NAME

screen – screen manager with VT100/ANSI terminal emulation

SYNOPSIS

screen [*–options*] [*cmd* [*args*]]
screen –r [[*pid.*]*tty*[*.host*]]

DESCRIPTION

Screen is a full-screen window manager that multiplexes a physical terminal between several processes (typically interactive shells). Each virtual terminal provides the functions of a DEC VT100 terminal and, in addition, several control functions from the ANSI X3.64 (ISO 6429) and ISO 2022 standards (e. g. insert/delete line and support for multiple character sets). There is a scrollback history buffer for each virtual terminal and a copy-and-paste mechanism that allows moving text regions between windows.

When *screen* is called, it creates a single window with a shell in it (or the specified command) and then gets out of your way so that you can use the program as you normally would. Then, at any time, you can create new (full-screen) windows with other programs in them (including more shells), kill existing windows, view a list of windows, turn output logging on and off, copy-and-paste text between windows, view the scrollback history, switch between windows in whatever manner you wish, etc. When a program terminates, *screen* (per default) kills the window that contained it. If this window was in the foreground, the display switches to the previous window; if none are left, *screen* exits.

Everything you type is sent to the program running in the current window. The only exception to this is the one keystroke that is used to initiate a command to the window manager. By default, each command begins with a control-a (abbreviated C-a from now on), and is followed by one other keystroke. The command character and all the key bindings can be fully customized to be anything you like, though they are always two characters in length.

The standard way to create a new window is to type ‘‘C-a c’’. This creates a new window running a shell and switches to that window immediately, regardless of the state of the process running in the current window. Similarly, you can create a new window with a custom command in it by first binding the command to a keystroke (in your .screenrc file or at the ‘‘C-a :’’ command line) and then using it just like the ‘‘C-a c’’ command. In addition, new windows can be created by running a command like:

screen emacs prog.c

from a shell prompt within a previously created window. This will not run another copy of *screen*, but will instead supply the command name and its arguments to the window manager (specified in the $STY environment variable) who will use it to create the new window. The above example would start the emacs editor (editing prog.c) and switch to its window.

If ‘‘/etc/utmp’’ is writable by *screen*, an appropriate record will be written to this file for each window, and removed when the window is terminated. This is useful for working with ‘‘talk’’, ‘‘script’’, ‘‘shutdown’’, ‘‘rsend’’, ‘‘sccs’’ and other similar programs that use the utmp file to determine who you are. As long as *screen* is active on your terminal, the terminal’s own record is removed from the utmp file. See also ‘‘C-a L’’.

GETTING STARTED

Before you begin to use *screen* you’ll need to make sure you have correctly selected your terminal type, just as you would for any other termcap/terminfo program. (You can do this by using *tset* for example.)

If you're impatient and want to get started without doing a lot more reading, you should remember this one command: ''C-a ?''. Typing these two characters will display a list of the available *screen* commands and their bindings. Each keystroke is discussed in the section ''DEFAULT KEY BINDINGS''. The manual section ''CUSTOMIZATION'' deals with the contents of your .screenrc.

If possible, choose a version of your terminal's termcap that has automatic margins turned *off*. This will ensure an accurate and optimal update of the screen in all circumstances. The next best thing is an auto-margin terminal that allows the last position on the screen to be updated without scrolling the screen (such as a VT100). This also allows the entire screen to be updated. Lastly, if all you've got is a ''true'' auto-margin terminal *screen* will be content to use it, but updating a character put into the last position on the screen may not be possible until the screen scrolls or the character is moved into a safe position in some other way. This delay can be shortened by using a terminal with insert-character capability.

If your terminal is of the second type (firm-margined 'am'), you will want to let *screen* know about this, since a normal termcap doesn't distinguish this type of automatic margins from a ''true'' 'am' terminal. You do this by specifying the 'LP' capability in your termcap (see the ''termcap'' .screenrc command), or by using the **–L** command-line option. *Screen* needs this information to correctly update the screen. Note that a 'xv' together with an 'am' flag effects like 'LP'.

If you are using a ''true'' auto-margin terminal (no 'LP') at low baud rates, you may want to turn on a more optimal output mode by including the flag 'OP' in your termcap entry, or by specifying the **–O** command-line option. The trade-off is that *screen* will no-longer accurately emulate the VT100's line-end quirks (e. g. the screen will scroll after putting *one* character in the last screen position).

COMMAND-LINE OPTIONS

Screen has the following command-line options:

–a include *all* capabilities (with some minor exceptions) in each window's termcap, even if *screen* must redraw parts of the display in order to implement a function.

–A Adapt the sizes of all windows to the size of the current terminal. By default, *screen* tries to restore its old window sizes when attaching to resizable terminals (those with ''WS'' in its description, e.g. suncmd or some xterm).

–c *file*
override the default configuration file from ''$HOME/.screenrc'' to *file*.

–d|**–D** [*pid.tty.host*]
does not start *screen*, but detaches the elsewhere running *screen* session. It has the same effect as typing ''C-a d'' from *screen's* controlling terminal. **–D** is the equivalent to the power detach key. If no session can be detached, this option is ignored. The combination ''screen –D –r'' can be used to 'transport' the elsewhere running session to this terminal and logout there. Note: It is a good idea to have the status of your sessions checked by means of ''screen –list''.

–e *xy* specifies the command character to be *x* and the character generating a literal command character to *y* (when typed after the command character). The default is ''C-a'' and 'a', which can be specified as ''-eˆAa''. When creating a *screen* session, this option sets the default command character. In a multiuser session all users added will start off with this command character. But when attaching to an already running session, this option changes only the command character of the attaching user. This option is equivalent to either the commands ''defescape'' or ''escape'' respectively.

–f, **–fn**, and **–fa**
turns flow-control on, off, or ''automatic switching mode''. This can also be defined through the ''defflow'' .screenrc command.

–h *num*

Specifies the history scrollback buffer to be *num* lines high.

–i will cause the interrupt key (usually C-c) to interrupt the display immediately when flow-control is on. See the ‘‘defflow’’ .screenrc command for details. The use of this option is discouraged.

–l and **–ln**
turns login mode on or off (for /etc/utmp updating). This can also be defined through the ‘‘deflogin’’ .screenrc command.

–ls and **–list**
does not start *screen*, but prints a list of *pid.tty.host* strings identifying your *screen* sessions. Sessions marked ‘detached’ can be resumed with ‘‘screen -r’’. Those marked ‘attached’ are running and have a controlling terminal. Sessions marked as ‘dead’ should be thoroughly checked and removed. Ask your system administrator if you are not sure. Remove sessions with the **-wipe** option.

–L tells *screen* your auto-margin terminal has a writable last-position on the screen. This can also be set in your .screenrc by specifying ‘LP’ in a ‘‘termcap’’ command.

–m causes *screen* to ignore the $STY environment variable. With ‘‘screen -m’’ creation of a new session is enforced, regardless whether *screen* is called from within another *screen* session or not.

–O selects a more optimal output mode for your terminal rather than true VT100 emulation (only affects auto-margin terminals without ‘LP’). This can also be set in your .screenrc by specifying ‘OP’ in a ‘‘termcap’’ command.

–r [*pid.tty.host*]
resumes a detached *screen* session. No other options (except ‘‘-d -r’’ or ‘‘-D -r’’) may be specified, though an optional prefix of [*pid.*]*tty.host* may be needed to distinguish between multiple detached *screen* sessions.

–R attempts to resume the first detached *screen* session it finds. If successful, all other command-line options are ignored. If no detached session exists, starts a new session using the specified options, just as if **–R** had not been specified. The option is set by default if *screen* is run as a login-shell.

–s sets the default shell to the program specified, instead of the value in the environment variable $SHELL (or ‘‘/bin/sh’’ if not defined). This can also be defined through the ‘‘shell’’ .screenrc command.

–S *sessionname*
When creating a new session, this option can be used to specify a meaningful name for the session. This name identifies the session for ‘‘screen -list’’ and ‘‘screen -r’’ actions. It substitutes the default [*tty.host*] suffix.

–t *name*
sets the title (a. k. a.) for the default shell or specified program. See also the ‘‘shelltitle’’ .screenrc command.

–v Print version number.

–wipe
does the same as ‘‘screen -ls’’, but removes destroyed sessions instead of marking them as ‘dead’.

–x Attach to a not detached *screen* session. (Multi display mode).

DEFAULT KEY BINDINGS

As mentioned, each *screen* command consists of a ‘‘C-a’’ followed by one other character. For your convenience, all commands that are bound to lower-case letters are also bound to their control character counterparts (with the exception of ‘‘C-a a’’; see below), thus, ‘‘C-a c’’ as well as ‘‘C-a C-c’’ can be used to create a window. See section ‘‘CUSTOMIZATION’’ for a description of the command.

The following table shows the default key bindings:

C-a '		
C-a "	(select)	Prompt for a window name or number to switch to.
C-a 0	(select 0)	
...	...	
C-a 9	(select 9)	Switch to window number 0 – 9.
C-a C-a	(other)	Toggle to the window displayed previously. Note that this binding defaults to the command character typed twice, unless overridden; for instance, if you use the option ''**–e]x**'', this function becomes '']]'', not '']C-a''.
C-a a	(meta)	Send the command character (C-a) to window. See *escape* command.
C-a A	(title)	Allow the user to enter a name for the current window.
C-a b		
C-a C-b	(break)	Send a break to window.
C-a B	(pow_break)	Reopen the terminal line and send a break.
C-a c		
C-a C-c	(screen)	Create a new window with a shell and switch to that window.
C-a C	(clear)	Clear the screen.
C-a d		
C-a C-d	(detach)	Detach *screen* from this terminal.
C-a D D	(pow_detach)	Detach and logout.
C-a f		
C-a C-f	(flow)	Toggle flow *on*, *off* or *auto*.
C-a C-g	(vbell)	Toggles *screen's* visual bell mode.
C-a h	(hardcopy)	Write a hardcopy of the current window to the file ''hardcopy.*n*''.
C-a H	(log)	Begins/ends logging of the current window to the file ''screenlog.*n*''.
C-a i		
C-a C-i	(info)	Show info about this window.
C-a k		
C-a C-k	(kill)	Destroy current window.
C-a l		
C-a C-l	(redisplay)	Fully refresh current window.
C-a L	(login)	Toggle this windows login slot. Available only if *screen* is configured to update the utmp database.
C-a m		
C-a C-m	(lastmsg)	Repeat the last message displayed in the message line.
C-a M	(monitor)	Toggles monitoring of the current window.
C-a space		
C-a n		
C-a C-n	(next)	Switch to the next window.
C-a N	(number)	Show the number (and title) of the current window.
C-a backspace		
C-a h		
C-a p		

C-a C-p	(prev)	Switch to the previous window (opposite of **C-a n**).
C-a q **C-a C-q**	(xon)	Send a control-q to the current window.
C-a r **C-a C-r**	(wrap)	Toggle the current window's line-wrap setting (turn the current window's automatic margins on and off).
C-a s **C-a C-s**	(xoff)	Send a control-s to the current window.
C-a t **C-a C-t**	(time)	Show system information.
C-a v **C-a C-v**	(version)	Display the version and compilation date.
C-a w **C-a C-w**	(windows)	Show a list of window.
C-a W	(width)	Toggle 80/132 columns.
C-a x **C-a C-x**	(lockscreen)	Lock this terminal.
C-a z **C-a C-z**	(suspend)	Suspend *screen*. Your system must support BSD-style job-control.
C-a Z	(reset)	Reset the virtual terminal to its ''power-on'' values.
C-a .	(dumptermcap)	Write out a ''.termcap'' file.
C-a ?	(help)	Show key bindings.
C-a C-	(quit)	Kill all windows and terminate *screen*.
C-a :	(colon)	Enter command line mode.
C-a [**C-a C-[** **C-a esc**	(copy)	Enter copy/scrollback mode.
C-a]	(paste .)	Write the contents of the paste buffer to the stdin queue of the current window.
C-a { **C-a }**	(history)	Copy and paste a previous (command) line.
C-a >	(writebuf)	Write paste buffer to a file.
C-a <	(readbuf)	Reads the screen-exchange file into the paste buffer.
C-a =	(removebuf)	Removes the file used by **C-a <** and **C-a >**.
C-a ,	(license)	Shows where *screen* comes from, where it went to and why you can use it.
C-a _	(silence)	Start/stop monitoring the current window for inactivity.

CUSTOMIZATION

The ''socket directory'' defaults either to $HOME/.screen or simply to /tmp/screens or preferably to /usr/local/screens chosen at compile-time. If *screen* is installed setuid-root, then the administrator should compile *screen* with an adequate (not NFS mounted) socket directory. If *screen* is not running setuid-root, the user can specify any mode 777 directory in the environment variable $SCREENDIR.

When *screen* is invoked, it executes initialization commands from the files ‘‘/usr/local/etc/screenrc’’ and ‘‘.screenrc’’ in the user’s home directory. These are the ‘‘programmer’s defaults’’ that can be overridden in the following ways: for the global screenrc file *screen* searches for the environment variable $SYSSCREENRC (this override feature may be disabled at compile-time). The user specific screenrc file is searched in $ISCREENRC, then $SCREENRC, then $HOME/.iscreenrc and finally defaults to $HOME/.screenrc. The command line option **-c** takes precedence over the above user screenrc files.

Commands in these files are used to set options, bind functions to keys, and to automatically establish one or more windows at the beginning of your *screen* session. Commands are listed one per line, with empty lines being ignored. A command’s arguments are separated by tabs or spaces, and may be surrounded by single or double quotes. A ‘#’ turns the rest of the line into a comment, except in quotes. Unintelligible lines are warned about and ignored. Commands may contain references to environment variables. The syntax is the shell-like "$VAR " or "${VAR}". Note that this causes incompatibility with previous *screen* versions, as now the ’$’-character has to be protected with ’\’ if no variable substitution shall be performed. A string in single-quotes is also protected from variable substitution.

Customization can also be done ’on-line’. To enter the command mode type ‘C-a :’. Note that commands starting with ‘‘def’’ change default values, while others change current settings.

The following commands are available:

acladd *usernames*

Enable users to fully access this screen session. *Usernames* can be one user or a comma seperated list of users. This command enables to attach to the *screen* session and performs the equivalent of ‘aclchg *usernames* +rwx "#?"’. executed. To add a user with restricted access, use the ‘aclchg’ command below. Multi user mode only.

aclchg *usernames permbits list*

Change permissions for a comma seperated list of users. Permission bits are represented as ‘r’, ‘w’ and ‘x’. Prefixing ‘+’ grants the permission, ‘-’ removes it. The third parameter is a comma seperated list of commands and/or windows (specified either by number or title). The special list ‘#’ refers to all windows, ‘?’ to all commands. if *usernames* consists of a single ‘∗’, all known users are affected. A command can be executed when the user has the ‘x’ bit for it. The user can type input to a window when he has its ‘w’ bit set and no other user obtains a writelock for this window. Other bits are currently ignored. To withdraw the writelock from another user in window 2: ‘aclchg *username* -w+w 2’. To allow readonly access to the session: ‘aclchg *username* -w "#"’. As soon as a user’s name is known to *screen* he can attach to the session and (per default) has full permissions for all command and windows. Execution permission for the acl commands, ‘at’ and others should also be removed or the user may be able to regain write permission. Multi user mode only.

acldel *username*

Remove a user from *screen*’s access control list. If currently attached, all the user’s displays are detached from the session. He cannot attach again. Multi user mode only.

activity *message*

When any activity occurs in a background window that is being monitored, *screen* displays a notification in the message line. The notification message can be re-defined by means of the ‘‘activity’’ command. Each occurrence of ‘%’ in *message* is replaced by the number of the window in which activity has occurred, and each occurrence of ‘˜’ is replaced by the definition for bell in your termcap (usually an audible bell). The default message is

’Activity in window %’

Note that monitoring is off for all windows by default, but can be altered by use of the ‘‘monitor’’

command (C-a M).

allpartial on|off

If set to on, only the current cursor line is refreshed on window change. This affects all windows and is useful for slow terminal lines. The previous setting of full/partial refresh for each window is restored with ‘‘allpartial off’’. This is a global flag that immediately takes effect on all windows overriding the ‘‘partial’’ settings. It does not change the default redraw behaviour of newly created windows.

at [*identifier*][**#**|*****|**%**] *command* [*args ...*]

Execute a command at other displays or windows as if it had been entered there. ‘‘At’’ changes the context (the ‘current window’ or ‘current display’ setting) of the command. If the first parameter describes a non-unique context, the command will be executed multiple times. If the first parameter is of the form ‘*identifier**’ then identifier is matched against user names. The command is executed once for each display of the selected user(s). If the first parameter is of the form ‘*identifier*%’ identifier is matched against displays. Displays are named after the ttys they attach. The prefix ‘/dev/’ or ‘/dev/tty’ may be omitted from the identifier. If *identifier* has a ‘#’ or nothing appended it is matched against window numbers and titles. Omitting an identifier in front of the ‘#’, ‘*’ or ‘%’-character selects all users, displays or windows because a prefix-match is performed. Note that on the affected display(s) a short message will describe what happened. Caution: Permission is checked for the owners or the affected display(s), not for the initiator of the ‘at’ command.

autodetach on|off

Sets whether *screen* will automatically detach upon hangup, which saves all your running programs until they are resumed with a **screen -r** command. When turned off, a hangup signal will terminate *screen* and all the processes it contains. Autodetach is on by default.

autonuke on|off

Sets whether a clear screen sequence should nuke all the output that has not been written to the terminal. See also ‘‘obuflimit’’.

bell *message*

When a bell character is sent to a background window, *screen* displays a notification in the message line. The notification message can be re-defined by means of the ‘‘bell’’ command. Each occurrence of ‘%’ in *message* is replaced by the number of the window to which a bell has been sent, and each occurrence of ‘~’ is replaced by the definition for bell in your termcap (usually an audible bell). The default message is

’Bell in window %’

An empty message can be supplied to the ‘‘bell’’ command to suppress output of a message line (bell "").

bind *key* [*command* [*args*]]

Bind a command to a key. By default, most of the commands provided by *screen* are bound to one or more keys as indicated in the ‘‘DEFAULT KEY BINDINGS’’ section, e. g. the command to create a new window is bound to ‘‘C-c’’ and ‘‘c’’. The ‘‘bind’’ command can be used to redefine the key bindings and to define new bindings. The *key* argument is either a single character, a two-character sequence of the form ‘‘^x’’ (meaning ‘‘C-x’’), a backslash followed by an octal number (specifying the ASCII code of the character), or a backslash followed by a second character, such as ‘‘\^’’ or ‘‘\\’’. The argument can also be quoted, if you like. If no further argument is given, any previously established binding for this key is removed. The *command* argument can be any command listed in this

section.

Some examples:

```
bind ' ' windows
bind ^f screen telnet foobar
bind \033 screen -ln -t root -h 1000 9 su
```

would bind the space key to the command that displays a list of windows (so that the command usually invoked by ''C-a C-w'' would also be available as ''C-a space''), bind ''C-f'' to the command ''create a window with a TELNET connection to foobar'', and bind ''escape'' to the command that creates an non-login window with a. k. a. ''root'' in slot #9, with a super-user shell and a scrollback buffer of 1000 lines.

bindkey [**-d**] [**-m**] [**-a**] [[**-k**|**-t**] *string* [*cmd args*]]

This command manages screen's input translation tables. Every entry in one of the tables tells screen how to react if a certain sequence of characters is encountered. There are three tables: one that should contain actions programmed by the user, one for the default actions used for terminal emulation and one for screen's copy mode to do cursor movement. See section ''INPUT TRANSLATION'' for a list of default key bindings.
If the **-d** option is given, bindkey modifies the default table, **-m** changes the copy mode table and with neither option the user table is selected. The argument *string* is the sequence of characters to which an action is bound. This can either be a fixed string or a termcap keyboard capability name (selectable with the **-k** option).
Some keys on a VT100 terminal can send a different string if application mode is turned on (e.g the cursor keys). Such keys have two entries in the translation table. You can select the application mode entry by specifying the **-a** option.
The **-t** option tells screen not to do intercharacter timing. One cannot turn off the timing if a termcap capabilty is used.
Cmd can be any of screen's commands with an arbitrary number of *args*. If *cmd* is omitted the key-binding is removed from the table.
Here are some examples of keyboard bindings:

bindkey -d
Show all of the default key bindings. The application mode entries are marked with [A].

bindkey -k k1 select 1
Make the "F1" key switch to window one.

bindkey -t foo stuff barfoo
Make "foo" an abrevation of the word "barfoo". Timeout is disabled so that users can type slowly.

bindkey "\024" mapdefault
This keybinding makes ''^T'' an escape character for keybindings. If you did the above ''stuff barfoo'' binding, you can enter the word ''foo'' by typing ''^Tfoo''. If you want to insert a ''^T'' you have to press the key twice (i.e. escape the escape binding).

bindkey -k F1 command
Make the F11 (not F1!) key an alternative screen escape (besides ^A). Note that ''F11 F11'' does not work in the current release of screen.

break [*duration*]

Send a break signal for *duration*∗0.25 seconds to this window. Most useful if a character device is attached to the window rather than a shell process.

bufferfile [*exchange-file*]

Change the filename used for reading and writing with the paste buffer. If the optional argument to the ‘‘bufferfile’’ command is omitted, the default setting (‘‘/tmp/screen-exchange’’) is reactivated. The following example will paste the system’s password file into the *screen* window (using the paste buffer, where a copy remains):

C-a : bufferfile /etc/passwd
C-a < C-a]
C-a : bufferfile

c1 [**on**|**off**]

Change c1 code processing. ‘‘C1 on’’ tells screen to treat the input characters between 128 and 159 as control functions. Such an 8-bit code is normally the same as ESC followed by the corresponding 7-bit code. The default setting is to process c1 codes and can be changed with the ‘‘defc1’’ command. Users with fonts that have usable characters in the c1 positions may want to turn this off.

chdir [*directory*]

Change the *current directory* of *screen* to the specified directory or, if called without an argument, to your home directory (the value of the environment variable $HOME). All windows that are created by means of the ‘‘screen’’ command from within ‘‘.screenrc’’ or by means of ‘‘C-a : screen ...’’ or ‘‘C-a c’’ use this as their default directory. Without a chdir command, this would be the directory from which *screen* was invoked. Hardcopy and log files are always written to the *window’s* default directory, *not* the current directory of the process running in the window. You can use this command multiple times in your .screenrc to start various windows in different default directories, but the last chdir value will affect all the windows you create interactively.

clear

Clears the current window and saves its image to the scrollback buffer.

colon

Allows you to enter ‘‘.screenrc’’ command lines. Useful for on-the-fly modification of key bindings, specific window creation and changing settings. Note that the ‘‘set’’ keyword no longer exists! Usually commands affect the current window rather than default settings for future windows. Change defaults with commands starting with ’def...’.

If you consider this as the ‘Ex command mode’ of *screen*, you may regard ‘‘C-a esc’’ (copy mode) as its ‘Vi command mode’.

command

This command has the same effect as typing the screen escape character (^A). It is probably only useful for key bindings. See also ‘‘bindkey’’.

console [**on**|**off**]

Grabs or ungrabs the machines console output to a window.

copy

Enter copy/scrollback mode. This allows you to copy text from the current window and its history into the paste buffer. In this mode a vi-like ‘full screen editor’ is active:
Movement keys:
h, **j**, **k**, **l** move the cursor line by line or column by column.
0, **^** and **$** move to the leftmost column, to the first or last non-whitespace character on the line.

H, **M** and **L** move the cursor to the leftmost column of the top, center or bottom line of the window.
+ and **–** positions one line up and down.
G moves to the specified absolute line (default: end of buffer).
| moves to the specified absolute column.
w, **b**, **e** move the cursor word by word.
C-u and **C-d** scroll the display up/down by the specified amount of lines while preserving the cursor position. (Default: half screen-full).
C-b and **C-f** scroll the display up/down a full screen.
g moves to the beginning of the buffer.
% jumps to the specified percentage of the buffer.

Note:
Emacs style movement keys can be customized by a .screenrc command. (E. g. markkeys "h=^B:l=^F:$=^E") There is no simple method for a full emacs-style keymap, as this involves multi-character codes.

Marking:
The copy range is specified by setting two marks. The text between these marks will be highlighted. Press
space to set the first or second mark respectively.
Y and **y** used to mark one whole line or to mark from start of line.
W marks exactly one word.
Repeat count:
Any of these commands can be prefixed with a repeat count number by pressing digits
0..**9** which is taken as a repeat count.
Example: ‘‘C-a C-[H 10 j 5 Y’’ will copy lines 11 to 15 into the paste buffer.
Searching:
/ *Vi*-like search forward.
? *Vi*-like search backward.
C-a s *Emacs* style incremental search forward.
C-r *Emacs* style reverse i-search.
Specials:
There are however some keys that act differently than in *vi*. *Vi* does not allow one to yank rectangular blocks of text, but *screen* does. Press
c or **C** to set the left or right margin respectively. If no repeat count is given, both default to the current cursor position.
Example: Try this on a rather full text screen: ‘‘C-a [M 20 l SPACE c 10 l 5 j C SPACE’’.

This moves one to the middle line of the screen, moves in 20 columns left, marks the beginning of the paste buffer, sets the left column, moves 5 columns down, sets the right column, and then marks the end of the paste buffer. Now try:
‘‘C-a [M 20 l SPACE 10 l 5 j SPACE’’

and notice the difference in the amount of text copied.
J joins lines. It toggles between 3 modes: lines separated by a newline character (012), lines glued seamless, lines separated by a single whitespace. Note that you can prepend the newline character with a carriage return character, by issuing a ‘‘crlf on’’.
v is for all the *vi* users with ‘‘:set numbers’’ – it toggles the left margin between column 9 and 1. Press
a before the final space key to toggle in append mode. Thus the contents of the paste buffer will not be overwritten, but is appended to.
A toggles in append mode and sets a (second) mark.
> sets the (second) mark and writes the contents of the paste buffer to the screen-exchange file

(/tmp/screen-exchange per default) once copy-mode is finished.
This example demonstrates how to dump the whole scrollback buffer to that file: ‘‘C-A [g SPACE G $ >’’.
C-g gives information about the current line and column.
@ does nothing. Does not even exit copy mode.

copy_reg [*key*]

No longer exists, use ‘‘readreg’’ instead.

crlf on|off

This affects the copying of text regions with the ‘C-a [’ command. If it is set to ‘on’, lines will be separated by the two character sequence ‘CR’ - ‘LF’. Otherwise (default) only ‘LF’ is used.

debug on|off

Turns runtime debugging on or off. If *screen* has been compiled with option -DDEBUG debugging available and is turned on per default. Note that this command only affects debugging output from the main ‘‘SCREEN’’ process.

defc1 on|off

Same as the **c1** command except that the default setting for new windows is changed. Initial setting is ‘on’.

defautonuke on|off

Same as the **autonuke** command except that the default setting for new displays is changed. Initial setting is ‘off’. Note that you can use the special ‘AN’ terminal capability if you want to have a dependency on the terminal type.

defescape *xy*

Set the default command characters. This is equivalent to the ‘‘escape’’ except that it is useful multiuser sessions only. In a multiuser session ‘‘escape’’ changes the command character of the calling user, where ‘‘defescape’’ changes the default command characters for users that will be added later.

defflow on|off|auto [**interrupt**]

Same as the **flow** command except that the default setting for new windows is changed. Initial setting is ‘auto’. Specifying ‘‘defflow auto interrupt’’ is the same as the command-line options **–fa** and **–i**.

defgr on|off

Same as the **gr** command except that the default setting for new windows is changed. Initial setting is ‘off’.

defkanji jis|sjis|euc

Same as the **kanji** command except that the default setting for new windows is changed. Initial setting is ‘off’, i.e. ‘jis’.

deflogin on|off

Same as the **login** command except that the default setting for new windows is changed. This is initialized with ‘on’ as distributed (see config.h.in).

defmode *mode*

The mode of each newly allocated pseudo-tty is set to *mode*. *Mode* is an octal number. When no ‘‘defmode’’ command is given, mode 0622 is used.

defmonitor on|off

Same as the **monitor** command except that the default setting for new windows is changed. Initial setting is ‘off’.

defobuflimit *limit*

Same as the **obuflimit** command except that the default setting for new displays is changed. Initial setting is 256 bytes. Note that you can use the special ’OL’ terminal capability if you want to have a dependency on the terminal type.

defscrollback *num*

Same as the **scrollback** command except that the default setting for new windows is changed. Initial setting is 100.

defwrap on|off

Same as the **wrap** command except that the default setting for new windows is changed. Initially linewrap is on and can be toggled with the ‘‘wrap’’ command (‘‘C-a r’’) or by means of "C-a : wrap on|off".

defwritelock on|off|auto

Same as the **writelock** command except that the default setting for new windows is changed. Initially writelocks will operate in automatic mode.

defzombie [*keys*]

Synonym to the **zombie** command. Both currently change the default. See there.

detach

Detach the *screen* session (disconnect it from the terminal and put it into the background). This returns you to the shell where you invoked *screen*. A detached *screen* can be resumed by invoking *screen* with the **–r** option. (See also section ‘‘COMMAND-LINE OPTIONS’’.)

dumptermcap

Write the termcap entry for the virtual terminal optimized for the currently active window to the file ‘‘.termcap’’ in the user’s ‘‘$HOME/.screen’’ directory (or wherever *screen* stores its sockets. See the ‘‘FILES’’ section below). This termcap entry is identical to the value of the environment variable $TERMCAP that is set up by *screen* for each window. For terminfo based systems you will need to run a converter like *captoinfo* and then compile the entry with *tic*.

echo [**-n**] *message*

The echo command may be used to annoy *screen* users with a ’message of the day’. Typically installed in a global /local/etc/screenrc. See also ‘‘sleep’’. Echo is also useful for online checking of environment variables.

escape *xy*

Set the command character to *x* and the character generating a literal command character to *y* (just like in the –e option). Each argument is either a single character, a two-character sequence of the form ‘‘^x’’ (meaning ‘‘C-x’’), a backslash followed by an octal number (specifying the ASCII code of the character), or a backslash followed by a second character, such as ‘‘\^’’ or ‘‘\\’’. The default is ‘‘^Aa’’.

exec [[*fdpat*] *newcommand* [*args ...*]]

Run a subprocess (newcommand) in the current window. The flow of data between newcommand’s stdin/stdout/stderr, the process already running (shell) and screen itself (window) is controlled by the filedescriptor pattern fdpat. This pattern is basically a three character sequence representing stdin, stdout and stderr of newcommand. A dot (.) connects the file descriptor to *screen*. An exclamation mark (!) causes the file descriptor to be connected to the already running process. A colon (:) combines both. User input will go to newcommand unless newcommand requests the old process’ output (fdpats first character is ‘!’ or ‘:’) or a pipe (|) is added to the end of fdpat.
Invoking ‘exec’ without arguments shows name and arguments of the currently running subprocess in this window.
When a subprocess is running the ‘kill’ command will affect it instead of the windows process.
Refer to the postscript file ‘fdpat.ips’ for illustration of all 21 possible combinations. Each drawing shows the numbers 210 representing the three file descriptors of newcommand. The box marked ‘W’ is usual pty that has the old process (shell) on its slave side. The box marked ‘P’ is the secondary pty that now has *screen* at its master side.

Abbreviations:
Whitespace between the word ‘exec’ and fdpat and the command can be omitted. Trailing dots and a fdpat consisting only of dots can be omitted. A simple ‘|’ is synonymous for the pattern ‘!..|’; the word exec can be omitted here and can always be replaced by ‘!’.

Examples:

```
exec ... /bin/sh
exec /bin/sh
!/bin/sh
```

Creates another shell in the same window, while the original shell is still running. Output of both shells is displayed and user input is sent to the new /bin/sh.

```
exec !.. stty 19200
exec ! stty 19200
!!stty 19200
```

Set the speed of the window’s tty. If your stty command operates on stdout, then add another ‘!’.

```
exec !..| less
|less
```

This adds a pager to the window output. The special character ‘|’ is needed to give the user control over the pager although it gets its input from the original process.

```
!:sed -n s/.*Error.*/\007/p
```

Sends window output to both, the user and the sed command. The sed inserts an additional bell character (oct. 007) to the window output seen by *screen*. This will cause "Bell in window x" messages, whenever the string "Error" appears in the window.

flow [**on**|**off**|**auto**]

Sets the flow-control mode for this window. Without parameters it cycles the current window’s flow-control setting from "automatic" to "on" to "off". See the discussion on ‘‘FLOW-CONTROL’’ later on in this document for full details and note, that this is subject to change in future releases. Default is set by ‘defflow’.

gr [**on**|**off**]

Turn GR charset switching on/off. Whenever screens sees an input char with an 8th bit set, it will use the charset stored in the GR slot and print the character with the 8th bit stripped. The default (see also ‘‘defgr’’) is not to process GR switching because otherwise the ISO88591 charset would not work.

hardcopy

Writes out the currently displayed image to a file *hardcopy.n* in the window’s default directory, where *n* is the number of the current window. This either appends or overwrites the file if it exists. See below.

hardcopy_append on|**off**

If set to "on", *screen* will append to the "hardcopy.n" files created by the command ‘‘C-a h’’, otherwise these files are overwritten each time. Default is ‘off’.

hardcopydir *directory*

Defines a directory where hardcopy files will be placed. If unset, hardcopys are dumped in *screen*’s current working directory.

hardstatus [**on**|**off**]

Toggles the use of the terminal’s hardware status line. If "on", *screen* will use this facility to display one line messages. Otherwise these messages are overlayed in reverse video mode at the display line. Note that the hardstatus feature can only be used if the termcap/terminfo capabilities "hs", "ts", "fs" and "ds" are set properly. Default is ‘on’ whenever the "hs" capability is present.

height [*lines*]

Set the display height to a specified number of lines. When no argument is given it toggles between 24 and 42 lines display.

help

Not really a online help, but displays a help screen showing you all the key bindings. The first pages list all the internal commands followed by their current bindings. Subsequent pages will display the custom commands, one command per key. Press space when you’re done reading each page, or return to exit early. All other characters are ignored, except for the command character, which will allow you to execute commands even when the help screen is still visible. See also ‘‘DEFAULT KEY BINDINGS’’ section.

history

Usually users work with a shell that allows easy access to previous commands. For example csh has the command ‘‘!!’’ to repeat the last command executed. *Screen* allows you to have a primitive way of re-calling ‘‘the command that started ...’’: You just type the first letter of that command, then hit ‘C-a {’ and *screen* tries to find a previous line that matches with the ‘prompt character’ to the left of the cursor. This line is pasted into this window’s input queue. Thus you have a crude command history (made up by the visible window and its scrollback buffer).

info

Uses the message line to display some information about the current window: the cursor position in the form ‘‘(column,row)’’ starting with ‘‘(1,1)’’, the terminal width and height plus the size of the scrollback buffer in lines, like in ’’(80,24)+50’’, various flag settings (flow-control, insert mode, origin mode, wrap mode, application-keypad mode, output logging, activity monitoring and redraw (‘+’ indicates enabled, ‘–’ not)), the currently active character set (*G0*, *G1*, *G2*, or *G3*), and in square brackets the terminal character sets that are currently designated as *G0* through *G3*. For system information use the

''time'' command.

ins_reg [*key*]

No longer exists, use ''paste'' instead.

kanji jis|euc|sjis [**jis|euc|sjis**]

Tell screen how to process kanji input/output. The first argument sets the kanji type of the current window. Each window can emulate a different type. The optional second parameter tells screen how to write the kanji codes to the connected terminal. The preferred method of setting the display type is to use the ''KJ'' termcap entry. See also ''defkanji'', which changes the default setting of a new window.

kill

Kill current window.
If there is an 'exec' command running then it is killed. Otherwise the process (shell) running in the window receives a HANGUP condition, the window structure is removed and *screen* switches to the previously displayed window. When the last window is destroyed, *screen* exits. Note: *Emacs* users should keep this command in mind, when killing a line. It is recommended not to use ''C-a'' as the *screen* escape key or to rebind kill to ''C-a K''.

lastmsg

Redisplay the last contents of the message/status line. Useful if you're typing when a message appears, because the message goes away when you press a key (unless your terminal has a hardware status line). Refer to the commands ''msgwait'' and ''msgminwait'' for fine tuning.

license

Display the disclaimer page. This is done whenever *screen* is started without options, which should be often enough. See also the ''startup_message'' command.

lockscreen

Lock this display. Call a screenlock program (/local/bin/lck or /usr/bin/lock or a builtin if no other is available). Screen does not accept any command keys until this program terminates. Meanwhile processes in the windows may continue, as the windows are in the 'detached' state. The screenlock program may be changed through the environment variable $LOCKPRG (which must be set in the shell from which *screen* is started) and is executed with the user's uid and gid.

log [**on|off**]

Start/stop writing output of the current window to a file ''screenlog.*n*'' in the window's default directory, where *n* is the number of the current window. If no parameter is given, the state of logging is toggled. The session log is appended to the previous contents of the file if it already exists. The current contents and the contents of the scrollback history are not included in the session log. Default is 'off'.

logdir *directory*

Defines a directory where logfiles will be placed. If unset logfiles are written in *screen's* current working directory.

login [**on|off**]

Adds or removes the entry in the utmp database file for the current window. This controls if the window is 'logged in'. When no parameter is given, the login state of the window is toggled. Additionally to that toggle, it is convenient having a 'log in' and a 'log out' key. E. g. 'bind I login on' and 'bind O login off' will map these keys to be C-a I and C-a O. The default setting (in config.h.in) should be

''on'' for a *screen* that runs under suid-root. Use the ''deflogin'' command to change the default login state for new windows. Both commands are only present when *screen* has been compiled with utmp support.

mapdefault

Tell screen that the next input character should only be looked up in the default bindkey table. See also ''bindkey''.

mapnotnext

Like mapdefault, but don't even look in the default bindkey table.

maptimeout [*timo*]

Set the intercharacter timer for input sequence detection to a timeout of *timo* ms. The default timeout is 300ms. Maptimeout with no arguments shows the current setting. See also ''bindkey''.

markkeys *string*

This is a method of changing the keymap used for copy/history mode. The string is made up of *oldchar=newchar* pairs which are separated by ':'. Example: The string ''B=^B:F=^F'' will change the keys 'C-b' and 'C-f' to the vi style binding (scroll up/down fill page). This happens to be the default binding for 'B' and 'F'. The command ''markkeys h=^B:l=^F:$=^E'' would set the mode for an emacs-style binding.

meta

Insert the command character (C-a) in the current window's input stream.

monitor [**on**|**off**]

Toggles activity monitoring of windows. When monitoring is turned on and an affected window is switched into the background, you will receive the activity notification message in the status line at the first sign of output and the window will also be marked with an '@' in the window-status display. Monitoring is initially off for all windows.

msgminwait *sec*

Defines the time *screen* delays a new message when one message is currently displayed. The default is 1 second.

msgwait *sec*

Defines the time a message is displayed if *screen* is not disturbed by other activity. The default is 5 seconds.

multiuser on|**off**

Switch between singleuser and multiuser mode. Standard *screen* operation is singleuser. In multiuser mode the commands 'acladd', 'aclchg' and 'acldel' can be used to enable (and disable) other users accessing this screen.

nethack on|**off**

Changes the kind of error messages used by *screen*. When you are familiar with the game ''nethack'', you may enjoy the nethack-style messages which will often blur the facts a little, but are much funnier to read. Anyway, standard messages often tend to be unclear as well.
This option is only available if *screen* was compiled with the NETHACK flag defined. The default setting is then determined by the presence of the environment variable $NETHACKOPTIONS.

next

Switch to the next window. This command can be used repeatedly to cycle through the list of windows.

number [*n*]

Change the current windows number. If the given number *n* is already used by another window, both windows exchange their numbers. If no argument is specified, the current window number (and title) is shown.

obuflimit [*limit*]

If the output buffer contains more bytes than the specified limit, no more data will be read from the windows. The default value is 256. If you have a fast display (like xterm), you can set it to some higher value. If no argument is specified, the current setting is displayed.

other

Switch to the window displayed previously.

partial on|off

Defines whether the display should be refreshed (as with *redisplay*) after switching to the current window. This command only affects the current window. To immediately affect all windows use the *allpartial* command. Default is 'off', of course. This default is fixed, as there is currently no *defpartial* command.

password [*crypted_pw*]

Present a crypted password in your ''.screenrc'' file and *screen* will ask for it, whenever someone attempts to resume a detached. This is useful if you have privileged programs running under *screen* and you want to protect your session from reattach attempts by another user masquerading as your uid (i.e. any superuser.) If no crypted password is specified, *screen* prompts twice for typing a password and places its encryption in the paste buffer. Default is 'none', this disables password checking.

paste [*registers*[*dest_reg*]]

Write the (concatenated) contents of the specified registers to the stdin queue of the current window. The register '.' is treated as the paste buffer. If no parameter is given the user is prompted for a single register to paste. The paste buffer can be filled with the *copy*, *history* and *readbuf* commands. Other registers can be filled with the *register*, *readreg* and *paste* commands. If *paste* is called with a second argument, the contents of the specified registers is pasted into the named destination register rather than the window. If '.' is used as the second argument, the displays paste buffer is the destination. Note, that ''paste'' uses a wide variety of resources: Whenever a second argument is specified no current window is needed. When the source specification only contains registers (not the paste buffer) then there need not be a current display (terminal attached), as the registers are a global resource. The paste buffer exists once for every user.

pastefont [**on**|**off**]

Tell screen to include font information in the paste buffer. The default is not to do so. This command is especially usefull for multi character fonts like kanji.

pow_break

Reopen the window's terminal line and send a break condition. See 'break'.

pow_detach

Power detach. Mainly the same as *detach*, but also sends a HANGUP signal to the parent process of *screen*. CAUTION: This will result in a logout, when *screen* was started from your login shell.

pow_detach_msg *message*

The *message* specified here is output whenever a 'Power detach' was performed. It may be used as a replacement for a logout message or to reset baud rate, etc.

prev

Switch to the window with the next lower number. This command can be used repeatedly to cycle through the list of windows.

printcmd [*cmd*]

If *cmd* is not an empty string, screen will not use the terminal capabilities ''po/pf'' if it detects an ansi print sequence **ESC [5 i**, but pipe the output into *cmd*. This should normally be a command like ''lpr'' or '''cat > /tmp/scrprint'''. **printcmd** without a command displays the current setting. The ansi sequence **ESC ** ends printing and closes the pipe.
Warning: Be careful with this command! If other user have write access to your terminal, they will be able to fire off print commands.

process [*key*]

Stuff the contents of the specified register into *screen*'s input queue. If no argument is given you are prompted for a register name. The text is parsed as if it had been typed in from the user's keyboard. This command can be used to bind multiple actions to a single key.

quit

Kill all windows and terminate *screen*. Note that on VT100-style terminals the keys C-4 and C-\ are identical. This makes the default bindings dangerous: Be careful not to type C-a C-4 when selecting window no. 4. Use the empty bind command (as in ''bind '^\''') to remove a key binding.

readbuf

Reads the contents of the current screen-exchange file into the paste buffer. See also ''bufferfile'' command.

readreg [*register* [*filename*]]

Does one of two things, dependent on number of arguments: with zero or one arguments it it duplicates the paste buffer contents into the register specified or entered at the prompt. With two arguments it reads the contents of the named file into the register, just as *readbuf* reads the screen-exchange file into the paste buffer. The following example will paste the system's password file into the screen window (using register p, where a copy remains):

C-a : readreg p /etc/passwd
C-a : paste p

redisplay

Redisplay the current window. Needed to get a full redisplay when in partial redraw mode.

register *key string*

Save the specified *string* to the register *key*. See also the ‘‘paste’’ command.

removebuf

Unlinks the screen-exchange file used by the commands ‘‘writebuf’’ and ‘‘readbuf’’.

reset

Reset the virtual terminal to its ‘‘power-on’’ values. Useful when strange settings (like scroll regions or graphics character set) are left over from an application.

screen [*-opts*] [*n*] [*cmd* [*args*]]

Establish a new window. The flow-control options (**–f**, **–fn** and **–fa**), title (a. k. a.) option (**–t**), login options (**-l** and **-ln**) , terminal type option (**-T <term>**) and scrollback option (**-h** <num>) may be specified for each command. If an optional number *n* in the range 0..9 is given, the window number *n* is assigned to the newly created window (or, if this number is already in-use, the next available number). If a command is specified after ‘‘screen’’, this command (with the given arguments) is started in the window; otherwise, a shell is created. Thus, if your ‘‘.screenrc’’ contains the lines

```
# example for .screenrc:
screen 1
screen -fn -t foobar 2 telnet foobar
```

screen creates a shell window (in window #1) and a window with a TELNET connection to the machine foobar (with no flow-control using the title ‘‘foobar’’ in window #2). Note, that unlike previous versions of *screen* no additional default window is created when ‘‘screen’’ commands are included in your ‘‘.screenrc’’ file. When the initialization is completed, *screen* switches to the last window specified in your .screenrc file or, if none, opens a default window #0.

scrollback *num*

Set the size of the scrollback buffer for the current windows to *num* lines. The default scrollback is 100 lines. See also the ‘‘defscrollback’’ command and use ‘‘C-a i’’ to view the current setting.

select [*n*]

Switch to the window with the number *n*. If no window number is specified, you get prompted for an identifier. This can be title (alphanumeric window name) or a number. When a new window is established, the first available number is assigned to this window. Thus, the first window can be activated by ‘‘select 0’’ (there can be no more than 10 windows present simultaneously unless *screen* is compiled with a higher MAXWIN setting).

sessionname [*name*]

Rename the current session. Note, that for ‘‘screen -list’’ the name shows up with the process-id prepended. If the argument ‘‘name’’ is omitted, the name of this session is displayed. Caution: The $STY environment variables still reflects the old name. This may result in confusion. The default is constructed from the tty and host names.

setenv [*var* [*string*]]

Set the environment variable *var* to value *string*. If only *var* is specified, the user will be prompted to enter a value. If no parameters are specified, the user will be prompted for both variable and value. The environment is inherited by all subsequently forked shells.

shell *command*

Set the command to be used to create a new shell. This overrides the value of the environment variable $SHELL. This is useful if you'd like to run a tty-enhancer which is expecting to execute the program specified in $SHELL. If the command begins with a '-' character, the shell will be started as a login-shell.

shelltitle *title*

Set the title for all shells created during startup or by the C-A C-c command. For details about what a title is, see the discussion entitled ''TITLES (naming windows)''.

silence [**on**|**off**|*sec*]

Toggles silence monitoring of windows. When silence is turned on and an affected window is switched into the background, you will receive the silence notification message in the status line after a specified period of inactivity (silence). The default timeout can be changed with the 'silencewait' command or by specifying a number of seconds instead of 'on' or 'off'. Silence is initially off for all windows.

silencewait *sec*

Define the time that all windows monitored for silence should wait before displaying a message. Default 30 seconds.

sleep *num*

This command will pause the execution of a .screenrc file for *num* seconds. Keyboard activity will end the sleep. It may be used to give users a chance to read the messages output by ''echo''.

slowpaste *usec*

Define the speed at which text is inserted by the paste ("C-a]") command. If the slowpaste value is nonzero text is written character by character. *screen* will make a pause of *usec* milliseconds after each single character write to allow the application to process its input. Only use slowpaste if your underlying system exposes flow control problems while pasting large amounts of text.

startup_message on|off

Select whether you want to see the copyright notice during startup. Default is 'on', as you probably noticed.

stuff *string*

Stuff the string *string* in the input buffer of the current window. This is like the ''paste'' command but with much less overhead. You cannot paste large buffers with the tuff'' command. It is most useful for key bindings. See also ''bindkey''.

suspend

Suspend *screen*. The windows are in the 'detached' state, while *screen* is suspended. This feature relies on the shell being able to do job control.

term *term*

In each window's environment *screen* opens, the $TERM variable is set to ''screen'' by default. But when no description for ''screen'' is installed in the local termcap or terminfo data base, you set $TERM to – say – ''vt100''. This won't do much harm, as *screen* is VT100/ANSI compatible. The use of the ''term'' command is discouraged for non-default purpose. That is, one may want to specify special $TERM settings (e.g. vt100) for the next ''screen rlogin othermachine'' command. Use the command ''screen -T vt100 rlogin othermachine'' rather than setting (''term vt100'') and resetting (''term

screen'') the default before and after the ''screen'' command.

termcap *term terminal-tweaks* [*window-tweaks*]
terminfo *term terminal-tweaks* [*window-tweaks*]

Use this command to modify your terminal's termcap entry without going through all the hassles involved in creating a custom termcap entry. Plus, you can optionally customize the termcap generated for the windows. If your system works with terminfo-database rather than with termcap, *screen* will understand the 'terminfo' command, which has the same effects as the 'termcap' command. Thus users can write one .screenrc file that handles both cases, although terminfo syntax is slightly different from termcap syntax.

The first argument specifies which terminal(s) should be affected by this definition. You can specify multiple terminal names by separating them with '|'s. Use '*' to match all terminals and 'vt*' to match all terminals that begin with ''vt''.

Each *tweak* argument contains one or more termcap defines (separated by ':'s) to be inserted at the start of the appropriate termcap entry, enhancing it or overriding existing values. The first tweak modifies your terminal's termcap, and contains definitions that your terminal uses to perform certain functions. Specify a null string to leave this unchanged (e. g. ''). The second (optional) tweak modifies all the window termcaps, and should contain definitions that *screen* understands (see the ''VIRTUAL TERMINAL'' section).

Some examples:

```
termcap xterm* LP:hs@
```

Informs *screen* that all terminals that begin with 'xterm' have firm auto-margins that allow the last position on the screen to be updated (LP), but they don't really have a status line (no 'hs' – append '@' to turn entries off). Note that we assume 'LP' for all terminal names that start with ''vt'', but only if you don't specify a termcap command for that terminal.

```
termcap vt* LP termcap vt102|vt220 Z0=\E[?3h:Z1=\E[?3l
```

Specifies the firm-margined 'LP' capability for all terminals that begin with 'vt', and the second line will also add the escape-sequences to switch into (Z0) and back out of (Z1) 132-character-per-line mode if this is a VT102 or VT220. (You must specify Z0 and Z1 in your termcap to use the width-changing commands.)

```
termcap vt100 "" l0=PF1:l1=PF2:l2=PF3:l3=PF4
```

This leaves your vt100 termcap alone and adds the function key labels to each window's termcap entry.

```
termcap h19|z19 am@:im=\E@:ei=\EO dc=\E[P
```

Takes a h19 or z19 termcap and turns off auto-margins (am@) and enables the insert mode (im) and end-insert (ei) capabilities (the '@' in the 'im' string is after the '=', so it is part of the string). Having the 'im' and 'ei' definitions put into your terminal's termcap will cause *screen* to automatically advertise the character-insert capability in each window's termcap. Each window will also get the delete-character capability (dc) added to its termcap, which *screen* will translate into a line-update for the terminal (we're pretending it doesn't support character deletion).

If you would like to fully specify each window's termcap entry, you should instead set the $SCREENCAP variable prior to running *screen*. See the discussion on the ''VIRTUAL TERMINAL'' in this manual, and the termcap(5) man page for more information on termcap definitions.

time

Uses the message line to display the time of day, the host name, and the load averages over 1, 5, and 15 minutes (if this is available on your system). For window specific information use ''info''.

title [*windowalias*]

Set the name of the current window to *windowalias*. If no name is specified, *screen* prompts for one. This command was known as ‘aka’ in previous releases.

unsetenv *var*

Unset an environment variable.

vbell on|off

If your terminal does not support a visual bell, a ‘vbell-message’ is displayed in the status line. Sets the visual bell setting for this window. If your terminal does not support a visual bell, a ‘vbell-message’ is displayed in the status line. Refer to the termcap variable ‘vb’ (terminfo: ’flash’).

vbell_msg *message*

Sets the visual bell message. *message* is printed to the status line if the window receives a bell character (^G) and vbell is set to ‘‘on’’. The default message is ‘‘Wuff, Wuff!!’’.

vbellwait *sec*

Define a delay in seconds after each display of *screen*’s visual bell message. The default is 1 second.

version

Print the current version and the compile date in the status line.

wall *message ...*

Write a message to all displays. The message will appear in the terminal’s status line.

width [*num*]

Toggle the window width between 80 and 132 columns or set it to *num* columns if an argument is specified. This requires a capable terminal and the termcap entries ‘‘Z0’’ and ‘‘Z1’’. See the ‘‘termcap’’ command for more information.

windows

Uses the message line to display a list of all the windows. Each window is listed by number with the name of process that has been started in the window (or its title); the current window is marked with a ‘*’; the previous window is marked with a ‘-’; all the windows that are ‘‘logged in’’ are marked with a ‘$’; a background window that has received a bell is marked with a ‘!’; a background window that is being monitored and has had activity occur is marked with an ‘@’; a window which has output logging turned on is marked with ‘(L)’; windows occupied by other users are marked with ‘&’; windows in the zombie state are marked with ‘Z’. If this list is too long to fit on the terminal’s status line only the portion around the current window is displayed.

wrap [**on|off**]

Sets the line-wrap setting for the current window. When line-wrap is on, the second consecutive printable character output at the last column of a line will wrap to the start of the following line. As an added feature, backspace (^H) will also wrap through the left margin to the previous line. Default is ‘on’.

writebuf

Writes the contents of the paste buffer to a public accessible screen-exchange file. This is thought of as a primitive means of communication between *screen* users on the same host. The filename can be set with the *bufferfile* command and defaults to ‘‘/tmp/screen-exchange’’.

writelock [**on**|**off**|**auto**]

In addition to access control lists, not all users may be able to write to the same window at once. Per default, writelock is in ‘auto’ mode and grants exclusive input permission to the user who is the first to switch to the particular window. When he leaves the window, other users may obtain the writelock (automatically). The writelock of the current window is disabled by the command ‘‘writelock off’’. If the user issues the command ‘‘writelock on’’ he keeps the exclusive write permission while switching to other windows.

xoff
xon

Insert a CTRL-s / CTRL-q character to the stdin queue of the current window.

zombie [*keys*]
defzombie [*keys*]

Per default *screen* windows are removed from the window list as soon as the windows process (e.g. shell) exits. When a string of two keys is specified to the zombie command, ‘dead’ windows will remain in the list. The **kill** kommand may be used to remove such a window. Pressing the first key in the dead window has the same effect. When pressing the second key, screen will attempt to resurrect the window. The process that was initially running in the window will be launched again. Calling **zombie** without parameters will clear the zombie setting, thus making windows disappear when their process exits.

As the zombie-setting is manipulated globally for all windows, this command should only be called **defzombie**. Until we need this as a per window setting, the commands **zombie** and **defzombie** are synonymous.

THE MESSAGE LINE

Screen displays informational messages and other diagnostics in a *message line*. While this line is distributed to appear at the bottom of the screen, it can be defined to appear at the top of the screen during compilation. If your terminal has a status line defined in its termcap, *screen* will use this for displaying its messages, otherwise a line of the current screen will be temporarily overwritten and output will be momentarily interrupted. The message line is automatically removed after a few seconds delay, but it can also be removed early (on terminals without a status line) by beginning to type.

The message line facility can be used by an application running in the current window by means of the ANSI *Privacy message* control sequence. For instance, from within the shell, try something like:

echo ’<esc>^Hello world from window ’$WINDOW’<esc>\\’

where ’<esc>’ is an *escape*, ’^’ is a literal up-arrow, and ’\\’ turns into a single backslash.

FLOW-CONTROL

Each window has a flow-control setting that determines how *screen* deals with the XON and XOFF characters (and perhaps the interrupt character). When flow-control is turned off, *screen* ignores the XON and XOFF characters, which allows the user to send them to the current program by simply typing them (useful for the *emacs* editor, for instance). The trade-off is that it will take longer for output from a ‘‘normal’’ program to pause in response to an XOFF. With flow-control turned on, XON and XOFF characters are used to immediately pause the output of the current window. You can still send

these characters to the current program, but you must use the appropriate two-character *screen* commands (typically ‘‘C-a q’’ (xon) and ‘‘C-a s’’ (xoff)). The xon/xoff commands are also useful for typing C-s and C-q past a terminal that intercepts these characters.

Each window has an initial flow-control value set with either the **–f** option or the ‘‘defflow’’ .screenrc command. Per default the windows are set to automatic flow-switching. It can then be toggled between the three states ’fixed on’, ’fixed off’ and

The automatic flow-switching mode deals with flow control using the TIOCPKT mode (like ‘‘rlogin’’ does). If the tty driver does not support TIOCPKT, *screen* tries to find out the right mode based on the current setting of the application keypad – when it is enabled, flow-control is turned off and visa versa. Of course, you can still manipulate flow-control manually when needed.

If you’re running with flow-control enabled and find that pressing the interrupt key (usually C-c) does not interrupt the display until another 6-8 lines have scrolled by, try running *screen* with the ‘‘interrupt’’ option (add the ‘‘interrupt’’ flag to the ‘‘flow’’ command in your .screenrc, or use the **–i** command-line option). This causes the output that *screen* has accumulated from the interrupted program to be flushed. One disadvantage is that the virtual terminal’s memory contains the non-flushed version of the output, which in rare cases can cause minor inaccuracies in the output. For example, if you switch screens and return, or update the screen with ‘‘C-a l’’ you would see the version of the output you would have gotten without ‘‘interrupt’’ being on. Also, you might need to turn off flow-control (or use auto-flow mode to turn it off automatically) when running a program that expects you to type the interrupt character as input, as it is possible to interrupt the output of the virtual terminal to your physical terminal when flow-control is enabled. If this happens, a simple refresh of the screen with ‘‘C-a l’’ will restore it. Give each mode a try, and use whichever mode you find more comfortable.

TITLES (naming windows)

You can customize each window’s name in the window display (viewed with the ‘‘windows’’ command (C-a w)) by setting it with one of the title commands. Normally the name displayed is the actual command name of the program created in the window. However, it is sometimes useful to distinguish various programs of the same name or to change the name on-the-fly to reflect the current state of the window.

The default name for all shell windows can be set with the ‘‘shelltitle’’ command in the .screenrc file, while all other windows are created with a ‘‘screen’’ command and thus can have their name set with the **–t** option. Interactively, there is the title-string escape-sequence (<esc>k*name*<esc>\) and the ‘‘title’’ command (C-a A). The former can be output from an application to control the window’s name under software control, and the latter will prompt for a name when typed. You can also bind pre-defined names to keys with the ‘‘title’’ command to set things quickly without prompting.

Finally, *screen* has a shell-specific heuristic that is enabled by setting the window’s name to ‘‘*search*|*name*’’ and arranging to have a null title escape-sequence output as a part of your prompt. The *search* portion specifies an end-of-prompt search string, while the *name* portion specifies the default shell name for the window. If the *name* ends in a ‘:’ *screen* will add what it believes to be the current command running in the window to the end of the window’s shell name (e. g. ‘‘*name:cmd*’’). Otherwise the current command name supersedes the shell name while it is running.

Here’s how it works: you must modify your shell prompt to output a null title-escape-sequence (<esc>k<esc>\) as a part of your prompt. The last part of your prompt must be the same as the string you specified for the *search* portion of the title. Once this is set up, *screen* will use the title-escape-sequence to clear the previous command name and get ready for the next command. Then, when a newline is received from the shell, a search is made for the end of the prompt. If found, it will grab the first word after the matched string and use it as the command name. If the command name begins with either ’!’, ’%’, or ’^’ *screen* will use the first word on the following line (if found) in preference to the just-found name. This helps csh users get better command names when using job control or history recall commands.

Here's some .screenrc examples:

```
screen -t top 2 nice top
```

Adding this line to your .screenrc would start a nice-d version of the ''top'' command in window 2 name ''top'' rather than ''nice''.

```
shelltitle '> |csh'
screen 1
```

These commands would start a shell with the given shelltitle. The title specified is an auto-title that would expect the prompt and the typed command to look something like the following:

```
/usr/joe/src/dir> trn
```

(it looks after the '> ' for the command name). The window status would show the name ''trn'' while the command was running, and revert to ''csh'' upon completion.

```
bind R screen -t '% |root:' su
```

Having this command in your .screenrc would bind the key sequence ''C-a R'' to the ''su'' command and give it an auto-title name of ''root:''. For this auto-title to work, the screen could look something like this:

```
% !em
emacs file.c
```

Here the user typed the csh history command ''!em'' which ran the previously entered ''emacs'' command. The window status would show ''root:emacs'' during the execution of the command, and revert to simply ''root:'' at its completion.

```
bind o title
bind E title ""
bind u title (unknown)
```

The first binding doesn't have any arguments, so it would prompt you for a title. when you type ''C-a o''. The second binding would clear an auto-title's current setting (C-a E). The third binding would set the current window's title to ''(unknown)'' (C-a u).

One thing to keep in mind when adding a null title-escape-sequence to your prompt is that some shells (like the csh) count all the non-control characters as part of the prompt's length. If these invisible characters aren't a multiple of 8 then backspacing over a tab will result in an incorrect display. One way to get around this is to use a prompt like this:

```
set prompt='^[[0000m^[k^[\% '
```

The escape-sequence ''<esc>[0000m'' not only normalizes the character attributes, but all the zeros round the length of the invisible characters up to 8. Bash users will probably want to echo the escape sequence in the PROMPT_COMMAND:

```
PROMPT_COMMAND='echo -n -e "\033k\033\134"'
```

(I used ''134'' to output a '\' because of a bug in bash v1.04).

THE VIRTUAL TERMINAL

Usually *screen* tries to emulate as much of the VT100/ANSI standard as possible. But if your terminal lacks certain capabilities, the emulation may not be complete. In these cases *screen* has to tell the applications that some of the features are missing. This is no problem on machines using termcap, because *screen* can use the $TERMCAP variable to customize the standard *screen* termcap.

But if you do a rlogin on another machine or your machine supports only terminfo this method fails. Because of this, *screen* offers a way to deal with these cases. Here is how it works:

When *screen* tries to figure out a terminal name for itself, it first looks for an entry named ''screen.<term>'', where <term> is the contents of your $TERM variable. If no such entry exists, *screen* tries ''screen'' (or ''screen-w'' if the terminal is wide (132 cols or more)). If even this entry cannot be found, ''vt100'' is used as a substitute.

The idea is that if you have a terminal which doesn't support an important feature (e.g. delete char or clear to EOS) you can build a new termcap/terminfo entry for *screen* (named ''screen.<dumbterm>'') in which this capability has been disabled. If this entry is installed on your machines you are able to do a rlogin and still keep the correct termcap/terminfo entry. The terminal name is put in the $TERM variable of all new windows. *Screen* also sets the $TERMCAP variable reflecting the capabilities of the virtual terminal emulated. Notice that, however, on machines using the terminfo database this variable has no effect. Furthermore, the variable $WINDOW is set to the window number of each window.

The actual set of capabilities supported by the virtual terminal depends on the capabilities supported by the physical terminal. If, for instance, the physical terminal does not support underscore mode, *screen* does not put the 'us' and 'ue' capabilities into the window's $TERMCAP variable, accordingly. However, a minimum number of capabilities must be supported by a terminal in order to run *screen*; namely scrolling, clear screen, and direct cursor addressing (in addition, *screen* does not run on hardcopy terminals or on terminals that over-strike).

Also, you can customize the $TERMCAP value used by *screen* by using the ''termcap'' .screenrc command, or by defining the variable $SCREENCAP prior to startup. When the is latter defined, its value will be copied verbatim into each window's $TERMCAP variable. This can either be the full terminal definition, or a filename where the terminal ''screen'' (and/or ''screen-w'') is defined.

Note that *screen* honors the ''terminfo'' .screenrc command if the system uses the terminfo database rather than termcap.

When the boolean 'G0' capability is present in the termcap entry for the terminal on which *screen* has been called, the terminal emulation of *screen* supports multiple character sets. This allows an application to make use of, for instance, the VT100 graphics character set or national character sets. The following control functions from ISO 2022 are supported: *lock shift G0* (*SI*), *lock shift G1* (*SO*), *lock shift G2*, *lock shift G3*, *single shift G2*, and *single shift G3*. When a virtual terminal is created or reset, the ASCII character set is designated as *G0* through *G3*. When the 'G0' capability is present, *screen* evaluates the capabilities 'S0', 'E0', and 'C0' if present. 'S0' is the sequence the terminal uses to enable and start the graphics character set rather than *SI*. 'E0' is the corresponding replacement for *SO*. 'C0' gives a character by character translation string that is used during semi-graphics mode. This string is built like the 'acsc' terminfo capability.

When the 'po' and 'pf' capabilities are present in the terminal's termcap entry, applications running in a *screen* window can send output to the printer port of the terminal. This allows a user to have an application in one window sending output to a printer connected to the terminal, while all other windows are still active (the printer port is enabled and disabled again for each chunk of output). As a side-effect, programs running in different windows can send output to the printer simultaneously. Data sent to the printer is not displayed in the window.

Some capabilities are only put into the $TERMCAP variable of the virtual terminal if they can be efficiently implemented by the physical terminal. For instance, 'dl' (delete line) is only put into the $TERMCAP variable if the terminal supports either delete line itself or scrolling regions. Note that this may provoke confusion, when the session is reattached on a different terminal, as the value of $TERMCAP cannot be modified by parent processes.

The following is a list of control sequences recognized by *screen*. ''(V)'' and ''(A)'' indicate VT100-specific and ANSI- or ISO-specific functions, respectively.

ESC E		Next Line
ESC D		Index
ESC M		Reverse Index
ESC H		Horizontal Tab Set
ESC Z		Send VT100 Identification String
ESC 7	(V)	Save Cursor and Attributes
ESC 8	(V)	Restore Cursor and Attributes
ESC [s	(A)	Save Cursor and Attributes
ESC [u	(A)	Restore Cursor and Attributes
ESC c		Reset to Initial State
ESC =	(V)	Application Keypad Mode
ESC >	(V)	Numeric Keypad Mode
ESC # 8	(V)	Fill Screen with E's
**ESC **	(A)	String Terminator
ESC ^	(A)	Privacy Message String (Message Line)
ESC !		Global Message String (Message Line)
ESC k		A. k. a. Definition String
ESC P	(A)	Device Control String. Outputs a string directly to the host terminal without interpretation.
ESC _	(A)	Application Program Command (not used)
ESC]	(A)	Operating System Command (not used)
Control-N	(A)	Lock Shift G1 (SO)
Control-O	(A)	Lock Shift G0 (SI)
ESC n	(A)	Lock Shift G2
ESC o	(A)	Lock Shift G3
ESC N	(A)	Single Shift G2
ESC O	(A)	Single Shift G3
ESC (Pcs	(A)	Designate character set as G0
ESC) Pcs	(A)	Designate character set as G1
ESC * Pcs	(A)	Designate character set as G2
ESC + Pcs	(A)	Designate character set as G3
ESC [Pn **;** Pn **H**		Direct Cursor Addressing
ESC [Pn **;** Pn **f**		Direct Cursor Addressing
ESC [Pn **J**		Erase in Display
Pn = None or **0**		>From Cursor to End of Screen
1		>From Beginning of Screen to Cursor
2		Entire Screen
ESC [Pn **K**		Erase in Line
Pn = None or **0**		>From Cursor to End of Line

1		>From Beginning of Line to Cursor
2		Entire Line
ESC [Pn **A**		Cursor Up
ESC [Pn **B**		Cursor Down
ESC [Pn **C**		Cursor Right
ESC [Pn **D**		Cursor Left
ESC [Ps **;...;** Ps **m**		Select Graphic Rendition
Ps = None or **0**		Default Rendition
1		Bold
2	(A)	Faint
3	(A)	*Standout* Mode (ANSI: Italicized)
4		Underlined
5		Blinking
7		Negative Image
22	(A)	Normal Intensity
23	(A)	*Standout* Mode off (ANSI: Italicized off)
24	(A)	Not Underlined
25	(A)	Not Blinking
27	(A)	Positive Image
ESC [Pn **g**		Tab Clear
Pn = None or **0**		Clear Tab at Current Position
3		Clear All Tabs
ESC [Pn **;** Pn **r**	(V)	Set Scrolling Region
ESC [Pn **I**	(A)	Horizontal Tab
ESC [Pn **Z**	(A)	Backward Tab
ESC [Pn **L**	(A)	Insert Line
ESC [Pn **M**	(A)	Delete Line
ESC [Pn **@**	(A)	Insert Character
ESC [Pn **P**	(A)	Delete Character
ESC [Ps **;...;** Ps **h**		Set Mode
ESC [Ps **;...;** Ps **l**		Reset Mode
Ps = **4**	(A)	Insert Mode
?1	(V)	Application Cursor Keys
?3	(V)	Change Terminal Width to 132 columns
?5	(V)	Visible Bell (*On* followed by *Off*)
?6	(V)	*Origin* Mode
?7	(V)	*Wrap* Mode
ESC [5 i	(A)	Start relay to printer (ANSI Media Copy)
ESC [4 i	(A)	Stop relay to printer (ANSI Media Copy)

ESC [8 ; Ph **;** Pw **t**	Resize the window to 'Ph' lines and 'Pw' columns (SunView special)
ESC [c	Send VT100 Identification String
ESC [> c	Send VT220 Secondary Device Attributes String
ESC [6 n	Send Cursor Position Report

INPUT TRANSLATION

In order to do a full VT100 emulation *screen* has to detect that a sequence of characters in the input stream was generated by a keypress on the user's keyboard and insert the VT100 style escape sequence. *Screen* has a very flexible way of doing this by making it posible to map arbitrary commands on arbitrary sequences of characters. For standard VT100 emulation the command will always insert a string in the input buffer of the window (see also command **stuff** in the command table). Because the sequences generated by a keypress can change after a reattach from a different terminal type, it is possible to bind commands to the termcap name of the keys. *Screen* will insert the correct binding after each reattach. See the **bindkey** command for further details on the syntax and examples.

Here is the table of the default key bindings. (A) means that the command is executed if the keyboard is switched into application mode.

Key name	Termcap name	Command	
Cursor up	ku	stuff \033[A	
		stuff \033OA	(A)
Cursor down	kd	stuff \033[B	
		stuff \033OB	(A)
Cursor right	kr	stuff \033[C	
		stuff \033OC	(A)
Cursor left	kl	stuff \033[D	
		stuff \033OD	(A)
Function key 0	k0	stuff \033[10~	
Function key 1	k1	stuff \033OP	
Function key 2	k2	stuff \033OQ	
Function key 3	k3	stuff \033OR	
Function key 4	k4	stuff \033OS	
Function key 5	k5	stuff \033[15~	
Function key 6	k6	stuff \033[17~	
Function key 7	k7	stuff \033[18~	
Function key 8	k8	stuff \033[19~	
Function key 9	k9	stuff \033[20~	
Function key 10	k;	stuff \033[21~	
Function key 11	F1	stuff \033[22~	
Function key 12	F2	stuff \033[23~	
Backspace	kb	stuff \010	
Home	kh	stuff \033[1~	
End	kH	stuff \033[4~	
Insert	kI	stuff \033[2~	
Delete	kD	stuff \033[3~	
Page up	kP	stuff \033[5~	
Page down	kN	stuff \033[6~	
Keypad 0	f0	stuff 0	
		stuff \033Op	(A)
Keypad 1	f1	stuff 1	
		stuff \033Oq	(A)

Keypad 2	f2	stuff 2	
		stuff \033Or	(A)
Keypad 3	f3	stuff 3	
		stuff \033Os	(A)
Keypad 4	f4	stuff 4	
		stuff \033Ot	(A)
Keypad 5	f5	stuff 5	
		stuff \033Ou	(A)
Keypad 6	f6	stuff 6	
		stuff \033Ov	(A)
Keypad 7	f7	stuff 7	
		stuff \033Ow	(A)
Keypad 8	f8	stuff 8	
		stuff \033Ox	(A)
Keypad 9	f9	stuff 9	
		stuff \033Oy	(A)
Keypad +	f+	stuff +	
		stuff \033Ok	(A)
Keypad -	f-	stuff -	
		stuff \033Om	(A)
Keypad *	f*	stuff *	
		stuff \033Oj	(A)
Keypad /	f/	stuff /	
		stuff \033Oo	(A)
Keypad =	fq	stuff =	
		stuff \033OX	(A)
Keypad .	f.	stuff .	
		stuff \033On	(A)
Keypad ,	f,	stuff ,	
		stuff \033Ol	(A)
Keypad enter	fe	stuff \015	
		stuff \033OM	(A)

SPECIAL TERMINAL CAPABILITIES

The following table describes all terminal capabilities that are recognized by *screen* and are not in the termcap(5) manual.

LP *(bool)* Terminal has VT100 style margins (‘magic margins’). Note that this capability is obsolete because *screen* uses ’xn’ instead.

Z0 *(str)* Change width to 132 columns.

Z1 *(str)* Change width to 80 columns.

WS *(str)* Resize display. This capability has the desired width and height as arguments. *SunView(tm)* example: ’\E[8;%d;%dt’.

B8 *(str)* Tell *screen* to look out for characters with 8th bit set. If such a character is found *screen* processes the specified string and than outputs the character with the 8th bit stripped off. Note that the string can contain any esc-sequences known to *screen*, too. (Example: Single Shift G2 = \EN.)

OP *(bool)* Don’t do a full VT100 style margin emulation. Same as the -O option.

NF *(bool)* Terminal doesn’t need flow control. Send ^S and ^Q direct to the application. Same as ’flow off’. The opposite of this capability is ’nx’.

G0 *(bool)* Terminal can deal with ISO 2022 font selection sequences.

S0 *(str)* Switch charset 'G0' to the specified charset. Default is '\E(%d'.

E0 *(str)* Switch charset 'G0' back to standard charset. Default is '\E(B'.

C0 *(str)* Use the string as a conversion table for font '0'. See the 'ac' capability for more details.

CS *(str)* Switch cursorkeys to application mode.

CE *(str)* Switch cursorkeys back to normal mode.

AN *(bool)* Turn on autonuke. See the 'autonuke' command for more details.

OL *(num)* Set the output buffer limit. See the 'obuflimit' command for more details.

KJ *(str)* Set the kanji type of the terminal. Valid strings are ''jis'', ''euc'' and ''sjis''.

ENVIRONMENT

COLUMNS	Number of columns on the terminal (overrides termcap entry).
HOME	Directory in which to look for .screenrc.
ISCREENRC	Alternate user screenrc file.
LINES	Number of lines on the terminal (overrides termcap entry).
LOCKPRG	Screen lock program.
NETHACKOPTIONS	Turns on nethack option.
PATH	Used for locating programs to run.
SCREENCAP	For customizing a terminal's TERMCAP value.
SCREENDIR	Alternate socket directory.
SCREENRC	Alternate user screenrc file.
SHELL	Default shell program for opening windows (default ''/bin/sh'').
STY	Alternate socket name.
SYSSCREENRC	Alternate system screenrc file.
TERM	Terminal name.
TERMCAP	Terminal description.

FILES

$SYSSCREENRC /local/etc/screenrc	*screen* initialization commands
$ISCREENRC $SCREENRC $HOME/.iscreenrc $HOME/.screenrc	Read in after /local/etc/screenrc
$ISCREENDIR/S-<login> $SCREENDIR/S-<login> /local/screens/S-<login>	Socket directories (default)
/usr/tmp/screens/S-<login>	Alternate socket directories.
<socket directory>/.termcap	Written by the "termcap" output function
/usr/tmp/screens/screen-exchange /tmp/screen-exchange	or *screen* 'interprocess communication buffer'
hardcopy.[0-9]	Screen images created by the hardcopy function
screenlog.[0-9]	Output log files created by the log function
/usr/lib/terminfo/?/* /etc/termcap	or Terminal capability databases
/etc/utmp	Login records
$LOCKPRG	Program that locks a terminal.

SEE ALSO

termcap(5), utmp(5), vi(1), captoinfo(1), tic(1)

AUTHORS

Originally created by Oliver Laumann, this latest version was produced by Wayne Davison, Juergen Weigert and Michael Schroeder.

COPYLEFT

CONTRIBUTORS

Ken Beal (kbeal@amber.ssd.csd.harris.com),
Rudolf Koenig (rfkoenig@immd4.informatik.uni-erlangen.de),
Toerless Eckert (eckert@immd4.informatik.uni-erlangen.de),
Wayne Davison (davison@borland.com),
Patrick Wolfe (pat@kai.com, kailand!pat),
Bart Schaefer (schaefer@cse.ogi.edu),
Nathan Glasser (nathan@brokaw.lcs.mit.edu),
Larry W. Virden (lvirden@cas.org),
Howard Chu (hyc@hanauma.jpl.nasa.gov),
Tim MacKenzie (tym@dibbler.cs.monash.edu.au),
Markku Jarvinen (mta@{cc,cs,ee}.tut.fi),
Marc Boucher (marc@CAM.ORG),
Doug Siebert (dsiebert@isca.uiowa.edu),
Ken Stillson (stillson@tsfsrv.mitre.org),
Ian Frechett (frechett@spot.Colorado.EDU),
Brian Koehmstedt (bpk@gnu.ai.mit.edu),
Don Smith (djs6015@ultb.isc.rit.edu),
Frank van der Linden (vdlinden@fwi.uva.nl),
Martin Schweikert (schweik@cpp.ob.open.de),
David Vrona (dave@sashimi.lcu.com),
E. Tye McQueen (tye%spillman.UUCP@uunet.uu.net),
Matthew Green (mrgreen@mame.mu.oz.au),
Christopher Williams (cgw@unt.edu),
Matt Mosley (mattm@access.digex.net),
Gregory Neil Shapiro (gshapiro@wpi.WPI.EDU).

VERSION

This is version 3.6.0. Its roots are a merge of a custom version 2.3PR7 by Wayne Davison and several

enhancements to Oliver Laumann's version 2.0. Note that all versions numbered 2.x are copyright by Oliver Laumann.

BUGS

- 'dm' (delete mode) and 'xs' are not handled correctly (they are ignored). 'xn' is treated as a magic-margin indicator.
- *Screen* has no clue about double-high or double-wide characters. But this is the only area where *vttest* is allowed to fail.
- *Screen* does not support color, although this has repeatedly been asked for.
- It is not possible to change the environment variable $TERMCAP when reattaching under a different terminal type.
- The support of terminfo based systems is very limited. Adding extra capabilities to $TERMCAP may not have any effects.
- *Screen* does not make use of hardware tabs.
- *Screen* must be installed as set-uid with owner root in order to be able to correctly change the owner of the tty device file for each window. Special permission may also be required to write the file ''/etc/utmp''.
- Entries in ''/etc/utmp'' are not removed when *screen* is killed with SIGKILL. This will cause some programs (like "w" or "rwho") to advertise that a user is logged on who really isn't.
- *Screen* may give a strange warning when your tty has no utmp entry.
- When the modem line was hung up, *screen* may not automatically detach (or quit) unless the device driver is configured to send a HANGUP signal. To detach a *screen* session use the -D or -d command line option.
- A weird imagination is most useful to gain full advantage of all the features.
- Send bugreports, fixes, enhancements, t-shirts, money, beer & pizza to **screen@uni-erlangen.de**.