Terminal Multiplexers: Screen, Tmux

command line options | key bindings | customization | terminology | documentation

Command Line Options

	screen	tmux
create session and attach	\$ screen	\$ tmux
create session foo and attach	\$ screen -S foo	\$ tmux new -s foo
create detached session foo	\$ screen -S foo -d -m	\$ tmux new -s foo -d
list sessions	\$ screen -list	\$ tmux 1s
attach	\$ screen -r	\$ tmux attach
attach to session foo	\$ screen -r foo	\$ tmux attach -t foo
attach to session by pid	\$ screen -r pid	
kill session foo	\$ screen -r foo -X quit	\$ tmux kill-session -t foo
send multiplexer command to session foo	\$ screen -r foo -X command	\$ tmux command -t foo
run Is in session foo	\$ screen -r foo -X stuff "ls \$(echo -ne '\015')"	\$ tmux send-keys -t foo 'ls' C-m
run vi in new window	\$ screen vi /etc/motd	\$ tmux new-window vi /etc/motd

C-a cmd

?

screen

help

sessions | windows | regions | panes | paste buffer | copy mode

send prefix to app	a	:meta	C-b	:send-prefix
suspend multiplexer	C-z	:suspend	C-z	:suspend-client
show previous multiplexer message	m C-m	:lastmsg	~	:show-messages
source file		:source file		:source-file file
detach	d C-d	:detach	d	:detach-client
		sessions		
	screen			tmux
	C-a cmd	command	C-b cmd	command
new session				:new
new named session				:new -s foo
switch session			s	:choose-session

:help

command

tmux

:list-keys

command

C-b cmd

?

	C-d			
		sessions		
	screen			tmux
	C-a cmd	command	C-b cmd	command
new session				:new
new named session				:new -s foo
switch session			S	:choose-session
rename session		:sessionname foo	\$	command-prompt -I #S "rename-session '%%'
kill session	C-\	:quit		
		windows	i) c	
	screen			tmux
	C-a cmd	command	C-b cmd	command
switch session rename session	screen	:quit windows	\$:choose-session command-prompt -I #S "rename-session '%%' tmux

new session				:new
new named session				:new -s foo
switch session			S	:choose-session
rename session		:sessionname foo	\$	command-prompt -I #S "rename-session '%%'"
kill session	C-\	:quit		
		windows		
	screen			tmux
	C-a cmd	command	C-b cmd	command
create new window	c C-c	:screen	с	:new-window
switch to next window	n C-n	:next	n	:next-window
	SPACE			

	22			
new named session				:new -s foo
switch session			s	:choose-session
rename session		:sessionname foo	\$	command-prompt -I #S "rename-session '%%'"
kill session	C-\	:quit		
		windows	in the second se	
	scre	een		tmux
	C-a cmd	command	C-b cmd	command
create new window	c C-c	:screen	С	:new-window
switch to next window	n C-n SPACE	:next	n	:next-window
switch to previous window	BACKSPACE h p C-p	:prev	р	:previous-window
toggle to last window	C-a	:other		
select window n			n	:select-window -t :n
list windows	w C-w	:windows		:list-windows

switch to next window	C-n SPACE	:next	n	:next-window
switch to previous window	BACKSPACE h p C-p	:prev	р	:previous-window
toggle to last window	C-a	:other		
select window n			n	:select-window -t :n
list windows	w C-w	:windows		:list-windows
show current window number and name	N	:number		
renumber current window			. position	:move-window
move current window to another session			. sessname . sessnum:position	
redraw current window	1 c-1	:redisplay	r	:refresh-client
choose window interactively		:windowlist -b	W	:choose-window
rename window	А	:title	,	
select window foo		:select	f foo	
alana averant veindave	6.1	-1-111	0	.1./11 . / . /

close current window C-k :kill :kill-window join window 1 to current window :join-pane -s 1 join region 0 of window 1 to current window :join-pane -s 1.0 regions screen tmux C-a cmd C-b cmd command command split into top and bottom regions 5 :split

move down to next region TAB :focus make regions same height :resize = close current region X :remove close all but current region Q :only clear current region C :clear file is screenlog.NN log region to file :log turn off logging :log off :resize +n make current region n rows taller/shorter :resize -n make current region n rows tall :resize n screen tmux C-a cmd C-b cmd command command split into left and right panes % :split-window -h split into top and bottom panes :split-window switch to next panes 0 :select-pane rotate panes :rotate-window C-o reverse rotate panes :rotate-window -D М-о

arrange panes side-by-side and same width M-1 arrange panes stacked and same height M-2 swap current and previous pane :swap-pane -U swap current and next pane :swap-pane -D change arrangement of panes SPACE :next-layout close current pane :confirm-before kill-pane break current pane into separate window :break-pane :list-panes list panes :display-panes display pane numbers clear current pane :clear-history log pane to file :pipe-pane "cat > /tmp/tmux.log" turn off logging :pipe-pane :resize-pane -L n resize pane left/up n cells :resize-pane -U n paste buffer screen tmux C-b cmd C-a cmd command command C-[enter copy mode :copy [:copy-mode ESC paste most recent buffer]] :paste-buffer list buffers only one buffer # :list-buffers choose buffer to paste interactively :choose-buffer writes to /tmp/screen-exchange: write buffer to file :writebuf path :save-buffer path copies from /tmp/screen-exchange: copy file to buffer :readbuf path :load-buffer path copy mode

command

tmux

on by default:

#[fg=red,bg=yellow]red text, yellow background#[default]

set-option status off

set-option status on

set-option status-left "string"

set-option status-right string

command

cmd

Emacs-style

C-SPACE

screen

cmd

Vim-style

splitonly by default:

caption string "string"

screen first arg is an identifier referenced by the caption string; the second and third argument set the refresh in seconds

caption always

backtick 1 60 60 cmd caption always "%1`"

%Y%m%d

%м.

%D

%C

%c%A

caption splitonly

%{ry}red text, yellow background%{dd}foo

when mark is set:

SPACE

default bindings

always show status bar

customize caption

shell output

month name

24 hr time

12 hr time

color

weekday name

date (YYYYMMDD)

copy from mark to point and exit copy mode

set mark

13	SPACE			
single column movement	also left right arrow h 1	also left right arr C-b C-h	OW C	
single line movement	also down up arrow j k	also down up arrow C-n C-p		
beginning of line	0	C-a		
end of line	\$	С-е		
forward word	e	M-f		
backward word	b	M-b		
page up	C-b	M-v		
page down	C-f	C-v		
beginning of buffer	g	M-<		
end of buffer	G	M->		
search backwards	? phrase	C-r phrase		
search forwards	/ phrase	C-s phrase		
exit copy mode	any unbound key also works: ESC	q		
Customization	<u>sc</u>	<u>creen</u>	<u>tmux</u>	
startup file	~/.screenrc	~/.screenrc		
scrollback length	defscrollback 2000	defscrollback 2000		
set prefix	how to set prefix to C-b, with a second C escape ^B^B	set-option -g prefix C-a		
define key binding	bind		bind-key	
undefine key binding				
set copy/scrollback key binding style	vi bindings by default. When redefining, use vi commands on left of equations: markkeys h=^B:l=^F:\$=^E	emacs by default: setw -g mode-keys vi		
disable startup message	startup_message off			
number windows from one		0 by default:		
			set -g base-index 1	

fully qualified hostname		#H		
hostname	%н	#h		
session name	%S added in 4.1	#S		
current window flag	%F	#F		
current window index	%n	#I		
current pane index	none	#P		
current pane title	none	#T		
window name	%t	#W		
literal % or #	%%	##		
Terminology server client session window region pane how ssh works:				
When a user logs in to a remote host using ssh, the ssh process contacts an sshd process listening on TCP port 22. The sshd process opens up a new TCP port and forks off a copy of itself for communicating with the ssh process. The new port and child process are for the exclusive use of the connection being established.				
The child sand process authenticates the san process, and if it passes it creates a pseudo-terminal. It then forks the remote user's shell which becomes the controlling process for the pseudo-terminal.				
If the network connection is closed, either explicitly by the ssh process or because of a loss of network connectivity, the child sshd process closes the pseudo-terminal, and this in turn causes the shell to exit.				
the SIGHUP problem:				
If the shell had any process groups running when it exits, they are sent a SIGHUP signal followed by a SIGCONT signal. By default processes exit when they receive SIGHUP. This makes it challenging to run long-running jobs on a remote host when the network connection is unreliable.				
A simple solution to the SIGHUP problem is to run each job with nohup. Optionally, shells such as bash and zsh have a disown built-in which can be used on a process that is already running, should the user have neglected to run it with nohup.				
The fish shell when invoking a process in the background with & sets the signal handling state of the process to ignore SIGHUP. It will do the same if the process is suspended with ^Z and then put in the background with bg.				
Multiplexers offer a solution which protects the shell instead of the job. The user doesn't need to remember to run each job with nohup. As an added benefit any state kept by the shell is preserved.				
anyor:				

caption escapes

#(cmd)

To see the output of a shell the user must connect to the multiplexer server with a multiplexer client process. If the multiplexer is being run on a remote machine and the user's connection is lost, the server and its terminals and controlling processes persist, but the client process exits.

server:

user's shell.

across computers.

session:

session.

client:

a different socket; a new server is created for each socket. window: Both Screen and Tmux have entities which they call windows.

The multiplexer server creates pseudo-terminals which are used for running and interacting with programs.

Screen and Tmux servers can create multiple pseudo-terminals. The controlling process for each pseudo-terminal is the

When multiple client processes connect to the same server they see the same output. This is a way to share a display

Multiplexers support multiple sessions. Each multiplexer session has its own set of terminals and controlling processes

controlling processes in that session. Sessions can be given names to make it easy for the client to choose the correct

Screen launches a separate server process for each session. Screen servers and clients communicate via named pipes.

Tmux by default will only run one server process per user, and this server process can have multiple sessions. A Tmux client and the server communicate via a Unix domain socket in the /tmp directory. The -L option can be used to specify

A Screen window has a single pseudo-terminal and shell. A Tmux window can have multiple pseudo-terminals and shells.

which it is running. The client must choose a session to attach to, and will only be able to see the output of the

region: Screen can divide the viewport into multiple regions. Screen regions can be empty or they can contain a window. The same window can be displayed in more than one region.

Screen regions are stacked on top of each other and extend the full width of the window.

Screen windows can share the viewport. The Tmux viewport can only display one window at a time.

Both Screen and Tmux windows are numbered starting from zero.

When regions share a window their content is identical.

pane: Tmux can divide windows into multiple panes.

Tmux panes can be moved between windows. command character (prefix):

Multiplexers pass most input on to the shell in the region with focus, but a special command character is used to send

Tmux windows can be divided both horizontally and vertically into panes. Each division can be subdivided further.

Tmux panes contain a single pseudo-terminal with a shell, and each pseudo-terminal and shell belongs to only one pane.

The default command character in Screen is C-a. The keystrokes which follow C-a are interpreted by Screen instead of being passed to the shell. Tmux calls the command character the prefix and the default value is C-b. The keystrokes following the prefix are

interpreted by Tmux instead of being passed to shell.

Tmux panes are numbered.

commands to the multiplexer.

a common paste buffer history.

Screen and Tmux keep a history of the output of each shell. The maximum length of the output in lines is configurable. Screen calls the history the scrollback buffer.

passed to the shell.

shell including output that may have scrolled off the top of the region.

scrollback buffer (history): copy/scrollback mode (copy mode):

Screen and Tmux support two modes for each region. In default mode, input which is not intercepted by the multiplexer is When the region is in copy mode the region behaves like a read-only buffer of an editor. The contents are the output of the

The Tmux calls copy/scrollback mode simply copy mode. The keybindings are by default Emacs-style. paste buffer: Screen has a single paste buffer.

The keybindings used by Screen in copy/scrollback mode are Vim-style. It is possible to customize them to be Emacs-

Tmux has multiple paste buffers. The Tmux paste buffers are numbered; the most recent is number zero. Sessions share

off. The status line contains information from Tmux which can be customized. issue tracker | content of this page licensed under creative commons attribution-sharealike 3.0

caption (status line): When a Screen window is split into multiple regions, a caption line is placed at the bottom of each region. When a window contains a single region, Screen by default does not display a caption. The caption, when present, contains information from Screen. The information that is displayed can be customized. Tmux calls the line at the bottom of a window the status line. By default it is always displayed, though it can be turned