Command: **elvis – clone of the Berkeley vi editor**
Syntax: **elvis** [**–Rerv**] [**–t** *tag*] [*file*] ...
Flags: **–R** Set the read-only option
–e Start up emulating *ex*
–r Tell the user to use *elvrec* instead
–t Start editing at the given tag
–v Start up emulating *vi*
Examples: elvis # Call the editor
elvis prog.c # edit *prog.c*

Elvis is a full-screen editor closely modeled on the famous Berkeley *vi* editor. It provides essentially the same interface to the user as *vi*, but the code is completely new, written from scratch. This document provides a brief introduction to *vi*. It is not intended as a tutorial for beginners. Most books on UNIX cover *vi*.

Like *vi*, *elvis* can operate as a screen editor (*vi* mode) or as a line editor (*ex*) mode. It can be called either as *elvis vi*,or as *ex*, depending on which is desired. They are all links to the same file.

Vi Commands

Below is a list of the *vi* commands supported. The following symbols are used in the table:

count	Integer parameter telling how many or how much
key	One character parameter to the command
inp	Interactive input expected
mv	Indicates how much for commands like *delete* and *change*: (Previous sentence) Next sentence { Previous paragraph } Next paragraph (delimited by blank line, *.PP*, *.LP*, *.IP* etc.) [Previous section (delimited by *.SH* or *.NH*) A repeated command character means the scope is this line
MOVE	Indicates commands that may also be used where *mv* is specified
EDIT	These commands affect text and may be repeated by the . command

In addition to the above notation, the caret (^) is used as an abbreviation for CTRL. For example, ^A means CTRL-A.

Count	Command	Description	Type
	^A	(Not defined)	
	^B	Move toward the top of the file by 1 screenful	
	^C	(Not defined)	
count	^D	Scroll down *count* lines (default 1/2 screen)	
count	^E	Scroll up *count* lines	

	^F	Move toward the bottom of the file by 1 screenful	
	^G	Show file status, and the current line	
count	^H	Move left, like *h*	MOVE
	^I	(Not defined)	
count	^J	Move down	MOVE
	^K	(Not defined)	
	^l	Redraw the screen	
count	^M	Move to the front of the next line	MOVE
count	^N	Move down	MOVE
	^O	(Not defined)	
count	^P	Move up	MOVE
	^Q	(Not defined)	
	^R	Redraw the screen	
	^S	(Not defined)	
	^T	(Not defined)	
count	^U	Scroll up *count* lines (default 1/2 screen)	
	^V	(Not defined)	
	^W	(Not defined)	
	^X	(Not defined)	
count	^Y	Scroll down *count* lines	
	^Z	(Not defined)	
	ESC	(Not defined)	
	^\	(Not defined)	
	^]	If the cursor is on a tag name, go to that tag	
	^^	Save this file and edit the previous file	
	^_	(Not defined)	
count	SPACE	Move right,like *l*	MOVE
	! mv	Run the selected lines thru an external filter program	
	" key	Select which cut buffer to use next	
	#	(Not defined)	
	$	Move to the rear of the current line	MOVE
	%	move to the matching (){}[] character	MOVE
	&	(Not defined)	
	’ key	Move to a marked line	MOVE
count	(	Move backward *count* sentences	MOVE
count	)	Move forward *count* sentences	MOVE
	*	(Not defined)	
count	+	Move to the front of the next line	MOVE
count	,	Repeat the previous [*fFtT*] but the other way	MOVE
count	–	Move to the front of the preceding line	MOVE
	.	Repeat the previous ‘‘edit’’ command	
	/	Text search forward for a given regular expr	MOVE
	0	If not part of count, move to 1st char of this line	MOVE

	1	Part of count	
	2	Part of count	
	3	Part of count	
	4	Part of count	
	5	Part of count	
	6	Part of count	
	7	Part of count	
	8	Part of count	
	9	Part of count	
	:	Text. Run single *ex* cmd	
count	;	Repeat the previous [fFtT] cmd	MOVE
count	< mv	Shift text left	EDIT
	=	(Not defined)	
count	> mv	Shift text right	EDIT
	? text	Search backward for a given regular expression	MOVE
	@	(Not defined)	
count	A inp	Append at end of the line	EDIT
count	B	Move back Word	MOVE
	C inp	Change text from cursor through end of line	EDIT
	D	Delete text from cursor through end of line	EDIT
count	E	Move end of Word	MOVE
count	F key	Move leftward to a given character	MOVE
count	G	Move to line #*count* (default is the bottom line)	MOVE
count	H	Move to home row (the line at the top of the screen)	
count	I inp	Insert at the front of the line (after indents)	EDIT
count	J	Join lines, to form one big line	EDIT
	K	Look up keyword	
count	L	Move to last row (the line at the bottom of the screen)	
	M	Move to middle row (the line in the middle)	
	N	Repeat previous search, but the opposite way	MOVE
count	O inp	Open up a new line above the current line	EDIT
	P	Paste text before the cursor	
	Q	Quit to EX mode	
	R inp	Overtype	EDIT
count	S inp	Change lines, like *count*cc	
count	T key	Move leftward *almost* to a given character	MOVE
	U	Undo all recent changes to the current line	
	V	(Not defined)	
count	W	Move forward *count* Words	MOVE
count	X	Delete the character(s) to the left of the cursor	EDIT
count	Y	Yank text line(s) (copy them into a cut buffer)	
	Z Z	Save the file & exit	
	[[	Move back 1 section	MOVE

	\	(Not defined)	
	]]	Move forward 1 section	MOVE
	^	Move to the front of the current line (after indent)	MOVE
	_	(Not defined)	
	‘ key	Move to a marked character	MOVE
count	a inp	Insert text after the cursor	EDIT
count	b	Move back *count* words	MOVE
	c mv	Change text	EDIT
	d mv	Delete text	EDIT
count	e	Move forward to the end of the current word	MOVE
count	f key	Move rightward to a given character	MOVE
	g	(Not defined)	
count	h	Move left	MOVE
count	i inp	Insert text at the cursor	EDIT
count	j	Move down	MOVE
count	k	Move up	MOVE
count	l	Move right	MOVE
	m key	Mark a line or character	
	n	Repeat the previous search	MOVE
count	o inp	Open a new line below the current line	EDIT
	p	Paste text after the cursor	
	q	(Not defined)	
count	r key	Replace *count* chars by a given character	EDIT
count	s inp	Replace *count* chars with text from the user	EDIT
count	t key	Move rightward *almost* to a given character	MOVE
	u	Undo the previous edit command	
	v	(Not defined)	
count	w	Move forward *count* words	MOVE
count	x	Delete the character that the cursor’s on	EDIT
	y mv	Yank text (copy it into a cut buffer)	
	z key	Scroll current line to the screen’s +=top -=bottom .=middle	
count	{	Move back *count* paragraphs	MOVE
count	|	Move to column *count* (the leftmost column is 1)	
count	}	Move forward *count* paragraphs	MOVE
count	~	Switch a character between upper & lower case	EDIT
	DEL	(Not defined)	

Ex Commands

Below is a list of the *ex* commands supported. All can be abbreviated.

General

[line]	append	
	args	[files]
	cd	[directory]
	chdir	[directory]
[line][,line]	change	
[line][,line]	copy	line
[line][,line]	debug[!]	
[line][,line]	Delete	[’’x]
	edit[!]	[file]
	ex[!]	[file]
	file	
[line][,line]	global	/regexp/ command
[line]	Insert	
[line][,line]	join	
[line][,line]	list	
	map[!]	key mapped_to
[line]	mark	x
	mkexrc	
[line][,line]	Move	line
	next[!]	[files]
	Next[!]	
	previous[!]	
[line][,line]	print	
[line]	put	[’’x]
	quit[!]	
[line]	read	file
	rewind[!]	
	set	[options]
[line][,line]	substitute	/regexp/replacement/[p][g]
	tag[!]	tagname
[line][,line]	to	line
	Undo	
	unmap[!]	key
	validate[!]	
	version	
[line][,line]	vglobal	/regexp/ command
	visual	
	wq	
[line][,line]	write[!]	[[>>]file]
	xit[!]	

[line][,line]	yank	[’’x]
[line][,line]	!	command
[line][,line]	<	
[line][,line]	=	
[line][,line]	>	

Text Entry

[line]	append
[line][,line]	change [’’x]
[line]	Insert

The (a)ppend command inserts text after the specified line.

The (i)nsert command inserts text before the specified line.

The (c)hange command copies the range of lines into a cut buffer, deletes them, and inserts new text where the old text used to be.

For all of these commands, you indicate the end of the text you’re inserting by hitting ^D or by entering a line which contains only a period.

Cut & Paste

[line][,line]	Delete [’’x]
[line][,line]	yank [’’x]
[line]	put[!] [’’x]
[line][,line]	copy line
[line][,line]	to line
[line][,line]	Move line

The (d)elete command copies the specified range of lines into a cut buffer, and then deletes them.

The (y)ank command copies the specified range of lines into a cut buffer, but does *not* delete them.

The (pu)t command inserts text from a cut buffer after the specified line—or before it if the ! is present.

The (co)py and (t)o commands yank the specified range of lines and then

immediately paste them after some other line.

The (m)ove command deletes the specified range of lines and then immediately pastes them after some other line. If the destination line comes after the deleted text, then it will be adjusted automatically to account for the deleted lines.

Displaying Text

[line][,line] print
[line][,line] list

The (p)rint command displays the specified range of lines.

The (l)ist command also displays them, but it is careful to make control characters visible.

Global Operations

[line][,line] global /regexp/ command
[line][,line] vglobal /regexp/ command

The (g)lobal command searches through the lines of the specified range (or through the whole file if no range is specified) for lines that contain a given regular expression. It then moves the cursor to each of these lines and runs some other command on them.

The (v)global command is similar, but it searches for lines that *do not* contain the regular expression.

Line Editing

[line][,line] join
[line][,line] ! program
[line][,line] <
[line][,line] >
[line][,line] substitute /regexp/replacement/[p][g]

The (j)oin command concatenates all lines in the specified range together to form one big line. If only a single line is specified, then the following line is catenated onto it.

The ! command runs an external filter program, and feeds the specified range of

lines to it's stdin. The lines are then replaced by the output of the filter. A typical example would be ‘‘:'a,'z!sort -n'' to sort the lines 'a,'z according to their numeric values.

The < and > commands shift the specified range of lines left or right, normally by the width of 1 tab character. The ‘‘shiftwidth'' option determines the shifting amount.

The (s)ubstitute command finds the regular expression in each line, and replaces it with the replacement text. The ‘‘p'' option causes the altered lines to be printed, and the ‘‘g'' option permits all instances of the regular expression to be found & replaced. (Without ‘‘g'', only the first occurrence is replaced.)

Undo

	undo

The (u)ndo command restores the file to the state it was in before your most recent command which changed text.

Configuration & Status

	map[!] [key mapped_to]
	unmap[!] key
	set [options]
	mkexrc
[line]	mark x
	visual
	version
[line][,line]	=
	file

The (ma)p command allows you to configure *elvis* to recognize your function keys, and treat them as though they transmitted some other sequence of characters. Normally this mapping is done only when in the visual command mode, but with the [!]present it will map keys under all contexts. When this command is given with no arguments, it prints a table showing all mappings currently in effect. When called with two arguments, the first is the sequence that your function key really sends, and the second is the sequence that you want *elvis* to treat it as having sent.

The (unm)ap command removes key definitions that were made via the map

command.

The (se)t command allows you examine or set various options. With no arguments, it displays the values of options that have been changed. With the single argument ‘‘all’’ it displays the values of all options, regardless of whether they’ve been explicitly set or not. Otherwise, the arguments are treated as options to be set.

The (mk)exrc command saves the current configuration to a file called *.exrc* in the current directory.

The mar(k) command defines a named mark to refer to a specific place in the file. This mark may be used later to specify lines for other commands.

The (vi)sual command puts the editor into visual mode. Instead of emulating ex, *elvis* will start emulating vi.

The (ve)rsion command tells you that what version of *elvis* this is.

The = command tells you what line you specified, or, if you specified a range of lines, it will tell you both endpoints and the number of lines included in the range.

The file command tells you the name of the file, whether it has been modified, the number of lines in the file, and the current line number.

Multiple Files

args [files]
next[!] [files]
Next[!]
previous[!]
rewind[!]

When you invoke *elvis* from your shell’s command line, any filenames that you give to *elvis* as arguments are stored in the args list. The (ar)gs command will display this list, or define a new one.

The (n)ext command switches from the current file to the next one in the args list. You may specify a new args list here, too.

The (N)ext and (pre)vious commands (they’re really aliases for the same command) switch from the current file to the preceding file in the args list.

The (rew)ind command switches from the current file to the first file in the args list.

Switching Files

edit[!] [file]
tag[!] tagname

The (e)dit command allows to switch from the current file to some other file. This has nothing to do with the args list, by the way.

The (ta)g command looks up a given tagname in a file called ‘‘tags". This tells it which file the tag is in, and how to find it in that file. *Elvis* then switches to the tag’s file and finds the tag.

Exiting

quit[!]
wq
xit

The (q)uit command exits from the editor without saving your file.

The (wq) and (x)it commands (really two names for the same command) both write the file before exiting.

File I/O

[line]	read file
[line][,line]	write[!][[>>]file]

The (r)ead command gets text from another file and inserts it after the specified line.

The (w)rite command writes the whole file, or just part of it, to some other file. The !, if present, will permit the lines to be written even if you’ve set the readonly option. If you precede the filename by >> then the lies will be appended to the file.

Directory

cd [directory]
chdir [directory]
shell

The (cd) and (chd)ir commands (really two names for one command) switch the current working directory.

The (sh)ell command starts an interactive shell.

Debugging

[line][,line] debug[!]
validate[!]

These commands are only available if you compile *elvis* with the **-DDEBUG** flag.

The de(b)ug command lists stats for the blocks which contain the specified range of lines. If the ! is present, then the contents of those blocks is displayed, too.

The (va)lidate command checks certain variables for internal consistency. Normally it does not output anything unless it detects a problem. With the !, though, it will always produce *some* output.

Extensions

In addition to the standard commands, a variety of extra features are present in *elvis* that are not present in *vi*. They are described below.

.exrc
Elvis first runs a *.exrc* file (if there is one) from your $HOME directory. After that, it runs a *.exrc* (if there is one) from the current directory. The one in the current directory may override settings made by the one in the $HOME directory.

:mkexrc
:mk
This EX command saves the current :set and :map configurations in the ‘‘.exrc’’ file in your current directory.

:args
:ar

You can use the :args command to define a new args list, as in:

```
:args *.h
```

After you have defined a new args list, the next time you issue a :next command *elvis* will switch to the first file of the new list.

:Next
:previous
:N
:pre

These commands move backwards through the args list.

zz

In VI, the (lowercase) ‘‘zz’’ command will center the current line on the screen, like ‘‘z="

.

The default count value for . is the same as the previous command which . is meant to repeat. However, you can supply a new count if you wish. For example, after ‘‘3dw’’, ‘‘.’’ will delete 3 words, but ‘‘5.’’ will delete 5 words.

"

The text which was most recently input (via a ‘‘cw’’ command, or something similar) is saved in a cut buffer called ". (which is a pretty hard name to write in an English sentence). You can use this with the ‘‘p’’ or ‘‘P’’ commands thusly:

```
".p
```

K

You can move the cursor onto a word and press shift-K to have *elvis* run a reference program to look that word up. This command alone is worth the price of admission! See the ctags and ref programs.

input

You can backspace back past the beginning of the line. If you type CTRL-A, then the text that you input last time is inserted. You will remain in input mode, so you can backspace over part of it, or add more to it. (This is sort of like CTRL-@ on the real vi, except that CTRL-A really works.)

Real *vi* can only remember up to 128 characters of input, but *elvis* can remember any amount.

:set charattr
:se ca

Elvis can display ‘‘backslash-f’’ style character attributes on the screen as you edit. The following example shows the recognized attributes:

normal **boldface** *italics*

NOTE: you must compile *elvis* without the –DSET_NOCHARATTR flag for this to work.

Omissions

A few *vi* features are missing. The replace mode is a hack. It does not save the text that it overwrites.

Long lines are displayed differently—where the real vi would wrap a long line onto several rows of the screen, *elvis* simply displays part of the line, and allows you to scroll the screen sideways to see the rest of it.

The ‘‘:preserve’’ and ‘‘:recover’’ commands are missing, as is the **–r** flag. ‘‘:Preserve" is practically never used and since use of ‘‘:recover*(CQ is so rare, it was decided to implement it as a separate program. There’s no need to load the recovery code into memory every time you edit a file.

LISP support is missing. The ‘‘@’’ and ‘‘:@’’ commands are missing. You cannot APPEND to a cut buffer.

Options

A variety of options can be set as described below:

Name	Abbr	Type	Default	Description
autoindent	as	Bool	FALSE	autoindent during input?
autowrite	aw	Bool	FALSE	write file for :n command?
charattr	ca	Bool	FALSE	display bold & underline chars?
columns	co	Number	80	width of screen, in characters
directory	dir	String	/usr/tmp	where tmp files are kept
errorbells	eb	Bool	TRUE	ring bell on error?
exrefresh	er	Bool	TRUE	EX mode calls write() often?
ignorecase	ic	Bool	FALSE	searches: upper/lowercase OK?
keytime	kt	Number	1	allow slow receipt of ESC seq?
keywordprg	kp	String	/usr/bin/ref	program to run for shift-K
lines	ln	Number	25	height of screen, in lines
list	li	Bool	FALSE	show tabs as ‘‘^I’’?
magic	ma	Bool	TRUE	searches: allow metacharacters?

paragraphs	pa	String	PPppPApa	paragraphs start with .PP, etc.
readonly	ro	Bool	FALSE	no file should be written back?
report	re	Number	5	report changes to X lines?
scroll	sc	Number	12	default #lines for ^U and ^D
sections	se	String	SEseSHsh	sections start with .SE, etc.
shell	sh	String	*/bin/sh*	shell program, from environment
shiftwidth	sw	Number	8	width of < or > commands
sidescroll	ss	Number	8	#chars to scroll sideways by
sync	sy	Bool	FALSE	call sync() after each change?
tabstop	ts	Number	8	width of a tab character
term	te	String	"?"	terminal type, from environment
vbell	vb	Bool	TRUE	use visible bell if possible?
warn	wa	Bool	TRUE	warn if file not saved for :!cmd
wrapmargin	wm	Number	0	Insert newline after which col?
wrapscan	ws	Bool	TRUE	searches: wrap at EOF?

autoindent

During input mode, the autoindent option will cause each added line to begin with the same amount of leading whitespace as the line above it. Without autoindent, added lines are initially empty.

autowrite

When you're editing one file and decide to switch to another—via the :tag command, or :next command, perhaps—if your current file has been modified, then *elvis* will normally print an error message and refuse to switch.

However, if the autowrite option is on, then *elvis* will write the modified version of the current file and successfully switch to the new file.

charattr

Many text formatting programs allow you to designate portions of your text to be underlined, italicized, or boldface by embedding the special strings \fU, \fI, and \fB in your text. The special string \fR marks the end of underlined or boldface text.

Elvis normally treats those special strings just like any other text. However, if the *charattr* option is on, then *elvis* will interpret those special strings correctly, to display underlined or boldface text on the screen. (This only works, of course, if your terminal can display underlined and boldface, and if the TERMCAP entry says how to do it.)

columns

This is a ''read only'' option. You cannot change its value, but you can have

elvis print it. It shows how wide your screen is.

directory

Elvis uses temporary files to store changed text. This option allows you to control where those temporary files will be. Ideally, you should store them on in fast non-volatile memory, such as a hard disk.

This option can only be set in the ".exrc" file.

errorbells

Normally, *elvis* will ring your terminal's bell if you make an error. However, in noerrorbells mode, your terminal will remain silent.

exrefresh

The EX mode of *elvis* writes many lines to the screen. You can make *elvis* either write each line to the screen separately, or save up many lines and write them all at once.

The exrefresh option is normally on, so each line is written to the screen separately.

You may wish to turn the exrefresh option off (:se noer) if the ''write'' system call is costly on your machine, or if you're using a windowing environment. (Windowing environments scroll text a lot faster when you write many lines at once.)

This option has no effect in *vi* mode.

ignorecase

Normally, when *elvis* searches for text, it treats uppercase letters as being different for lowercase letters.

When the ignorecase option is on, uppercase and lowercase are treated as equal.

keytime

The arrow keys of most terminals send a multi-character sequence. It takes a measurable amount of time for these sequences to be transmitted. The keytime option allows you to control the maximum amount of time to allow for an arrow key (or other mapped key) to be received in full.

The default keytime value is 2. Because of the way UNIX timekeeping works, the actual amount of time allowed will vary slightly, but it will always be between 1 and 2 seconds.

If you set keytime to 1, then the actual amount of time allowed will be between 0 and 1 second. This will generally make the keyboard's response be a little faster (mostly for the ESC key), but on those occasions where the time allowed happens to be closer to 0 than 1 second, *elvis* may fail to allow enough time for an arrow key's sequence to be received fully. Ugh.

As a special case, you can set keytime to 0 to disable this time limit stuff altogether. The big problem here is: If your arrow keys' sequences start with an ESC, then every time you hit your ESC key *elvis* will wait... and wait... to see if maybe that ESC was part of an arrow key's sequence.

NOTE: this option is a generalization of the timeout option of the real vi.

keywordprg

Elvis has a special keyword lookup feature. You move the cursor onto a word, and hit shift-K, and *elvis* uses another program to look up the word and display information about it.

This option says which program gets run. It should contain the full pathname of the program; your whole execution path is *not* checked.

The default value of this option is */usr/bin/ref*, which is a program that looks up the definition of a function in C. It looks up the function name in a file called ‘‘refs’’ which is created by ctags.

You can substitute other programs, such as an English dictionary program or the online manual. *elvis* runs the program, using the keyword as its only argument. The program should write information to stdout. The program's exit status should be 0, unless you want *elvis* to print ‘‘<<< failed >>>".

lines

This ‘‘read only’’ option shows how many lines you screen has.

list

Normally (in ‘‘nolist" mode) *elvis* will expand tabs to the proper number of spaces on the screen, so that the file appears the same would it would be if you printed it or looked at it with *more*.

Sometimes, though, it can be handy to have the tabs displayed as ‘‘^I". In ‘‘list" mode, *elvis* does this, and also displays a ‘‘$" after the end of the line.

magic

The search mechanism in *elvis* can accept ‘‘regular expressions’’—strings in

which certain characters have special meaning. The magic option is normally on, which causes these characters to be treated specially. If you turn the magic option off (:se noma), then all characters except ^ and $ are treated literally. ^ and $ retain their special meanings regardless of the setting of magic.

paragraphs

The { and } commands move the cursor forward or backward in increments of one paragraph. Paragraphs may be separated by blank lines, or by a ‘‘dot’’ command of a text formatter. Different text formatters use different ‘‘dot’’ commands. This option allows you to configure *elvis* to work with your text formatter.

It is assumed that your formatter uses commands that start with a ".’’ character at the front of a line, and then have a one- or two-character command name.

The value of the paragraphs option is a string in which each pair of characters is one possible form of your text formatter’s paragraph command.

readonly

Normally, *elvis* will let you write back any file to which you have write permission. If you do not have write permission, then you can only write the changed version of the file to a *different* file.

If you set the readonly option, then *elvis* will pretend you do not have write permission to *any* file you edit. It is useful when you really only mean to use *elvis* to look at a file, not to change it. This way you cannot change it accidentally.

This option is normally off, unless you use the ‘‘view’’ alias of *elvis*. ‘‘View’’ is like *vi* except that the readonly option is on.

report

Commands in *elvis* may affect many lines. For commands that affect a lot of lines, *elvis* will output a message saying what was done and how many lines were affected. This option allows you to define what ‘‘a lot of lines’’ means. The default is 5, so any command which affects 5 or more lines will cause a message to be shown.

scroll

The CTRL-U and CTRL-D keys normally scroll backward or forward by half a screenful, but this is adjustable. The value of this option says how many lines those keys should scroll by.

sections

The [[and]] commands move the cursor backward or forward in increment of 1 section. Sections may be delimited by a { character in column 1 (which is useful for C source code) or by means of a text formatter's ''dot'' commands.

This option allows you to configure *elvis* to work with your text formatter's ''section'' command, in exactly the same way that the paragraphs option makes it work with the formatter's ''paragraphs" command.

shell

When *elvis* forks a shell (perhaps for the :! or :shell commands) this is the program that is uses as a shell. This is */bin/sh* by default, unless you have set the SHELL environment variable, it which case the default value is copied from the environment.

shiftwidth

The < and > commands shift text left or right by some uniform number of columns. The shiftwidth option defines that uniform number. The default is 8.

sidescroll

For long lines, *elvis* scrolls sideways. (This is different from the real *vi*, which wraps a single long line onto several rows of the screen.) To minimize the number of scrolls needed, *elvis* moves the screen sideways by several characters at a time. The value of this option says how many characters' widths to scroll at a time. Generally, the faster your screen can be redrawn, the lower the value you will want in this option.

sync

If the system crashes during an edit session, then most of your work can be recovered from the temporary file that *elvis* uses to store changes. However, sometimes MINIX will not copy changes to the hard disk immediately, so recovery might not be possible. The [no]sync option lets you control this. In nosync mode (which is the default), *elvis* lets the operating system control when data is written to the disk. This is generally faster. In sync mode, *elvis* forces all changes out to disk every time you make a change. This is generally safer, but slower.

tabstop

Tab characters are normally 8 characters wide, but you can change their widths by means of this option.

term

This ''read only'' option shows the name of the termcap entry that *elvis* is using for your terminal.

vbell

If your termcap entry describes a visible alternative to ringing your terminal's bell, then this option will say whether the visible version gets used or not. Normally it will be.

If your termcap does NOT include a visible bell capability, then the vbell option will be off, and you cannot turn it on.

warn

Elvis will normally warn you if you run a shell command without saving your changed version of a file. The ‘‘nowarn" option prevents this warning.

wrapmargin

Normally (with wrapmargin=0) *elvis* will let you type in extremely long lines, if you wish. However, with wrapmargin set to something other that 0 (wrapmargin=65 is nice), *elvis* will automatically cause long lines to be ‘‘wrapped" on a word break for lines longer than wrapmargin's setting.

wrapscan

Normally, when you search for something, *elvis* will find it no matter where it is in the file. *elvis* starts at the cursor position, and searches forward. If *elvis* hits EOF without finding what you're looking for, then it wraps around to continue searching from line 1.

If you turn off the wrapscan option (:se nows), then when *elvis* hits EOF during a search, it will stop and say so.

Cflags

Elvis uses many preprocessor symbols to control compilation. Most of these flags allow you to disable small sets of features. MINIX-ST users will probably want all features enabled, but MINIX-PC users will have to disable one or two feature sets because otherwise *elvis* would be too large to compile and run.

These symbols can be defined via flags passed to the compiler. The best way to do this is to edit the Makefile, and append the flag to the ‘‘CFLAGS=’’ line. After you do that, you must recompile elvis completely by saying

```
make clean
make
```

–DM_SYSV

This flag causes *elvis* to use System-V ioctl() calls for controlling your terminal;

normally it uses v7/BSD/MINIX ioctl() calls.

–DDATE
The symbol DATE should be defined to look like a string constant, giving the date when *elvis* was compiled. This date is reported by the ‘‘:version’’ command.

You can also leave DATE undefined, in which case ‘‘:version’’ will not report the compilation date.

–DCRUNCH
This flag causes several large often-used macros to be replaced by equivalent functions. This saves about 4K of space in the ‘‘.text’’ segment, and it does not cost you any features.

–DDEBUG
This adds many internal consistency checks and the ‘‘:debug’’ and ‘‘:validate’’ commands. It increases the size of ‘‘text’’ by about 5K bytes.

–DNO_CHARATTR
This permanenently disables the ‘‘charattr’’ option. It reduces the size of ‘‘.text’’ by about 850 bytes.

–DNO_RECYCLE
Normally, *elvis* will recycle space in the temporary file which contains totally obsolete text. The **–DNO_RECYCLE** option disables this, making your ‘‘.text’’ segment smaller by about 1K but also permitting the temporary file to grow very quickly. If you have less than two megabytes of free space on your disk, then do not even consider using this flag.

–DNO_SENTENCE
This leaves out the ‘‘(’’ and ‘‘)’’ visual commands, and removes the code that allows the ‘‘[[’’, ‘‘]]’’, ‘‘{’’, and ‘‘}’’ commands to recognize nroff macros. The ‘‘[[’’ and ‘‘]]’’ commands will still move to the start of the previous/next C function source code, though, and ‘‘{’’ and ‘‘}’’ will move to the previous/next blank line. This saves about 650 bytes from the ‘‘.text’’ segment.

–DNO_CHARSEARCH
This leaves out the visual commands which locate a given character in the current line: ‘‘f’’, ‘‘t’’, ‘‘F’’, ‘‘T’’, ‘‘;’’, and ‘‘,’’. This saves about 900 bytes.

–DNO_EXTENSIONS
This leaves out the ‘‘:mkexrc’’ command, and the ‘‘K’’ and ‘‘#’’ visual

commands. Other extensions are either inherent in the design of *elvis*, or are too tiny to be worth removing. This saves about 500 bytes.

–DNO_MAGIC
This permanently disables the ‘‘magic’’ option, so that most meta-characters in a regular expression are not recognized. This saves about 3K bytes from the ‘‘.text’’ segment.

Termcap

Elvis can use standard termcap entries, but it also recognizes and uses several extra capabilities, if you give them. All of these are optional.

Capability	Description
:PU=:	sequence received from the <PgUp> key
:PD=:	sequence received from the <PgDn> key
:HM=:	sequence received from the <Home> key
:EN=:	sequence received from the <End> key
:VB=:	sequence sent to start bold printing
:Vb=:	sequence sent to end bold printing

Author

Elvis was written by Steve Kirkendall. He can be reached by email at: kirkenda@cs.pdx.edu for comments regarding *elvis*.