatic margins, the **termcap** description usually assumes that this feature is on, *i.e.*, **am**. If the terminal has a command that moves to the first column of the next line, that command can be given as **nw** (newline). It is permissible for this to clear the remainder of the current line, so if the terminal has no correctly-working CR and LF it may still be possible to craft a working **nw** out of one or both of them.

These capabilities suffice to describe hardcopy and "glass-tty" terminals. Thus the Teletype model 33 is described as

```
T3|tty33|33|tty|Teletype model 33:\
        :bl=^G:co#72:cr=^M:do=^J:hc:os:
```

and the Lear Siegler ADM-3 is described as

```
l3|adm3|3|LSI ADM-3:\
:am:bl=^G:cl=^Z:co#80:cr=^M:do=^J:le=^H:li#24:sf=^J:
```

Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with printf(3)-like escapes **%x** in it, while other characters are passed through unchanged. For example, to address the cursor the **cm** capability is given, using two parameters: the row and column to move to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory. If the terminal has memory-relative cursor addressing, that can be indicated by an analogous **CM** capability.)

The **%** encodings have the following meanings:

%%	output ‘%’
%d	output value as in printf(3) %d
%2	output value as in printf(3) %2d
%3	output value as in printf(3) %3d
%.	output value as in printf(3) %c
%+*x*	add *x* to value, then do %.
%>*xy*	if value > *x* then add *y*, no output
%r	reverse order of two parameters, no output
%i	increment by one, no output
%n	exclusive-or all parameters with 0140 (Datamedia 2500)
%B	BCD (16*(value/10)) + (value%10), no output
%D	Reverse coding (value - 2*(value%16)), no output (Delta Data).

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent "\E&a12c03Y" padded for 6 milliseconds. Note that the order of the row and column coordinates is reversed here and that the row and column are sent as two-digit integers. Thus its **cm** capability is "cm=6\E&%r%2c%2Y".

The Datamedia 2500 needs the current row and column sent encoded in binary using "%.". Terminals that use "%." need to be able to backspace the cursor (**le**) and to move the cursor up one line on the screen (**up**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (Programs using **termcap** must set terminal modes so that tabs are not expanded, so **\t** is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the Lear Siegler ADM-3a, which offsets row and column by a blank character, thus "cm=\E=%+ %+ ".

Row or column absolute cursor addressing can be given as single parameter capabilities **ch** (horizontal position absolute) and **cv** (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cm**. If there are parameterized local motions (*e.g.*, move *n* positions to the right) these can be given as **DO**, **LE**, **RI**, and **UP** with a single parameter indicating how many positions to move. These are primarily useful if the terminal does not have **cm**, such as the Tektronix 4025.

Cursor Motions

If the terminal has a fast way to home the cursor (to the very upper left corner of the screen), this can be given as **ho**. Similarly, a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **up** from the home position, but a program should never do this itself (unless **ll** does), because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as cursor address (0,0): to the top left corner of the screen, not of memory. (Therefore, the "\EH" sequence on Hewlett-Packard terminals cannot be used for **ho**.)

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **ce**. If the terminal can clear from the current position to the end of the display, this should be given as **cd**. **cd** must only be invoked from the first column of a line. (Therefore, it can be simulated by a request to delete a large number of lines, if a true **cd** is not available.)

Insert/Delete Line

If the terminal can open a new blank line before the line containing the cursor, this should be given as **al**; this must be invoked only from the first position of a line. The cursor must then appear at the left of the newly blank line. If the terminal can delete the line that the cursor is on, this should be given as **dl**; this must only be used from the first position on the line to be deleted. Versions of **al** and **dl** which take a single parameter and insert or delete that many lines can be given as **AL** and **DL**. If the terminal has a settable scrolling region (like the VT100), the command to set this can be described with the **cs** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command -- the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **sr** or **sf** on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

If the terminal has the ability to define a window as part of memory which all commands affect, it should be given as the parameterized string **wi**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order. (This terminfo(5) capability is described for completeness. It is unlikely that any **termcap**-using program will support it.)

If the terminal can retain display memory above the screen, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling may bring non-blank lines up from below or that scrolling back with **sr** may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character that can be described using **termcap**. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept-100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen then typing text separated by cursor motions. Type "abc   def" using local cursor motions (not spaces) between the "abc" and the "def". Then position the cursor before the "abc" and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the "abc" shifts over to the "def" which then move together around the end of the current line and onto the next as you insert, then you have the second type of terminal and should give the capability **in**, which stands for "insert null". While these are two logically separate attributes (one line *vs*. multi-line insert mode, and special treatment of untyped spaces), we have seen no terminals whose insert mode cannot be described with the single attribute.

The **termcap** entries can describe both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. Give as **im** the sequence to get into insert mode. Give as **ei** the sequence to leave insert mode. Now give as **ic** any sequence that needs to be sent just before each character to be inserted. Most terminals with a true insert mode will not give **ic**; terminals that use a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to **ic**. Do not give both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds in **ip** (a string option). Any other sequence that may need to be sent after insertion of a single character can also be given in **ip**. If your terminal needs to be placed into an ‘insert mode’ and needs a special code preceding each inserted character, then both **im**/ **ei** and **ic** can be given, and both will be used. The **IC** capability, with one parameter *n*, will repeat the effects of **ic** *n* times.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (*e.g.*, if there is a tab after the insertion position). If your terminal allows motion while in insert mode, you can give the capability **mi** to speed up inserting in this case. Omitting **mi** will affect only speed. Some terminals (notably Datamedia's) must not have **mi** because of the way their insert mode works.

Finally, you can specify **dc** to delete a single character, **DC** with one parameter *n* to delete *n* characters, and delete mode by giving **dm** and **ed** to enter and exit delete mode (which is any mode the terminal needs to be placed in for **dc** to work).

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as *standout mode*, representing a good high-contrast, easy-on-the-eyes format for highlighting error messages and other attention getters. (If you have a choice, reverse video plus half-bright is good, or reverse video alone.) The sequences to enter and exit standout mode are given as **so** and **se**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces or garbage characters on the screen, as the TVI 912 and Teleray 1061 do, then **sg** should be given to tell how many characters are left.

Codes to begin underlining and end underlining can be given as **us** and **ue**, respectively. Underline mode change garbage is specified by **ug**, similar to **sg**. If the terminal has a code to underline the current character and move the cursor one position to the right, such as the Microterm Mime, this can be given as **uc**.

Other capabilities to enter various highlighting modes include **mb** (blinking), **md** (bold or extra bright), **mh** (dim or half-bright), **mk** (blanking or invisible text), **mp** (protected), **mr** (reverse video), **me** (turn off *all* attribute modes), **as** (enter alternate character set mode), and **ae** (exit alternate character set mode). Turning on any of these modes singly may or may not turn off other modes.

If there is a sequence to set arbitrary combinations of mode, this should be given as **sa** (set attributes), taking 9 parameters. Each parameter is either 0 or 1, as the corresponding attributes is on or off. The 9 parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, and alternate character set. Not all modes need be supported by **sa**, only those for which corresponding attribute commands exist. (It is unlikely that a **termcap**-using program will support this capability, which is defined for compatibility with terminfo(5).)

Terminals with the "magic cookie" glitches (**sg** and **ug**), rather than maintaining extra attribute bits for each character cell, instead deposit special "cookies", or "garbage characters", when they receive mode-setting sequences, which affect the display algorithm.

Some terminals, such as the Hewlett-Packard 2621, automatically leave standout mode when they move to a new line or when the cursor is addressed. Programs using standout mode should exit standout mode on such terminals before moving the cursor or sending a newline. On terminals where this is not a problem, the **ms** capability should be present to say that this overhead is unnecessary.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), this can be given as **vb**; it must not move the cursor.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to change, for example, a non-blinking underline into an easier-to-find block or blinking underline), give this sequence as **vs**. If there is a way to make the cursor completely invisible, give that as **vi**. The capability **ve**, which undoes the effects of both of these modes, should also be given.

If your terminal correctly displays underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability **ul**. If overstrikes are erasable with a blank, this should be indicated by giving **eo**.

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local mode (this applies, for example, to the unshifted Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **ks** and **ke**. Otherwise the keypad is assumed to always transmit. The codes sent by the left-arrow, right-arrow, up-arrow, down-arrow, and home keys can be given as **kl**, **kr**, **ku**, **kd**, and **kh**, respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as **k0**, **k1**, ..., **k9**. If these keys have labels other than the default f0 through f9, the labels can be given as **l0**, **l1**, ..., **l9**. The codes transmitted by certain other special keys can be given: **kH** (home down), **kb** (backspace), **ka** (clear all tabs), **kt** (clear the tab stop in this column), **kC** (clear screen or erase), **kD** (delete character), **kL** (delete line), **kM** (exit insert mode), **kE** (clear to end of line), **kS** (clear to end of screen), **kI** (insert character or enter insert mode), **kA** (insert line), **kN** (next page), **kP** (previous page), **kF** (scroll forward/down), **kR** (scroll backward/up), and **kT** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, then the other five keys can be given as **K1**, **K2**, **K3**, **K4**, and **K5**. These keys are useful when the effects of a 3 by 3 directional pad are needed. The obsolete **ko** capability formerly used to describe "other" function keys has been completely supplanted by the above capabilities.

The **ma** entry is also used to indicate arrow keys on terminals that have single-character arrow keys. It is obsolete but still in use in version 2 of **vi** which must be run on some minicomputers due to memory limitations. This field is redundant with **kl**, **kr**, **ku**, **kd**, and **kh**. It consists of groups of two characters. In each group, the first character is what an arrow key sends, and the second character is the corresponding **vi** command. These commands are *h* for **kl**, *j* for **kd**, *k* for **ku**, *l* for **kr**, and *H* for **kh**. For example, the Mime would have "ma=^Hh^Kj^Zk^Xl" indicating arrow keys left (^H), down (^K), up (^Z), and right (^X). (There is no home key on the Mime.)

Tabs and Initialization

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **ti** and **te**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory-relative cursor addressing and not screen-relative cursor addressing, a screen-sized window must be fixed into the display for cursor addressing to work properly. This is also used for the Tektronix 4025, where **ti** sets the command character to be the one used by **termcap**.

Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the **termcap** description. They are normally sent to the terminal by the tset(1) program each time the user logs in. They will be printed in the following order: **is**; setting tabs using **ct** and **st**; and finally **if**. (Terminfo uses **i1-i2** instead of **is** and runs the program **iP** and prints **i3** after the other initializations.) A pair of sequences that does a harder reset from a totally unknown state can be analogously given as **rs** and **if**. These strings are output by the reset(1) program, which is used when the terminal gets into a wedged state. (Terminfo uses **r1-r3** instead of **rs**.) Commands are normally placed in **rs** and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the VT100 into 80-column mode would normally be part of **is**, but it causes an annoying glitch of the screen and is not normally needed since the terminal is usually already in 80-column mode.

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ta** (usually **^I**). A "backtab" command which moves leftward to the previous tab stop can be given as **bt**. By convention, if the terminal driver modes indicate that tab stops are being expanded by the computer rather than being sent to the terminal, programs should not use **ta** or **bt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs that are initially set every *n* positions when the terminal is powered up, then the numeric parameter **it** is given, showing the number of positions between tab stops. This is normally used by the tset(1) command to determine whether to set the driver mode for hardware tab expansion, and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the **termcap** description can assume that they are properly set.

If there are commands to set and clear tab stops, they can be given as **ct** (clear all tab stops) and **st** (set a tab stop in the current column of every row). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in **is** or **if**.

Delays

Certain capabilities control padding in the terminal driver. These are primarily needed by hardcopy terminals and are used by the tset(1) program to set terminal driver modes appropriately. Delays embedded in the capabilities **cr**, **sf**, **le**, **ff**, and **ta** will cause the appropriate delay bits to be set in the terminal driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**. For 4.2BSD tset(1), the delays are given as numeric capabilities **dC**, **dN**, **dB**, **dF**, and **dT** instead.

Miscellaneous

If the terminal requires other than a NUL (zero) character as a pad, this can be given as **pc**. Only the first character of the **pc** string is used.

If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**.

If the terminal has an extra "status line" that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, then the capability **hs** should be given. Special strings to go to a position in the status line and to return from the status line can be given as **ts** and **fs**. (**fs** must leave the cursor position in the same place that it was before **ts**. If necessary, the **sc** and **rc** strings can be included in **ts** and **fs** to get this effect.) The capability **ts** takes one parameter, which is the column number of the status line to which the cursor is to be moved. If escape sequences and other special commands such as tab work while in the status line, the flag **es** can be given. A string that turns off the status line (or otherwise erases its contents) should be given as **ds**. The status line is normally assumed to be the same width as the rest of the screen, *i.e.*, **co**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded), then its width in columns can be indicated with the numeric parameter **ws**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as **ff** (usually **^L**).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters), this can be indicated with the parameterized string **rp**. The first parameter is the character to be repeated and the second is the number of times to repeat it. (This is a terminfo(5) feature that is unlikely to be supported by a program that uses **termcap**.)

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with **CC**. A prototype command character is chosen which is used in all capabilities. This character is given in the **CC** capability to identify it. The following convention is supported on some UNIX systems: The environment is to be searched for a CC variable, and if found, all occurrences of the prototype character are replaced by the character in the environment variable. This use of the CC environment variable is a very bad idea, as it conflicts with make(1).

Terminal descriptions that do not represent a specific kind of known terminal, such as *switch*, *dialup*, *patch*, and *network*, should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to *virtual* terminal descriptions for which the escape sequences are known.)

If the terminal uses xoff/xon (DC3/DC1) handshaking for flow control, give **xo**. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted.

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, then this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as **mm** and **mo**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. An explicit value of 0 indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

If the terminal is one of those supported by the UNIX system virtual terminal protocol, the terminal number can be given as **vt**.

Media copy strings which control an auxiliary printer connected to the terminal can be given as **ps**: print the contents of the screen; **pf**: turn off the printer; and **po**: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. It is undefined whether the text is also displayed on the terminal screen when the printer is on. A variation **pO** takes one parameter and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. All text, including **pf**, is transparently passed to the printer while **pO** is in effect.

Strings to program function keys can be given as **pk**, **pl**, and **px**. Each of these strings takes two parameters: the function key number to program (from 0 to 9) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal-dependent manner. The differences among the capabilities are that **pk** causes pressing the given key to be the same as the user typing the given string; **pl** causes the string to be executed by the terminal in local mode; and **px** causes the string to be transmitted to the computer. Unfortunately, due to lack of a definition for string parameters in **termcap**, only terminfo(5) supports these capabilities.

For the xterm(1) (*ports/x11/xterm*) terminal emulator the traditional behavior in FreeBSD when exiting a pager such as less(1) or more(1), or an editor such as vi(1) is *NOT* to clear the screen after the program exits. If you prefer to clear the screen there are a number of "xterm-clear" entries that add this capability in the **termcap** file that you can use directly, or as examples.

Glitches and Braindamage

Hazeltine terminals, which do not allow '~' characters to be displayed, should indicate **hz**.

The **nc** capability, now obsolete, formerly indicated Datamedia terminals, which echo **\r \n** for carriage return then ignore a following linefeed.

Terminals that ignore a linefeed immediately after an **am** wrap, such as the Concept, should indicate **xn**.

If **ce** is required to get rid of standout (instead of merely writing normal text on top of it), **xs** should be given.

Teleray terminals, where tabs turn all characters moved over to blanks, should indicate **xt** (destructive tabs). This glitch is also taken to mean that it is not possible to position the cursor on top of a "magic cookie", and that to erase standout mode it is necessary to use delete and insert line.

The Beehive Superbee, which is unable to correctly transmit the ESC or **^C** characters, has **xb**, indicating that the "f1" key is used for ESC and "f2" for ^C. (Only certain Superbees have this problem, depending on the ROM.)

Other specific terminal problems may be corrected by adding more capabilities of the form **x***x*.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **tc** can be given with the name of the similar terminal. This capability must be *last*, and the combined length of the entries must not exceed 1024. The capabilities given before **tc** override those in the terminal type invoked by **tc**. A capability can be canceled by placing **xx@** to the left of the **tc** invocation, where **xx** is the capability. For example, the entry

```
hn|2621-nl:ks@:ke@:tc=2621:
```

defines a "2621-nl" that does not have the **ks** or **ke** capabilities, hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

FILES

/usr/share/misc/termcap	File containing terminal descriptions.
/usr/share/misc/termcap.db	Hash database file containing terminal descriptions (see cap_mkdb(1)).

SEE ALSO

cap_mkdb(1), ex(1), more(1), tset(1), ul(1), vi(1), xterm(1) (*ports/x11/xterm*), ncurses(3), printf(3), termcap(3), term(5)

CAVEATS AND BUGS

The *Note*: **termcap** functions were replaced by terminfo(5) in AT&T System V UNIX Release 2.0. The transition will be relatively painless if capabilities flagged as "obsolete" are avoided.

Lines and columns are now stored by the kernel as well as in the termcap entry. Most programs now use the kernel information primarily; the information in this file is used only if the kernel does not have any information.

The vi(1) program allows only 256 characters for string capabilities, and the routines in termlib(3) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

Not all programs support all entries.

HISTORY

The **termcap** file format appeared in 3BSD.