NAME

screen − screen manager with VT100/ANSI terminal emulation

SYNOPSIS

screen [**−***options*] [*cmd* [*args*]]
screen −r [[*pid***.**]*tty*[**.***host*]]
screen −r *sessionowner***/**[[*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 ISO 6429 (ECMA 48, ANSI X3.64) 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. All windows run their programs completely independent of each other. Programs continue to run when their window is currently not visible and even when the whole *screen* session is detached from the user's terminal. 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. Shells usually distinguish between running as login-shell or sub-shell. Screen runs them as sub-shells, unless told otherwise (See ''shell'' .screenrc command).

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.

Screen does not understand the prefix ''C-'' to mean control, although this notation is used in this manual for readability. Please use the caret notation (''^A'' instead of ''C-a'') as arguments to e.g. the *escape* command or the *-e* option. *Screen* will also print out control characters in caret notation.

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. - Note that you cannot transport environment variables from the invoking shell to the application (emacs in this case), because it is forked from the parent screen process, not from the invoking shell.

If ''/run/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 your terminal is a ''true'' auto-margin terminal (it doesn't allow the last position on the screen to be updated without scrolling the screen) consider using 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. Most terminals nowadays have ''magic'' margins (automatic margins plus usable last column). This is the VT100 style type and perfectly suited for *screen*. 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.

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. In combination with the **−r/−R** option more powerful effects can be achieved:

−d −r Reattach a session and if necessary detach it first.

−d −R Reattach a session and if necessary detach or even create it first.

−d −RR Reattach a session and if necessary detach or create it. Use the first session if more than one session is available.

−D −r Reattach a session. If necessary detach and logout remotely first.

−D −R Attach here and now. In detail this means: If a session is running, then reattach. If necessary detach and logout remotely first. If it was not running create it and notify the user. This is the author's favorite.

−D −RR
Attach here and now. Whatever that means, just do it.

Note: It is always a good idea to check the status of your sessions 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 multi-user 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 /run/utmp updating). This can also be defined through the ‘‘deflogin’’ .screenrc command.

–ls [*match*]
–list [*match*]
does not start *screen*, but prints a list of *pid.tty.host* strings and creation timestamps identifying your *screen* sessions. Sessions marked ‘detached’ can be resumed with ‘‘screen –r’’. Those marked ‘attached’ are running and have a controlling terminal. If the session runs in multiuser mode, it is marked ‘multi’. Sessions marked as ‘unreachable’ either live on a different host or are ‘dead’. An unreachable session is considered dead, when its name matches either the name of the local host, or the specified parameter, if any. See the **-r** flag for a description how to construct matches. 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* to turn on automatic output logging for the windows.

–Logfile *file*
By default logfile name is ‘‘screenlog.0‘‘. You can set new logfile name with the ‘‘-Logfile‘‘ option.

–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. This flag has a special meaning in connection with the ‘–d’ option:

–d –m Start *screen* in ‘‘detached’’ mode. This creates a new session but doesn’t attach to it. This is useful for system startup scripts.

–D –m This also starts screen in ‘‘detached’’ mode, but doesn’t fork a new process. The command exits if the session terminates.

–O selects an 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.

–p *number_or_name*|-|=|+
Preselect a window. This is useful when you want to reattach to a specific window or you want to send a command via the ‘‘-X’’ option to a specific window. As with screen’s select command, ‘‘-’’ selects the blank window. As a special case for reattach, ‘‘=’’ brings up the windowlist on the blank window, while a ‘‘+’’ will create a new window. The command will not be executed if the specified window could not be found.

–q Suppress printing of error messages. In combination with ‘‘-ls’’ the exit value is as follows: 9 indicates a directory without sessions. 10 indicates a directory with running but not attachable sessions. 11 (or more) indicates 1 (or more) usable sessions. In combination with ‘‘-r’’ the exit value is as follows: 10 indicates that there is no session to resume. 12 (or more) indicates that there are 2 (or more) sessions to resume and you should specify which one to choose. In all other cases ‘‘-q’’ has no effect.

−Q Some commands now can be queried from a remote session using this flag, e.g. ‘‘screen −Q windows’’. The commands will send the response to the stdout of the querying process. If there was an error in the command, then the querying process will exit with a non-zero status.

The commands that can be queried now are:
echo
info
lastmsg
number
select
time
title
windows

−r [*pid.tty.host*]
−r *sessionowner/[pid.tty.host*]
resumes a detached *screen* session. No other options (except combinations with **−d/−D**) may be specified, though an optional prefix of [*pid.*]*tty.host* may be needed to distinguish between multiple detached *screen* sessions. The second form is used to connect to another user’s screen session which runs in multiuser mode. This indicates that screen should look for sessions in another user’s directory. This requires setuid-root.

−R resumes screen only when it’s unambiguous which one to attach, usually when only one *screen* is detached. Otherwise lists available sessions. **−RR** attempts to resume the youngest (in terms of creation time) 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 (actually screen uses ‘‘−xRR’’ in that case). For combinations with the **−d/−D** option see there. **Note:** Time-based session selection is a Debian addition.

−s *program*
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. See also there.

−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.

−T *term*
Set the $TERM environment variable using the specified term as opposed to the default setting of **screen**.

−U Run screen in UTF-8 mode. This option tells screen that your terminal sends and understands UTF-8 encoded characters. It also sets the default encoding for new windows to ‘utf8’.

−v Print version number.

−wipe [*match*]
does the same as ‘‘screen −ls’’, but removes destroyed sessions instead of marking them as ‘dead’. An unreachable session is considered dead, when its name matches either the name of the local host, or the explicitly given parameter, if any. See the **-r** flag for a description how to construct matches.

−x Attach to a not detached *screen* session. (Multi display mode). *Screen* refuses to attach from within itself. But when cascading multiple screens, loops are not detected; take care.

–X Send the specified command to a running screen session. You may use the **-S** option to specify the screen session if you have several screen sessions running. You can use the **-d** or **-r** option to tell screen to look only for attached or detached screen sessions. Note that this command doesn't work if the session is password protected.

–4 Resolve hostnames only to IPv4 addresses.

–6 Resolve hostnames only to IPv6 addresses.

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. The trailing commas in boxes with multiple keystroke entries are separators, not part of the bindings.

C-a '	(select)	Prompt for a window name or number to switch to.
C-a "	(windowlist -b)	Present a list of all windows for selection.
C-a *digit*	(select 0-9)	Switch to window number 0 – 9
C-a -	(select -)	Switch to window number 0 – 9, or to the blank window.
C-a tab	(focus)	Switch the input focus to the next region. See also *split, remove, only*.
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 command becomes '']]''.
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 F	(fit)	Resize the window to the current region size.
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 C-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 Q	(only)	Delete all regions but the current one. See also *split, remove, focus*.
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 S	(split)	Split the current region horizontally into two new ones. See also *only, remove, focus*.

C-a t, **C-a C-t**	(time)	Show system information.
C-a v	(version)	Display the version and compilation date.
C-a C-v	(digraph)	Enter digraph.
C-a w, **C-a C-w**	(windows)	Show a list of window.
C-a W	(width)	Toggle 80/132 columns.
C-a x or **C-a C-x**	(lockscreen)	Lock this terminal.
C-a X	(remove)	Kill the current region. See also *split, only, focus*.
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 **	(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 C-], **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.
C-a |	(split -v)	Split the current region vertically into two new ones.
C-a *	(displays)	Show a listing of all currently attached displays.

CUSTOMIZATION

The ‘‘socket directory’’ defaults either to $HOME/.screen or simply to /tmp/screens or preferably to /run/screen 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 700 directory in the environment variable $SCREENDIR.

When *screen* is invoked, it executes initialization commands from the files ‘‘/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 $SCREENRC, then $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.

Two configuration files are shipped as examples with your screen distribution: ‘‘etc/screenrc’’ and ‘‘etc/etc-screenrc’’. They contain a number of useful examples for various commands.

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* [*crypted-pw*]

addacl *usernames*

Enable users to fully access this screen session. *Usernames* can be one user or a comma separated 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. If an optional second parameter is supplied, it should be a crypted password for the named user(s). ‘Addacl’ is a synonym to ‘acladd’. Multi user mode only.

aclchg *usernames permbits list*

chacl *usernames permbits list*

Change permissions for a comma separated list of users. Permission bits are represented as ‘r’, ‘w’ and ‘x’. Prefixing ‘+’ grants the permission, ‘−’ removes it. The third parameter is a comma separated 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 read-only 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. Rights of the special username **nobody** cannot be changed (see the ‘‘su’’ command). ‘Chacl’ is a synonym to ‘aclchg’. 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.

aclgrp *username* [*groupname*]

Creates groups of users that share common access rights. The name of the group is the username of the group leader. Each member of the group inherits the permissions that are granted to the group leader. That means, if a user fails an access check, another check is made for the group leader. A user is removed from all groups the special value ‘‘none’’ is used for *groupname*. If the second parameter is omitted all groups

the user is in are listed.

aclumask [[*users*] *+bits* | [*users*] *-bits...*]

umask [[*users*] *+bits* | [*users*] *-bits...*]

This specifies the access other users have to windows that will be created by the caller of the command. *Users* may be no, one or a comma separated list of known usernames. If no users are specified, a list of all currently known users is assumed. *Bits* is any combination of access control bits allowed defined with the ''aclchg'' command. The special username ''?'' predefines the access that not yet known users will be granted to any window initially. The special username ''??'' predefines the access that not yet known users are granted to any command. Rights of the special username **nobody** cannot be changed (see the ''su'' command). 'Umask' is a synonym to 'aclumask'.

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 '^G' is replaced by the definition for bell in your termcap (usually an audible bell). The default message is

'Activity in window %n'

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 behavior of newly created windows.

altscreen on|off

If set to on, "alternate screen" support is enabled in virtual terminals, just like in xterm. Initial setting is 'off'.

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. Permission is checked for initiator of the ''at'' command, not for the owners of the affected display(s). Note that the '#' character works as a comment introducer when it is preceded by whitespace. This can be escaped by prefixing a '\'. Permission is checked for the initiator of the ''at'' command, not for the owners of the affected display(s).

Caveat: When matching against windows, the command is executed at least once per window. Commands that change the internal arrangement of windows (like ''other'') may be called again. In shared windows the command will be repeated for each attached display. Beware, when issuing toggle commands like ''login''! Some commands (e.g. ''process'') require that a display is associated with the target windows. These commands may not work correctly under ''at'' looping over windows.

attrcolor *attrib* [*attribute/color-modifier*]

This command can be used to highlight attributes by changing the color of the text. If the attribute *attrib* is in use, the specified attribute/color modifier is also applied. If no modifier is given, the current one is deleted. See the ''STRING ESCAPES'' chapter for the syntax of the modifier. Screen understands two pseudo-attributes, ''i'' stands for high-intensity foreground color and ''I'' for high-intensity background color.

Examples:

attrcolor b "R"

Change the color to bright red if bold text is to be printed.

attrcolor u "−u b"

Use blue text instead of underline.

attrcolor b ".I"

Use bright colors for bold text. Most terminal emulators do this already.

attrcolor i "+b"

Make bright colored text also bold.

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''.

backtick *id lifespan autorefresh cmd args...*

backtick *id*

Program the backtick command with the numerical id *id*. The output of such a command is used for substitution of the ''%'' string escape. The specified *lifespan* is the number of seconds the output is considered valid. After this time, the command is run again if a corresponding string escape is encountered. The *autorefresh* parameter triggers an automatic refresh for caption and hardstatus strings after the specified number of seconds. Only the last line of output is used for substitution.

If both the *lifespan* and the *autorefresh* parameters are zero, the backtick program is expected to stay in the background and generate output once in a while. In this case, the command is executed right away and screen stores the last line of output. If a new line gets printed screen will automatically refresh the hardstatus or the captions.

The second form of the command deletes the backtick command with the numerical id *id*.

bce [**on|off**]

Change background-color-erase setting. If ''bce'' is set to on, all characters cleared by an erase/insert/scroll/clear operation will be displayed in the current background color. Otherwise the default background color is used.

bell_msg [*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 this 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 '^G' is replaced by the definition for bell in your termcap (usually an audible bell). The default message is

'Bell in window %n'

An empty message can be supplied to the ''bell_msg'' command to suppress output of a message line (bell_msg ""). Without parameter, the current message is shown.

bind [*class*] *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.

If a command class is specified via the ''−c'' option, the key is bound for the specified class. Use the ''command'' command to activate a class. Command classes can be used to create multiple command keys or multi-character bindings.

Some examples:

```
bind ' ' windows
bind ^k
bind k
bind K kill
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''). The next three lines remove the default kill binding from ''C-a C-k'' and ''C-a k''. ''C-a K'' is then bound to the kill command. Then it binds ''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 superuser shell and a scrollback buffer of 1000 lines.

```
bind −c demo1 0 select 10
bind −c demo1 1 select 11
bind −c demo1 2 select 12
bindkey "^B" command −c demo1
```

makes ''C-b 0'' select window 10, ''C-b 1'' window 11, etc.

```
bind −c demo2 0 select 10
bind −c demo2 1 select 11
bind −c demo2 2 select 12
bind − command −c demo2
```

makes ''C-a − 0'' select window 10, ''C-a − 1'' window 11, etc.

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 inter-character timing. One cannot turn off the timing if a termcap capability 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 abbreviation of the word "barfoo". Timeout is disabled so that users can type slowly.

bindkey "\024" mapdefault

This key-binding makes ''^T'' an escape character for key-bindings. 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).

break[*duration*]

Send a break signal for *duration**0.25 seconds to this window. For non-Posix systems the time interval may be rounded up to full seconds. Most useful if a character device is attached to the window rather than a shell process (See also chapter ''WINDOW TYPES''). The maximum duration of a break signal is limited to 15 seconds.

blanker

Activate the screen blanker. First the screen is cleared. If no blanker program is defined, the cursor is turned off, otherwise, the program is started and it's output is written to the screen. The screen blanker is killed with the first keypress, the read key is discarded.

This command is normally used together with the ''idle'' command.

blankerprg [*program-args*]

Defines a blanker program. Disables the blanker program if an empty argument is given. Shows the currently set blanker program if no arguments are given.

breaktype [*tcsendbreak*|*TIOCSBRK*|*TCSBRK*]

Choose one of the available methods of generating a break signal for terminal devices. This command should affect the current window only. But it still behaves identical to ''defbreaktype''. This will be changed in the future. Calling ''breaktype'' with no parameter displays the break method for the current window.

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

bumpleft

Swaps window with previous one on window list.

bumpright

Swaps window with next one on window list.

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.

caption [top | bottom] always|splitonly[**string**]

caption string [*string*]

This command controls the display of the window captions. Normally a caption is only used if more than one window is shown on the display (split screen mode). But if the type is set to **always** screen shows a caption even if only one window is displayed. The default is **splitonly**.

The second form changes the text used for the caption. You can use all escapes from the ''STRING ESCAPES'' chapter. Screen uses a default of '%3n %t'.

You can mix both forms by providing a string as an additional argument.

You can have the caption displayed either at the top or bottom of the window. The default is **bottom**.

charset *set*

Change the current character set slot designation and charset mapping. The first four character of *set* are treated as charset designators while the fifth and sixth character must be in range '0' to '3' and set the GL/GR charset mapping. On every position a '.' may be used to indicate that the corresponding charset/mapping should not be changed (*set* is padded to six characters internally by appending '.' chars). New windows have "BBBB02" as default charset, unless a ''encoding'' command is active.
The current setting can be viewed with the ''info'' command.

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.

cjkwidth [on | off]

Treat ambiguous width characters as full/half width.

clear

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

collapse

Reorders window on window list, removing number gaps between them.

colon [*prefix*]

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 [**−c** *class*]

This command has the same effect as typing the screen escape character (^A). It is probably only useful for key bindings. If the ‘‘−c’’ option is given, select the specified command class. See also ‘‘bind’’ and ‘‘bindkey’’.

compacthist [**on**|**off**]

This tells screen whether to suppress trailing blank lines when scrolling up text into the history buffer.

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

Grabs or un-grabs the machines console output to a window. *Note*: Only the owner of /dev/console can grab the console output. This command is only available if the machine supports the ioctl TIOCCONS.

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:
The editor's movement keys are:

h, **C-h**, **left arrow**	move the cursor left.
j, **C-n**, **down arrow**	move the cursor down.
k, **C-p**, **up arrow**	move the cursor up.
l ('el'), **right arrow**	move the cursor right.
0 (zero) **C-a**	move to the leftmost column.
+ and **−**	positions one line up and down.
H, **M** and **L**	move the cursor to the leftmost column of the top, center or bottom line of the window.
|	moves to the specified absolute column.
g or **home**	moves to the beginning of the buffer.
G or **end**	moves to the specified absolute line (default: end of buffer).
%	jumps to the specified percentage of the buffer.
^ or **$**	move to the leftmost column, to the first or last non-whitespace character on the line.
w, **b**, and **e**	move the cursor word by word.
B, **E**	move the cursor WORD by WORD (as in vi).

f/F, **t/T**	move the cursor forward/backward to the next occurrence of the target. (eg, '3fy' will move the cursor to the 3rd 'y' to the right.)
; and **,**	Repeat the last f/F/t/T command in the same/opposite direction.
C-e and **C-y**	scroll the display up/down by one line while preserving the cursor position.
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.

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.

Some keys are defined to do mark and replace operations.

The copy range is specified by setting two marks. The text between these marks will be highlighted. Press:

space or **enter** to set the first or second mark respectively. If **mousetrack** is set to 'on', marks can also be set using **left mouse click**.

Y and **y** used to mark one whole line or to mark from start of line.

W marks exactly one word.

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.

The following search keys are defined:

/ *Vi*-like search forward.

? *Vi*-like search backward.

C-a s *Emacs* style incremental search forward.

C-r *Emacs* style reverse i-search.

n Find next search pattern.

N Find previous search pattern.

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 4 modes: lines separated by a newline character (012), lines glued seamless, lines separated by a single whitespace and comma separated lines. Note that you can prepend the newline character with a carriage return character, by issuing a ‘‘crlf on’’.

v or **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.

x or **o** exchanges the first mark and the current cursor position. You can use this to adjust an already placed mark.

C-l (’el’) will redraw the screen.

@ does nothing. Does not even exit copy mode.

All keys not described here 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. When no parameter is given, the state is toggled.

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 correctly. Debug output from attacher processes can only be turned off once and forever.

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.

defbce on|off

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

defbreaktype [*tcsendbreak*|*TIOCSBRK*|*TCSBRK*]

Choose one of the available methods of generating a break signal for terminal devices. The preferred methods are *tcsendbreak* and *TIOCSBRK*. The third, *TCSBRK*, blocks the complete *screen* session for the duration of the break, but it may be the only way to generate long breaks. *Tcsendbreak* and *TIOCSBRK* may or may not produce long breaks with spikes (e.g. 4 per second). This is not only system-dependent, this also differs between serial board drivers. Calling ‘‘defbreaktype’’ with no parameter displays the current setting.

defcharset [*set*]

Like the **charset** command except that the default setting for new windows is changed. Shows current default if called without argument.

defdynamictitle on|off

Set default behaviour for new windows regarding if screen should change window title when seeing proper escape sequence. See also "TITLES (naming windows)" section.

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’.

defhstatus [*status*]

The hardstatus line that all new windows will get is set to *status*. This command is useful to make the hardstatus of every window display the window number or title or the like. *Status* may contain the same directives as in the window messages, but the directive escape character is ’^E’ (octal 005) instead of ’%’. This was done to make a misinterpretation of program generated hardstatus lines impossible. If the parameter *status* is omitted, the current default string is displayed. Per default the hardstatus line of new windows is empty.

defencoding *enc*

Same as the **encoding** command except that the default setting for new windows is changed. Initial setting is the encoding taken from the terminal.

deflog on|off

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

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’.

defmousetrack on|off

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

defnonblock on|off|*numsecs*

Same as the **nonblock** command except that the default setting for displays 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.

defshell *command*

Synonym to the **shell** .screenrc command. See there.

defsilence on|off

Same as the **silence** command except that the default setting for new windows is changed. Initial setting is 'off'.

defslowpaste *msec*

Same as the **slowpaste** command except that the default setting for new windows is changed. Initial setting is 0 milliseconds, meaning 'off'.

defutf8 on|off

Same as the **utf8** command except that the default setting for new windows is changed. Initial setting is 'on' if screen was started with ''−U'', otherwise 'off'.

defwrap on|off

Same as the **wrap** command except that the default setting for new windows is changed. Initially line-wrap 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 write-locks will off.

detach [**−h**]

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''). The **−h** option tells screen to immediately close the connection to the terminal (''hangup'').

dinfo

Show what screen thinks about your terminal. Useful if you want to know why features like color or the alternate charset don't work.

displays

Shows a tabular listing of all currently connected user front-ends (displays). This is most useful for multi-user sessions. The following keys can be used in displays list:

k, **C-p**, or **up**	Move up one line.
j, **C-n**, or **down**	Move down one line.
C-a or **home**	Move to the first line.
C-e or **end**	Move to the last line.
C-u or **C-d**	Move one half page up or down.
C-b or **C-f**	Move one full page up or down.
mouseclick	Move to the selected line. Available when ''mousetrack'' is set to on.

space	Refresh the list
d	Detach that display
D	Power detach that display
C-g, **enter**, or **escape**	Exit the list

The following is an example of what ‘‘displays’’ could look like:

```
xterm 80x42 jnweiger@/dev/ttyp4     0(m11)    &rWx
facit 80x24 mlschroe@/dev/ttyhf nb 11(tcsh)    rwx
xterm 80x42 jnhollma@/dev/ttyp5     0(m11)    &R.x
 (A)    (B)      (C)      (D)      (E)  (F)(G)    (H)(I)
```

The legend is as follows:

(A) The terminal type known by screen for this display.

(B) Displays geometry as width x height.

(C) Username who is logged in at the display.

(D) Device name of the display or the attached device

(E) Display is in blocking or nonblocking mode. The available modes are "nb", "NB", "Z<", "Z>", and "BL".

(F) Number of the window

(G) Name/title of window

(H) Whether the window is shared

(I) Window permissions. Made up of three characters.

Window permissions indicators					
1st character		2nd character		3rd character	
–	no read	**–**	no write	**–**	no execute
r	read	**w**	write	**x**	execute
		W	own wlock		
Indicators of permissions suppressed by a foreign wlock					
R	read only	**.**	no write		

‘‘displays’’ needs a region size of at least 10 characters wide and 5 characters high in order to display.

digraph [*preset*[*unicode-value*]]

This command prompts the user for a digraph sequence. The next two characters typed are looked up in a builtin table and the resulting character is inserted in the input stream. For example, if the user enters ’a"’, an a-umlaut will be inserted. If the first character entered is a 0 (zero), *screen* will treat the following characters (up to three) as an octal number instead. The optional argument *preset* is treated as user input, thus one can create an ‘‘umlaut’’ key. For example the command "bindkey ^K digraph ’"’" enables the user to generate an a-umlaut by typing CTRL-K a. When a non-zero *unicode-value* is specified, a new digraph is created with the specified preset. The digraph is unset if a zero value is provided for the *unicode-value.*

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*.

dynamictitle on|off

Change behaviour for windows regarding if screen should change window title when seeing proper escape sequence. See also "TITLES (naming windows)" section.

echo [**−n**] *message*

The echo command may be used to annoy *screen* users with a 'message of the day'. Typically installed in a global /etc/screenrc. The option ''−n'' may be used to suppress the line feed. See also ''sleep''. Echo is also useful for online checking of environment variables.

encoding *enc* [*enc*]

Tell *screen* how to interpret the input/output. The first argument sets the encoding of the current window. Each window can emulate a different encoding. The optional second parameter overwrites the encoding of the connected terminal. It should never be needed as screen uses the locale setting to detect the encoding. There is also a way to select a terminal encoding depending on the terminal type by using the ''KJ'' termcap entry.

Supported encodings are eucJP, SJIS, eucKR, eucCN, Big5, GBK, KOI8-R, KOI8-U, CP1251, UTF-8, ISO8859-2, ISO8859-3, ISO8859-4, ISO8859-5, ISO8859-6, ISO8859-7, ISO8859-8, ISO8859-9, ISO8859-10, ISO8859-15, jis.

See also ''defencoding'', which changes the default setting of a new window.

escape *xy*

Set the command character to *x* and the character generating a literal command character (by triggering the ''meta'' command) to *y* (similar to 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''.

eval *command1*[*command2 ...*]

Parses and executes each argument as separate command.

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

Run a unix subprocess (specified by an executable path *newcommand* and its optional arguments) in the current window. The flow of data between newcommands stdin/stdout/stderr, the process originally started in the window (let us call it "application-process") and screen itself (window) is controlled by the file descriptor 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 application-process. A colon (:) combines both. User input will go to newcommand unless newcommand receives the application-process' output (fdpats first character is '!' or ':') or a pipe symbol (|) is added (as a fourth character) to the end of fdpat.

Invoking 'exec' without arguments shows name and arguments of the currently running subprocess in this window. Only one subprocess a time can be running in each window.

When a subprocess is running the 'kill' command will affect it instead of the windows process.

Refer to the postscript file 'doc/fdpat.ps' for a confusing illustration of all 21 possible combinations. Each drawing shows the digits 2,1,0 representing the three file descriptors of newcommand. The box marked 'W' is the usual pty that has the application-process 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 window's process. This works, because *less* listens on stderr (a behavior that *screen* would not expect without the '|') when its stdin is not a tty. *Less* versions newer than 177 fail miserably here; good old *pg* still works.

!: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.

fit

Change the window size to the size of the current region. This command is needed because screen doesn't adapt the window size automatically if the window is displayed more than once.

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'.

focus [**next|prev|up|down|left|right|top|bottom**]

Move the input focus to the next region. This is done in a cyclic way so that the top left region is selected after the bottom right one. If no option is given it defaults to 'next'. The next region to be selected is determined by how the regions are layered. Normally, the next region in the same layer would be selected. However, if that next region contains one or more layers, the first region in the highest layer is selected first. If you are at the last region of the current layer, 'next' will move the focus to the next region in the lower layer (if there is a lower layer). 'Prev' cycles in the opposite order. See ''split'' for more information about layers.

The rest of the options ('up', 'down', 'left', 'right', 'top', and 'bottom') are more indifferent to layers. The option 'up' will move the focus upward to the region that is touching the upper left corner of the current region. 'Down' will move downward to the region that is touching the lower left corner of the current region. The option 'left' will move the focus leftward to the region that is touching the upper left corner of the current region, while 'right' will move rightward to the region that is touching the upper right corner of the current region. Moving left from a left most region or moving right from a right most region will result in no action.

The option ‘top’ will move the focus to the very first region in the upper list corner of the screen, and ‘bottom’ will move to the region in the bottom right corner of the screen. Moving up from a top most region or moving down from a bottom most region will result in no action.

Useful bindings are (h, j, k, and l as in vi)
bind h focus left
bind j focus down
bind k focus up
bind l focus right
bind t focus top
bind b focus bottom
Note that **k** is traditionally bound to the *kill* command.

focusminsize [(*width*|**max|_) (** *height*|**max|_)]**

This forces any currently selected region to be automatically resized at least a certain *width* and *height*. All other surrounding regions will be resized in order to accommodate. This constraint follows every time the ‘‘focus’’ command is used. The ‘‘resize’’ command can be used to increase either dimension of a region, but never below what is set with ‘‘focusminsize’’. The underscore ‘_’ is a synonym for **max**. Setting a *width* and *height* of ‘0 0’ (zero zero) will undo any constraints and allow for manual resizing. Without any parameters, the minimum width and height is shown.

gr [**on|off**]

Turn GR charset switching on/off. Whenever screen sees an input character with the 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.

group [*grouptitle*]

Change or show the group the current window belongs to. Windows can be moved around between different groups by specifying the name of the destination group. Without specifying a group, the title of the current group is displayed.

hardcopy [**−***h*] [*file*]

Writes out the currently displayed image to the file *file*, or, if no filename is specified, to *hardcopy.n* in the default directory, where *n* is the number of the current window. This either appends or overwrites the file if it exists. See below. If the option **–h** is specified, dump also the contents of the scrollback buffer.

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**]

hardstatus [**always**]**firstline|lastline|message|ignore**[*string*]

hardstatus string[*string*]

This command configures the use and emulation of the terminal’s hardstatus line. The first form toggles whether *screen* will use the hardware status line to display messages. If the flag is set to ‘off’, these messages are overlaid in reverse video mode at the display line. The default setting is ‘on’.

The second form tells *screen* what to do if the terminal doesn’t have a hardstatus line (i.e. the termcap/terminfo capabilities "hs", "ts", "fs" and "ds" are not set). When ‘‘firstline/lastline’’ is used, *screen* will reserve the first/last line of the display for the hardstatus. ‘‘message’’ uses *screen’s* message mechanism and ‘‘ignore’’ tells *screen* never to display the hardstatus. If you prepend the word ‘‘always’’ to the type (e.g., ‘‘alwayslastline’’), *screen* will use the type even if the terminal supports a hardstatus.

The third form specifies the contents of the hardstatus line. '%h' is used as default string, i.e., the stored hardstatus of the current window (settable via "ESC]0;<string>^G" or "ESC_<string>ESC\") is displayed. You can customize this to any string you like including the escapes from the "STRING ESCAPES" chapter. If you leave out the argument *string*, the current string is displayed.

You can mix the second and third form by providing the string as additional argument.

height [**−w**|**−d**] [*lines* [*cols*]]

Set the display height to a specified number of lines. When no argument is given it toggles between 24 and 42 lines display. You can also specify a width if you want to change both values. The **−w** option tells screen to leave the display size unchanged and just set the window size, **−d** vice versa.

help[*class*]

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. If the "−c" option is given, display all bound commands for the specified command class. 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).

hstatus *status*

Change the window's hardstatus line to the string *status*.

idle [*timeout*[*cmd-args*]]

Sets a command that is run after the specified number of seconds inactivity is reached. This command will normally be the "blanker" command to create a screen blanker, but it can be any screen command. If no command is specified, only the timeout is set. A timeout of zero (or the special timeout **off**) disables the timer. If no arguments are given, the current settings are displayed.

ignorecase [**on**|**off**]

Tell screen to ignore the case of characters in searches. Default is 'off'. Without any options, the state of ignorecase is toggled.

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", the current state of window XON/XOFF flow control is shown like this (See also section FLOW CONTROL):

+flow	automatic flow control, currently on.
−flow	automatic flow control, currently off.
+(+)flow	flow control enabled. Agrees with automatic control.
−(+)flow	flow control disabled. Disagrees with automatic control.
+(−)flow	flow control enabled. Disagrees with automatic control.
−(−)flow	flow control disabled. Agrees with automatic control.

The current line wrap setting ('+wrap' indicates enabled, '−wrap' not) is also shown. The flags 'ins', 'org', 'app', 'log', 'mon' or 'nored' are displayed when the window is in insert mode, origin mode, application-keypad mode, has output logging, activity monitoring or partial redraw enabled.

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* is shown. If the window is in UTF-8 mode, the string ''UTF-8'' is shown instead.

Additional modes depending on the type of the window are displayed at the end of the status line (See also chapter ''WINDOW TYPES'').

If the state machine of the terminal emulator is in a non-default state, the info line is started with a string identifying the current state.

For system information use the ''time'' command.

ins_reg [*key*]

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

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* (your display) switches to another window. When the last window is destroyed, *screen* exits. After a kill *screen* switches to the previously displayed window.

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.

layout new [*title*]

Create a new layout. The screen will change to one whole region and be switched to the blank window. From here, you build the regions and the windows they show as you desire. The new layout will be numbered with the smallest available integer, starting with zero. You can optionally give a title to your new layout. Otherwise, it will have a default title of ''layout''. You can always change the title later by using the command **layout title**.

layout remove [*n*|*title*]

Remove, or in other words, delete the specified layout. Either the number or the title can be specified. Without either specification, *screen* will remove the current layout.

Removing a layout does not affect your set windows or regions.

layout next

Switch to the next layout available

layout prev

Switch to the previous layout available

layout select [*n*|*title*]

Select the desired layout. Either the number or the title can be specified. Without either specification, *screen* will prompt and ask which screen is desired. To see which layouts are available, use the **layout show** command.

layout show

List on the message line the number(s) and title(s) of the available layout(s). The current layout is flagged.

layout title [*title*]

Change or display the title of the current layout. A string given will be used to name the layout. Without any options, the current title and number is displayed on the message line.

layout number [*n*]

Change or display the number of the current layout. An integer given will be used to number the layout. Without any options, the current number and title is displayed on the message line.

layout attach [*title*|**:last**]

Change or display which layout to reattach back to. The default is **:last**, which tells *screen* to reattach back to the last used layout just before detachment. By supplying a title, You can instruct *screen* to reattach to a particular layout regardless which one was used at the time of detachment. Without any options, the layout to reattach to will be shown in the message line.

layout save [*n*|*title*]

Remember the current arrangement of regions. When used, *screen* will remember the arrangement of vertically and horizontally split regions. This arrangement is restored when a *screen* session is reattached or switched back from a different layout. If the session ends or the *screen* process dies, the layout arrangements are lost. The **layout dump** command should help in this siutation. If a number or title is supplied, *screen* will remember the arrangement of that particular layout. Without any options, *screen* will remember the current layout.

Saving your regions can be done automatically by using the **layout autosave** command.

layout autosave [**on**|**off**]

Change or display the status of automatcally saving layouts. The default is **on**, meaning when *screen* is detached or changed to a different layout, the arrangement of regions and windows will be remembered at the time of change and restored upon return. If autosave is set to **off**, that arrangement will only be restored to either to the last manual save, using **layout save**, or to when the layout was first created, to a single region with a single window. Without either an **on** or **off**, the current status is displayed on the message line.

layout dump [*filename*]

Write to a file the order of splits made in the current layout. This is useful to recreate the order of your regions used in your current layout. Only the current layout is recorded. While the order of the regions are recorded, the sizes of those regions and which windows correspond to which regions are not. If no filename is specified, the default is *layout-dump*, saved in the directory that the *screen* process was started in. If the file already exists, **layout dump** will append to that file. As an example:

C-a : layout dump /home/user/.screenrc

will save or append the layout to the user's *.screenrc* file.

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.

Warning: When you leave other shells unlocked and you have no password set on *screen*, the lock is void: One could easily re-attach from an unlocked shell. This feature should rather be called 'lockterminal'.

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. This filename can be changed with the ‘logfile’ command. 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’.

logfile *filename*

logfile flush *secs*

Defines the name the log files will get. The default is ‘‘screenlog.%n’’. The second form changes the number of seconds *screen* will wait before flushing the logfile buffer to the file-system. The default value is 10 seconds.

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.

logtstamp [**on**|**off**]

logtstamp after [*secs*]

logtstamp string
[*string*]

This command controls logfile time-stamp mechanism of *screen.* If time-stamps are turned ‘‘on’’, *screen* adds a string containing the current time to the logfile after two minutes of inactivity. When output continues and more than another two minutes have passed, a second time-stamp is added to document the restart of the output. You can change this timeout with the second form of the command. The third form is used for customizing the time-stamp string (‘–– %n:%t –– time-stamp –– %M/%d/%y %c:%s ––\n’ by default).

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 [*timeout*]

Set the inter-character timer for input sequence detection to a timeout of *timeout* 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. If your terminal sends characters, that cause you to abort copy mode, then this command may help by binding these characters to do nothing. The no-op character is ‘@’ and is used like this: ‘‘markkeys @=L=H’’ if you do not want to use the ‘H’ or ‘L’ commands any longer. As shown in this example, multiple keys can be assigned to one function in a single statement.

maxwin *num*

Set the maximum window number screen will create. Doesn’t affect already existing windows. The number

can be increased only when there are no existing windows.

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.

mousetrack [**on**|**off**]

This command determines whether *screen* will watch for mouse clicks. When this command is enabled, regions that have been split in various ways can be selected by pointing to them with a mouse and left-clicking them. Without specifying **on** or **off**, the current state is displayed. The default state is determined by the ''defmousetrack'' command.

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', 'aclgrp' and 'acldel' can be used to enable (and disable) other users accessing this *screen* session.

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 and the file ~/.nethackrc - if either one is present, the default is **on**.

next

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

nonblock [**on**|**off**|*numsecs*]

Tell screen how to deal with user interfaces (displays) that cease to accept output. This can happen if a user presses ^S or a TCP/modem connection gets cut but no hangup is received. If nonblock is **off** (this is the default) screen waits until the display restarts to accept the output. If nonblock is **on**, screen waits until the timeout is reached (**on** is treated as 1s). If the display still doesn't receive characters, screen will consider it ''blocked'' and stop sending characters to it. If at some time it restarts to accept characters, screen will unblock the display and redisplay the updated window contents.

number [[+|−]*n*]

Change the current window's 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. Using '+' or '−' will change the window's number by the relative amount specified.

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.

only

Kill all regions but the current one.

other

Switch to the window displayed previously. If this window does no longer exist, *other* has the same effect as *next*.

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 useful 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. Without parameter, the current message is shown.

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 [*encoding*] [*filename*]

Reads the contents of the specified file into the paste buffer. You can tell screen the encoding of the file via the **−e** option. If no file is specified, the screen-exchange filename is used. See also ''bufferfile'' command.

readreg [*encoding*] [*register* [*filename*]]

Does one of two things, dependent on number of arguments: with zero or one arguments 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. You can tell screen the encoding of the file via the **−e** option. 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 [**−e***encoding*]*key-string*

Save the specified *string* to the register *key*. The encoding of the string can be specified via the **−e** option. See also the ''paste'' command.

remove

Kill the current region. This is a no-op if there is only one region.

removebuf

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

rendition bell | monitor | silence | so *attr* [*color*]

Change the way *screen* renders the titles of windows that have monitor or bell flags set in caption or hardstatus or windowlist. See the ''STRING ESCAPES'' chapter for the syntax of the modifiers. The default for monitor is currently ''=b '' (bold, active colors), for bell ''=ub '' (underline, bold and active colors), and ''=u '' for silence.

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.

resize [**-h|-v|-b|-l|-p**] [[**+|-**] *n*[*%*] **|=|max|min|_|0**]

Resize the current region. The space will be removed from or added to the surrounding regions depending on the order of the splits. The available options for resizing are '-h'(horizontal), '-v'(vertical), '-b'(both), '-l'(local to layer), and '-p'(perpendicular). Horizontal resizes will add or remove width to a region, vertical will add or remove height, and both will add or remove size from both dimensions. Local and perpendicular are similar to horizontal and vertical, but they take in account of how a region was split. If a region's last split was horizontal, a local resize will work like a vertical resize. If a region's last split was vertical, a local resize will work like a horizontal resize. Perpendicular resizes work in opposite of local resizes. If no option is specified, local is the default.

The amount of lines to add or remove can be expressed a couple of different ways. By specifying a number *n* by itself will resize the region by that absolute amount. You can specify a relative amount by prefixing a plus '+' or minus '-' to the amount, such as adding *+n* lines or removing *-n* lines. Resizing can also be expressed as an absolute or relative percentage by postfixing a percent sign '*%*'. Using zero '0' is a synonym for 'min' and using an underscore '_' is a synonym for 'max'.

Some examples are:

resize +N
: increase current region by N

resize −N
: decrease current region by N

resize N
: set current region to N

resize 20%
: set current region to 20% of original size

resize +20%
: increase current region by 20%

resize -b =
: make all windows equally

resize max
: maximize current region

resize min
: minimize current region

Without any arguments, *screen* will prompt for how you would like to resize the current region.

See ''focusminsize'' if you want to restrict the minimum size a region can have.

screen [−*opts*] [*n*] [*cmd* [*args*]|**//group**]

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>), the all-capability-flag (**−a**) and scrollback option (**−h** <num>) may be specified with each command. The option (**−M**) turns monitoring on for this window. The option (**−L**) turns output logging on for this window. If an optional number *n* in the range 0..MAXWIN-1 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. If **//group** is supplied, a container-type window is created in which other windows may be created inside it.

Thus, if your ''.screenrc'' contains the lines

```
# example for .screenrc:
screen 1
screen −fn −t foobar −L 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) and will write a logfile (''screenlog.2'') of the telnet session. 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.

Screen has built in some functionality of ''cu'' and ''telnet''. See also chapter ''WINDOW TYPES''.

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 ‘‘info’’ to view the current setting. To access and use the contents in the scrollback buffer, use the ‘‘copy’’ command.

select [*WindowID*]

Switch to the window identified by *WindowID*. This can be a prefix of a window title (alphanumeric window name) or a window number. The parameter is optional and if omitted, you get prompted for an identifier. 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’’. The number of windows is limited at compile-time by the MAXWIN configuration parameter (which defaults to 40). There are two special WindowIDs, ‘‘−’’ selects the internal blank window and ‘‘.’’ selects the current window. The latter is useful if used with screen’s ‘‘−X’’ option.

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 will still reflect the old name in pre-existing shells. This may result in confusion. Use of this command is generally discouraged. Use the ‘‘−S’’ command-line option if you want to name a new session. 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.

setsid [**on**|**off**]

Normally screen uses different sessions and process groups for the windows. If setsid is turned *off*, this is not done anymore and all windows will be in the same process group as the screen backend process. This also breaks job-control, so be careful. The default is *on*, of course. This command is probably useful only in rare circumstances.

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. Typical shells do only minimal initialization when not started as a login-shell. E.g. Bash will not read your ‘‘~/.bashrc’’ unless it is 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 *msec*

Define the speed at which text is inserted into the current window by the paste ("C-a]") command. If the slowpaste value is nonzero text is written character by character. *screen* will make a pause of *msec* 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.

sort

Sort the windows in alphabetical order of the window tiles.

source *file*

Read and execute commands from file *file*. Source commands may be nested to a maximum recursion level of ten. If file is not an absolute path and screen is already processing a source command, the parent directory of the running source command file is used to search for the new command file before screen's current directory.

Note that termcap/terminfo/termcapinfo commands only work at startup and reattach time, so they must be reached via the default screenrc files to have an effect.

sorendition [*attr*[*color*]]

This command is deprecated. See "rendition so" instead.

split[**−v**]

Split the current region into two new ones. All regions on the display are resized to make room for the new region. The blank window is displayed in the new region. The default is to create a horizontal split, putting the new regions on the top and bottom of each other. Using ‘-v’ will create a vertical split, causing the new regions to appear side by side of each other. Use the ‘‘remove’’ or the ‘‘only’’ command to delete regions. Use ‘‘focus’’ to toggle between regions.

When a region is split opposite of how it was previously split (that is, vertical then horizontal or horizontal then vertical), a new layer is created. The layer is used to group together the regions that are split the same. Normally, as a user, you should not see nor have to worry about layers, but they will affect how some commands (‘‘focus’’ and ‘‘resize’’) behave.

With this current implementation of screen, scrolling data will appear much slower in a vertically split region than one that is not. This should be taken into consideration if you need to use system commands such as ‘‘cat’’ or ‘‘tail -f’’.

startup_message on|off

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

status [**top|up|down|bottom**] [**left|right**]

The status window by default is in bottom-left corner. This command can move status messages to any corner of the screen. **top** is the same as **up**, **down** is the same as **bottom**.

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. Without a parameter, screen will prompt for a string to stuff. You cannot paste large buffers with the ‘‘stuff’’ command. It is most useful for key bindings. See also ‘‘bindkey’’.

su [*username* [*password* [*password2*]]]

Substitute the user of a display. The command prompts for all parameters that are omitted. If passwords are specified as parameters, they have to be specified un-crypted. The first password is matched against the systems passwd database, the second password is matched against the *screen* password as set with the

commands ''acladd'' or ''password''. ''Su'' may be useful for the *screen* administrator to test multiuser setups. When the identification fails, the user has access to the commands available for user **nobody**. These are ''detach'', ''license'', ''version'', ''help'' and ''displays''.

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 and resetting the default.

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

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

termcapinfo *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. You have to place these commands in one of the screenrc startup files, as they are meaningless once the terminal emulator is booted.

If your system uses the terminfo database rather than termcap, *screen* will understand the 'terminfo' command, which has the same effects as the 'termcap' command. Two separate commands are provided, as there are subtle syntactic differences, e.g. when parameter interpolation (using '%') is required. Note that termcap names of the capabilities have to be used with the 'terminfo' command.

In many cases, where the arguments are valid in both terminfo and termcap syntax, you can use the command 'termcapinfo', which is just a shorthand for a pair of 'termcap' and 'terminfo' commands with identical arguments.

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 [*string*]

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''.

If a string is specified, it changes the format of the time report like it is described in the ''STRING ESCAPES'' chapter. Screen uses a default of "%c:%s %M %d %H%? %l%?".

title [*windowtitle*]

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

unbindall

Unbind all the bindings. This can be useful when screen is used solely for its detaching abilities, such as when letting a console application run as a daemon. If, for some reason, it is necessary to bind commands after this, use 'screen −X'.

unsetenv *var*

Unset an environment variable.

utf8 [**on**|**off**[**on**|**off**]]

Change the encoding used in the current window. If utf8 is enabled, the strings sent to the window will be UTF-8 encoded and vice versa. Omitting the parameter toggles the setting. If a second parameter is given, the display's encoding is also changed (this should rather be done with screen's ''−U'' option). See also ''defutf8'', which changes the default setting of a new window.

vbell [**on**|**off**]

Sets the visual bell setting for this window. Omitting the parameter toggles the setting. If vbell is switched on, but your terminal does not support a visual bell, a 'vbell-message' is displayed in the status line when the bell character (^G) is received. Visual bell support of a terminal is defined by the termcap variable 'vb' (terminfo: 'flash').

Per default, vbell is off, thus the audible bell is used. See also 'bell_msg'.

vbell_msg [*message*]

Sets the visual bell message. *message* is printed to the status line if the window receives a bell character (^G), vbell is set to ''on'', but the terminal does not support a visual bell. The default message is ''Wuff, Wuff!!''. Without a parameter, the current message is shown.

vbellwait *sec*

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

verbose [**on**|**off**]

If verbose is switched on, the command name is echoed, whenever a window is created (or resurrected from zombie state). Default is off. Without a parameter, the current setting is shown.

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 [**−w**|**−d**] [*cols* [*lines*]]

Toggle the window width between 80 and 132 columns or set it to *cols* 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. You can also specify a new height if you want to change both values. The **−w** option tells screen to leave the display size unchanged and just set the window size, **−d** vice versa.

windowlist [**−b**] [**−m**] [**−g**]

windowlist string [*string*]

windowlist title [*title*]

Display all windows in a table for visual window selection. If screen was in a window group, screen will back out of the group and then display the windows in that group. If the **−b** option is given, screen will switch to the blank window before presenting the list, so that the current window is also selectable. The **−m** option changes the order of the windows, instead of sorting by window numbers screen uses its internal most-recently-used list. The **−g** option will show the windows inside any groups in that level and downwards.

The following keys are used to navigate in ''windowlist'':

k, **C-p**, or **up**	Move up one line.
j, **C-n**, or **down**	Move down one line.
C-g or **escape**	Exit windowlist.
C-a or **home**	Move to the first line.
C-e or **end**	Move to the last line.
C-u or **C-d**	Move one half page up or down.
C-b or **C-f**	Move one full page up or down.
0..9	Using the number keys, move to the selected line.
mouseclick	Move to the selected line. Available when ''mousetrack'' is set to ''on''
/	Search.
n	Repeat search in the forward direction.
N	Repeat search in the backward direction.
m	Toggle MRU.
g	Toggle group nesting.
a	All window view.
C-h or backspace	Back out the group.
,	Switch numbers with the previous window.
.	Switch numbers with the next window.
K	Kill that window.
space or **enter**	Select that window.

The table format can be changed with the **string** and **title** option, the title is displayed as table heading,

while the lines are made by using the string setting. The default setting is ''Num Name%=Flags'' for the title and ''%3n %t%=%f'' for the lines. See the ''STRING ESCAPES'' chapter for more codes (e.g. color settings).

''Windowlist'' needs a region size of at least 10 characters wide and 6 characters high in order to display.

windows [string]

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. The optional string parameter follows the ''STRING ESCAPES'' format. If string parameter is passed, the output size is unlimited. The default command without any parameter is limited to a size of 1024 bytes.

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'. Without any options, the state of wrap is toggled.

writebuf [**−e** *encoding*] [*filename*]

Writes the contents of the paste buffer to the specified file, or the public accessible screen−exchange file if no filename is given. This is thought of as a primitive means of communication between *screen* users on the same host. If an encoding is specified the paste buffer is recoded on the fly to match the encoding. 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.

zmodem [**off**|**auto**|**catch**|**pass**]

zmodem sendcmd [*string*]

zmodem recvcmd [*string*]

Define zmodem support for screen. Screen understands two different modes when it detects a zmodem request: ''pass'' and ''catch''. If the mode is set to ''pass'', screen will relay all data to the attacher until the end of the transmission is reached. In ''catch'' mode screen acts as a zmodem endpoint and starts the corresponding rz/sz commands. If the mode is set to ''auto'', screen will use ''catch'' if the window is a tty (e.g. a serial line), otherwise it will use ''pass''.

You can define the templates screen uses in ''catch'' mode via the second and the third form.

Note also that this is an experimental feature.

zombie [*keys*[onerror]]

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** command 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 probably be called **defzombie**, but it isn't.

Optionally you can put the word ''onerror'' after the keys. This will cause screen to monitor exit status of the process running in the window. If it exits normally ('0'), the window disappears. Any other exit value causes the window to become a zombie.

zombie_timeout[*seconds*]

Per default *screen* windows are removed from the window list as soon as the windows process (e.g. shell) exits. If **zombie** keys are defined (compare with above **zombie** command), it is possible to also set a timeout when screen tries to automatically reconnect a dead screen window.

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.

WINDOW TYPES

Screen provides three different window types. New windows are created with *screen*'s **screen** command (see also the entry in chapter ''CUSTOMIZATION''). The first parameter to the **screen** command defines which type of window is created. The different window types are all special cases of the normal type. They have been added in order to allow *screen* to be used efficiently as a console multiplexer with 100 or more windows.

- The normal window contains a shell (default, if no parameter is given) or any other system command that could be executed from a shell (e.g. **slogin**, etc...)

- If a tty (character special device) name (e.g. ''/dev/ttya'') is specified as the first parameter, then the window is directly connected to this device. This window type is similar to ''screen cu –l /dev/ttya''. Read and write access is required on the device node, an exclusive open is attempted on the node to mark the connection line as busy. An optional parameter is allowed consisting of a comma separated list of flags in the notation used by stty(1):

  <baud_rate>
  Usually 300, 1200, 9600 or 19200. This affects transmission as well as receive speed.

cs8 or cs7
Specify the transmission of eight (or seven) bits per byte.

ixon or −ixon
Enables (or disables) software flow-control (CTRL-S/CTRL-Q) for sending data.

ixoff or −ixoff
Enables (or disables) software flow-control for receiving data.

istrip or −istrip
Clear (or keep) the eight bit in each received byte.

You may want to specify as many of these options as applicable. Unspecified options cause the terminal driver to make up the parameter values of the connection. These values are system dependent and may be in defaults or values saved from a previous connection.

For tty windows, the **info** command shows some of the modem control lines in the status line. These may include 'RTS', 'CTS', 'DTR', 'DSR', 'CD' and more. This depends on the available ioctl()'s and system header files as well as the on the physical capabilities of the serial board. Signals that are logical low (inactive) have their name preceded by an exclamation mark (!), otherwise the signal is logical high (active). Signals not supported by the hardware but available to the ioctl() interface are usually shown low.

When the CLOCAL status bit is true, the whole set of modem signals is placed inside curly braces ({ and }). When the CRTSCTS or TIOCSOFTCAR bit is set, the signals 'CTS' or 'CD' are shown in parenthesis, respectively.

For tty windows, the command **break** causes the Data transmission line (TxD) to go low for a specified period of time. This is expected to be interpreted as break signal on the other side. No data is sent and no modem control line is changed when a **break** is issued.

- If the first parameter is ''//telnet'', the second parameter is expected to be a host name, and an optional third parameter may specify a TCP port number (default decimal 23). Screen will connect to a server listening on the remote host and use the telnet protocol to communicate with that server.

For telnet windows, the command **info** shows details about the connection in square brackets ([and]) at the end of the status line.

b BINARY. The connection is in binary mode.

e ECHO. Local echo is disabled.

c SGA. The connection is in 'character mode' (default: 'line mode').

t TTYPE. The terminal type has been requested by the remote host. Screen sends the name ''screen'' unless instructed otherwise (see also the command 'term').

w NAWS. The remote site is notified about window size changes.

f LFLOW. The remote host will send flow control information. (Ignored at the moment.)

Additional flags for debugging are x, t and n (XDISPLOC, TSPEED and NEWENV).

For telnet windows, the command **break** sends the telnet code IAC BREAK (decimal 243) to the remote host.

This window type is only available if *screen* was compiled with the ENABLE_TELNET option defined.

STRING ESCAPES

Screen provides an escape mechanism to insert information like the current time into messages or file names. The escape character is '%' with one exception: inside of a window's hardstatus '^%' ('^E') is used instead.

Here is the full list of supported escapes:

%	the escape character itself
E	sets %? to true if the escape character has been pressed.
f	flags of the window, see ''windows'' for meanings of the various flags
F	sets %? to true if the window has the focus
h	hardstatus of the window
H	hostname of the system
n	window number
P	sets %? to true if the current region is in copy/paste mode
S	session name
s	window size
t	window title
u	all other users on this window
w	all window numbers and names. With '−' qualifier: up to the current window; with '+' qualifier: starting with the window after the current one.
W	all window numbers and names except the current one
x	the executed command including arguments running in this windows
X	the executed command without arguments running in this windows
?	the part to the next '%?' is displayed only if a '%' escape inside the part expands to a non-empty string
:	else part of '%?'
=	pad the string to the display's width (like TeX's hfill). If a number is specified, pad to the percentage of the window's width. A '0' qualifier tells screen to treat the number as absolute position. You can specify to pad relative to the last absolute pad position by adding a '+' qualifier or to pad relative to the right margin by using '−'. The padding truncates the string if the specified position lies before the current position. Add the 'L' qualifier to change this.
<	same as '%=' but just do truncation, do not fill with spaces
>	mark the current text position for the next truncation. When screen needs to do truncation, it tries to do it in a way that the marked position gets moved to the specified percentage of the output area. (The area starts from the last absolute pad position and ends with the position specified by the truncation operator.) The 'L' qualifier tells screen to mark the truncated parts with '...'.
{	attribute/color modifier string terminated by the next ''}''
‘	Substitute with the output of a 'backtick' command. The length qualifier is misused to identify one of the commands.

The 'c' and 'C' escape may be qualified with a '0' to make *screen* use zero instead of space as fill character. The '0' qualifier also makes the '=' escape use absolute positions. The 'n' and '=' escapes understand a length qualifier (e.g. '%3n'), 'D' and 'M' can be prefixed with 'L' to generate long names, 'w' and 'W' also show the window flags if 'L' is given.

An attribute/color modifier is used to change the attributes or the color settings. Its format is ''[attribute

modifier] [color description]''. The attribute modifier must be prefixed by a change type indicator if it can be confused with a color description. The following change types are known:

+ add the specified set to the current attributes

− remove the set from the current attributes

! invert the set in the current attributes

= change the current attributes to the specified set

The attribute set can either be specified as a hexadecimal number or a combination of the following letters:

d dim
u underline
b bold
r reverse
s standout
B blinking

Colors are coded either as a hexadecimal number or two letters specifying the desired background and foreground color (in that order). The following colors are known:

k black
r red
g green
y yellow
b blue
m magenta
c cyan
w white
d default color
. leave color unchanged

The capitalized versions of the letter specify bright colors. You can also use the pseudo-color 'i' to set just the brightness and leave the color unchanged.
A one digit/letter color description is treated as foreground or background color dependent on the current attributes: if reverse mode is set, the background color is changed instead of the foreground color. If you don't like this, prefix the color with a ".". If you want the same behavior for two-letter color descriptions, also prefix them with a ".".
As a special case, "%{−}" restores the attributes and colors that were set before the last change was made (i.e., pops one level of the color-change stack).

Examples:

"G" set color to bright green

"+b r" use bold red

"= yd" clear all attributes, write in default color on yellow background.

%−Lw%{= BW}%50>%n%f* %t%{−}%+Lw%<
The available windows centered at the current window and truncated to the available width. The current window is displayed white on blue. This can be used with "hardstatus alwayslastline".

%?%F%{.R.}%?%3n %t%? [%h]%?
The window number and title and the window's hardstatus, if one is set. Also use a red background if this is the active focus. Useful for "caption string".

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 ’automatic’ interactively with the ‘‘flow’’ command bound to "C-a f".

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. Changing title by this escape sequence can be controlled by **defdynamictitle** and **dynamictitle** commands.

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 named ‘‘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='printf "\033k\033\134"'

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

THE VIRTUAL TERMINAL

Each window in a *screen* session emulates a VT100 terminal, with some extra functions added. The VT100 emulator is hard-coded, no other terminal types can be emulated.

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 latter is 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. The *info* command displays a line starting 'PRIN' while the printer is active.

Screen maintains a hardstatus line for every window. If a window gets selected, the display's hardstatus will be updated to match the window's hardstatus line. If the display has no hardstatus the line will be displayed as a standard *screen* message. The hardstatus line can be changed with the ANSI Application Program Command (APC): ''ESC_<string>ESC\''. As a convenience for xterm users the sequence ''ESC]0..2;<string>^G'' is also accepted.

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 "alternate screen" capability is not enabled by default. Set the **altscreen** .screenrc command to enable it.

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 g		Visual Bell	
ESC Pn **p**		Cursor Visibility (97801)	
		Pn = **6**	Invisible
		Pn = **7**	Visible
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 (Hardstatus)	
ESC] 0 ; string ^G	(A)	Operating System Command (Hardstatus, xterm title hack)	
ESC] 83 ; cmd ^G	(A)	Execute screen command. This only works if multi-user support is compiled into screen. The pseudo-user ‘‘:window:’’ is used to check the access control list. Use ‘‘addacl :window: –rwx #?’’ to create a user with no rights and allow only the needed commands.	
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**		same as above	
ESC [Pn **J**		Erase in Display	
		Pn = None or **0**	From Cursor to End of Screen
		Pn = **1**	From Beginning of Screen to Cursor
		Pn = **2**	Entire Screen
ESC [Pn **K**		Erase in Line	
		Pn = None or **0**	From Cursor to End of Line
		Pn = **1**	From Beginning of Line to Cursor
		Pn = **2**	Entire Line
ESC [Pn **X**		Erase character	
ESC [Pn **A**		Cursor Up	
ESC [Pn **B**		Cursor Down	
ESC [Pn **C**		Cursor Right	
ESC [Pn **D**		Cursor Left	
ESC [Pn **E**		Cursor next line	
ESC [Pn **F**		Cursor previous line	
ESC [Pn **G**		Cursor horizontal position	
ESC [Pn **‘**		same as above	
ESC [Pn **d**		Cursor vertical position	
ESC [Ps **;...;** Ps **m**		Select Graphic Rendition	
		Ps = None or **0**	Default Rendition
		Ps = **1**	Bold
		Ps = **2** (A)	Faint
		Ps = **3** (A)	*Standout* Mode (ANSI: Italicized)
		Ps = **4**	Underlined
		Ps = **5**	Blinking
		Ps = **7**	Negative Image
		Ps = **22** (A)	Normal Intensity
		Ps = **23** (A)	*Standout* Mode off (ANSI: Italicized off)
		Ps = **24** (A)	Not Underlined
		Ps = **25** (A)	Not Blinking
		Ps = **27** (A)	Positive Image
		Ps = **30** (A)	Foreground Black
		Ps = **31** (A)	Foreground Red

		Ps = **32**	(A) Foreground Green
		Ps = **33**	(A) Foreground Yellow
		Ps = **34**	(A) Foreground Blue
		Ps = **35**	(A) Foreground Magenta
		Ps = **36**	(A) Foreground Cyan
		Ps = **37**	(A) Foreground White
		Ps = **39**	(A) Foreground Default
		Ps = **40**	(A) Background Black
		Ps = **...**	
		Ps = **49**	(A) Background Default
ESC [Pn **g**		Tab Clear	
		Pn = None or **0**	Clear Tab at Current Position
		Pn = **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 [Pn **S**		Scroll Scrolling Region Up	
ESC [Pn **T**		Scroll Scrolling Region Down	
ESC [Pn **^**		same as above	
ESC [Ps **;...;** Ps **h**		Set Mode	
ESC [Ps **;...;** Ps **l**		Reset Mode	
		Ps = **4**	(A) Insert Mode
		Ps = **20**	(A) *Automatic Linefeed* Mode
		Ps = **34**	Normal Cursor Visibility
		Ps = **?1**	(V) Application Cursor Keys
		Ps = **?3**	(V) Change Terminal Width to 132 columns
		Ps = **?5**	(V) Reverse Video
		Ps = **?6**	(V) *Origin* Mode
		Ps = **?7**	(V) *Wrap* Mode
		Ps = **?9**	X10 mouse tracking
		Ps = **?25**	(V) Visible Cursor
		Ps = **?47**	Alternate Screen (old xterm code)
		Ps = **?1000**	(V) VT200 mouse tracking
		Ps = **?1047**	Alternate Screen (new xterm code)

		Ps = **?1049** Alternate Screen (new xterm code)
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 [x		Send Terminal Parameter Report
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 possible 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. The fourth is what command is executed if the keyboard is switched into application mode.

Key name	Termcap name	Command	App mode
Cursor up	ku	\033[A	\033OA
Cursor down	kd	\033[B	\033OB
Cursor right	kr	\033[C	\033OC
Cursor left	kl	\033[D	\033OD
Function key 0	k0	\033[10~	
Function key 1	k1	\033OP	
Function key 2	k2	\033OQ	
Function key 3	k3	\033OR	
Function key 4	k4	\033OS	
Function key 5	k5	\033[15~	
Function key 6	k6	\033[17~	
Function key 7	k7	\033[18~	
Function key 8	k8	\033[19~	
Function key 9	k9	\033[20~	
Function key 10	k;	\033[21~	
Function key 11	F1	\033[23~	
Function key 12	F2	\033[24~	
Home	kh	\033[1~	
End	kH	\033[4~	
Insert	kI	\033[2~	
Delete	kD	\033[3~	
Page up	kP	\033[5~	
Page down	kN	\033[6~	
Keypad 0	f0	0	\033Op
Keypad 1	f1	1	\033Oq
Keypad 2	f2	2	\033Or
Keypad 3	f3	3	\033Os
Keypad 4	f4	4	\033Ot
Keypad 5	f5	5	\033Ou
Keypad 6	f6	6	\033Ov
Keypad 7	f7	7	\033Ow
Keypad 8	f8	8	\033Ox
Keypad 9	f9	9	\033Oy
Keypad +	f+	+	\033Ok
Keypad −	f−	−	\033Om
Keypad *	f*	*	\033Oj
Keypad /	f/	/	\033Oo
Keypad =	fq	=	\033OX
Keypad .	f.	.	\033On
Keypad ,	f,	,	\033Ol
Keypad enter	fe	\015	\033OM

SPECIAL TERMINAL CAPABILITIES

The following table describes all terminal capabilities that are recognized by *screen* and are not in the termcap(5) manual. You can place these capabilities in your termcap entries (in '/etc/termcap') or use them with the commands 'termcap', 'terminfo' and 'termcapinfo' in your screenrc files. It is often not possible to

place these capabilities in the terminfo database.

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

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(%.’.

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 cursor-keys to application mode.

CE *(str)* Switch cursor-keys 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 encoding of the terminal. See the ’encoding’ command for valid encodings.

AF *(str)* Change character foreground color in an ANSI conform way. This capability will almost always be set to ’\E[3%dm’ (’\E[3%p1%dm’ on terminfo machines).

AB *(str)* Same as ’AF’, but change background color.

AX *(bool)* Does understand ANSI set default fg/bg color (\E[39m / \E[49m).

XC *(str)* Describe a translation of characters to strings depending on the current font. More details follow in the next section.

XT *(bool)* Terminal understands special xterm sequences (OSC, mouse tracking).

C8 *(bool)* Terminal needs bold to display high-intensity colors (e.g. Eterm).

TF *(bool)* Add missing capabilities to the termcap/info entry. (Set by default).

CHARACTER TRANSLATION

Screen has a powerful mechanism to translate characters to arbitrary strings depending on the current font and terminal type. Use this feature if you want to work with a common standard character set (say ISO8851-latin1) even on terminals that scatter the more unusual characters over several national language font pages.

Syntax:

XC=<*charset-mapping*>{**,,**<*charset-mapping*>}
<*charset-mapping*> := <*designator*><*template*>{**,**<*mapping*>}
<*mapping*> := <*char-to-be-mapped*><*template-arg*>

The things in braces may be repeated any number of times.

A <*charset-mapping*> tells *screen* how to map characters in font <*designator*> (’B’: Ascii, ’A’: UK, ’K’: German, etc.) to strings. Every <*mapping*> describes to what string a single character will be translated. A template mechanism is used, as most of the time the codes have a lot in common (for example strings to switch to and from another charset). Each occurrence of ’%’ in <*template*> gets substituted with the <*template-arg*> specified together with the character. If your strings are not similar at all, then use ’%’ as a

template and place the full string in *<template-arg>*. A quoting mechanism was added to make it possible to use a real '%'. The '\' character quotes the special characters '\', '%', and ','.

Here is an example:

```
termcap hp700 'XC=B\E(K%\E(B,\304[,\326\\\\,\334]'
```

This tells *screen* how to translate ISOlatin1 (charset 'B') upper case umlaut characters on a hp700 terminal that has a German charset. '\304' gets translated to '\E(K[\E(B' and so on. Note that this line gets parsed *three* times before the internal lookup table is built, therefore a lot of quoting is needed to create a single '\'.

Another extension was added to allow more emulation: If a mapping translates the unquoted '%' char, it will be sent to the terminal whenever *screen* switches to the corresponding *<designator>*. In this special case the template is assumed to be just '%' because the charset switch sequence and the character mappings normally haven't much in common.

This example shows one use of the extension:

```
termcap xterm 'XC=K%,%\E(B,[\304,\\\\\326,]\334'
```

Here, a part of the German ('K') charset is emulated on an xterm. If *screen* has to change to the 'K' charset, '\E(B' will be sent to the terminal, i.e. the ASCII charset is used instead. The template is just '%', so the mapping is straightforward: '[' to '\304', '\' to '\326', and ']' to '\334'.

ENVIRONMENT

COLUMNS	Number of columns on the terminal (overrides termcap entry).
HOME	Directory in which to look for .screenrc.
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''). See also ''shell'' .screenrc command.
STY	Alternate socket name.
SYSSCREENRC	Alternate system screenrc file.
TERM	Terminal name.
TERMCAP	Terminal description.
WINDOW	Window number of a window (at creation time).

FILES

.../screen-4.?.??/etc/screenrc .../screen-4.?.??/etc/etcscreenrc	Examples in the *screen* distribution package for private and global initialization files.
$SYSSCREENRC /etc/screenrc	*screen* initialization commands
$SCREENRC $HOME/.screenrc	Read in after /etc/screenrc

$SCREENDIR/S−<login>	
/run/screen/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	or
/tmp/screen−exchange	*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/?/*	or
/etc/termcap	Terminal capability databases
/run/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. For a long time maintained and developed by Juergen Weigert, Michael Schroeder, Micah Cowan and Sadrul Habib Chowdhury. Since 2015 maintained and developed by Amadeusz Slawinski <amade@asmblr.net> and Alexander Naumov <alexander_naumov@opensuse.org>.

COPYLEFT

CONTRIBUTORS

Maarten ter Huurne <maarten@treewalker.org>,
Jussi Kukkonen <jussi.kukkonen@intel.com>,
Eric S. Raymond <esr@thyrsus.com>,
Thomas Renninger <treen@suse.com>,
Axel Beckert <abe@deuxchevaux.org>,
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 <mrg@eterna.com.au>,
Christopher Williams <cgw@pobox.com>,
Matt Mosley <mattm@access.digex.net>,
Gregory Neil Shapiro <gshapiro@wpi.WPI.EDU>,
Johannes Zellner <johannes@zellner.org>,
Pablo Averbuj <pablo@averbuj.com>.

AVAILABILITY

The latest official release of *screen* available via anonymous ftp from ftp.gnu.org/gnu/screen/ or any other *GNU* distribution site. The home site of *screen* is savannah.gnu.org/projects/screen/. If you want to help, send a note to screen-devel@gnu.org.

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.
- 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 on most systems 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 ''/run/utmp''.

- Entries in ‘‘/run/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.
- If a password is set, the command line options −d and −D still detach a session without asking.
- Both ‘‘breaktype’’ and ‘‘defbreaktype’’ change the break generating method used by all terminal devices. The first should change a window specific setting, where the latter should change only the default for new windows.
- When attaching to a multiuser session, the user’s .screenrc file is not sourced. Each user’s personal settings have to be included in the .screenrc file from which the session is booted, or have to be changed manually.
- A weird imagination is most useful to gain full advantage of all the features.
- Send bug-reports, fixes, enhancements, t-shirts, money, beer & pizza to **screen-devel@gnu.org**.