

ELVIS

- a clone of vi/ex -
version 1.8pl3

Hurry down doomsday, the bugs are taking over! – E.C.

Author: Steve Kirkendall
1500 SW Park #326
Portland OR, 97201 E-Mail: kirkenda@cs.pdx.edu
Phone: (503) 643-6980

TABLE OF CONTENTS

INTRODUCTION	1
What ELVIS does, Copyright, How to compile ELVIS, Overview	
VISUAL MODE COMMANDS	2
Normal interactive editing, Input mode, Arrow keys, Digraphs, Abbreviations, Auto-indentation	
COLON MODE COMMANDS	3
Line specifiers, Text entry, Cut & paste, Display text, Global operations, Line editing, Undo, Configuration & status, Multiple files, Switching files, Working with a compiler, Exiting, File I/O, Directory & shell, Debugging	
REGULAR EXPRESSIONS	4
Syntax, Options, Substitutions, Examples	
OPTIONS	5
Autoindent, Autoprint, etc.	
INITIALIZATION	6
Start-up initialization, File initialization, The :mkexrc command, Other techniques	
CUT BUFFERS	7
Putting text into a cut buffer, Pasting from a cut buffer, Macros, The effect of switching files	
DIFFERENCES BETWEEN ELVIS AND THE REAL VI/EX	8
INTERNAL	9
For programmers only, The temporary file, Implementation of editing, Marks and the cursor, Colon command interpretation, Screen control, Portability	
MAKEFILE	10
CFLAGS	11
TERMCAP	12
ENVIRONMENT VARIABLES	13
VERSIONS	14
QUESTIONS & ANSWERS	15

Traditional-style "man" pages appear at the end of this manual.

1. Introduction

ELVIS is a clone of *vi/ex*, the standard editor on open systems. ELVIS supports nearly all of the *vi/ex* commands, in both visual mode and colon mode.

Like *vi/ex*, ELVIS stores most of the text in a temporary file, instead of RAM. This allows it to edit files that are too large to fit in a single process' data space. Also, the edit buffer can survive a power failure or crash.

ELVIS runs under INTERIX, BSD, AT&T SysV, Minix, MS-DOS, Atari TOS, Coherent, OS9/68000, VMS, AmigaDos, and OS/2. The next version is also expected to add MS-Windows and MacOS. Contact me before you start porting it to some other OS, because somebody else may have already done it for you.

ELVIS is freely redistributable, in either source form or executable form. There are no restrictions on how you may use it.

1.1 Overview of ELVIS

The user interface of ELVIS/*vi/ex* is weird. There are two major command modes in ELVIS, and a few text input modes as well. Each command mode has a command which allows you to switch to the other mode.

You will probably use the *visual command mode* most of the time. This is the mode that ELVIS normally starts up in.

In visual command mode, the entire screen is filled with lines of text from your file. Each keystroke is interpreted as part of a visual command. If you start typing text, it will *not* be inserted, it will be treated as part of a command. To insert text, you must first give an "insert text" command. This will take some getting used to. (An alternative exists. Lookup the "inputmode" option.)

The *colon mode* is quite different. ELVIS displays a ":" character on the bottom line of the screen, as a prompt. You are then expected to type in a command line and hit the <Return> key. The set of commands recognized in the colon mode is different from visual mode's.

2. Visual Mode Commands

Most visual mode commands are one keystroke long. The following table lists the operation performed by each keystroke, and also denotes any options or arguments that it accepts. Notes at the end of the table describe the notation used in this table.

In addition to the keys listed here, your keyboard's "arrow" keys will be interpreted as the appropriate cursor movement commands. The same goes for <PgUp> and <PgDn>, if your keyboard has them. The <Insert> key will toggle between insert mode and replace mode. There is a colon mode command (**:map**, to be described later) which will allow you to define other keys, such as function keys.

A tip: visual command mode looks a lot like text input mode. If you forget which mode you're in, just hit the <Esc> key. If ELVIS beeps, then you're in visual command mode. If ELVIS does not beep, then you were in input mode, but by hitting <Esc> you will have switched to visual command mode. So, one way or another, after <Esc> ELVIS will be ready for a command.

Command	Description
^A	Search for next occurrence of word at cursor (MOVE)(EXT)
^B	Move toward the top of the file by 1 screenful
^C	— (usually sends SIGINT, to interrupt a command)
<i>cnt</i> ^D	Scroll down <i>cnt</i> lines (default 1/2 screen)
<i>cnt</i> ^E	Scroll up <i>cnt</i> lines
^F	Move toward the bottom of the file by 1 screenful
^G	Show file status, and the current line #
<i>cnt</i> ^H	Move left, like h (MOVE)
^I	—
<i>cnt</i> ^J	Move down (MOVE)
^K	—
^L	Redraw the screen
<i>cnt</i> ^M	Move to the front of the next line (MOVE)
<i>cnt</i> ^N	Move down (MOVE)
^O	—
<i>cnt</i> ^P	Move up (MOVE)
^Q	— (typically XON, which restarts screen updates)
^R	Redraw the screen
^S	— (typically XOFF, which stops screen updates)
^T	Return to source of previous :tag or ^] command.
<i>cnt</i> ^U	Scroll up <i>cnt</i> lines (default 1/2 screen)
^V	—
^W	—
<i>cnt</i> ^X	Move to a physical column number on the screen (MOVE) (EXT)
<i>cnt</i> ^Y	Scroll down <i>cnt</i> lines
^Z	— (sometimes sends SIGSUSP, to suspend execution)
ESC	—
^\	— (usually sends SIGQUIT, which is ignored)
^]	If the cursor is on a tag name, go to that tag
^^	Switch to the previous file, like ":e #"
<i>cnt</i> ^_	Move to a given screen-relative row (MOVE) (EXT)
<i>cnt</i> SPC	Move right, like l (MOVE)

Command	Description
! <i>mv</i>	Run the selected lines thru an external filter program
" <i>key</i>	Select which cut buffer to use next
<i>cnt</i> # +	Increment a number (EDIT) (EXT)
\$	Move to the rear of the current line (MOVE)
<i>cnt</i> %	Move to matching (){}[] or to a given % of file (MOVE) (EXT)
<i>cnt</i> &	Repeat the previous ":s//" command here (EDIT)
' <i>key</i>	Move to a marked line (MOVE)
<i>cnt</i> (	Move backward <i>cnt</i> sentences (MOVE)
<i>cnt</i>)	Move forward <i>cnt</i> sentences (MOVE)
*	Go to the next error in the errlist (EXT)
<i>cnt</i> +	Move to the front of the next line (MOVE)
<i>cnt</i> ,	Repeat the previous [fFtT] but in the other direction (MOVE)
<i>cnt</i> -	Move to the front of the preceding line (MOVE)
<i>cnt</i> .	Repeat the previous "edit" command
/ <i>text</i>	Search forward for a given regular expression (MOVE)
0	If 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
: <i>text</i>	Run single EX cmd
<i>cnt</i> ;	Repeat the previous [fFtT] cmd (MOVE)
< <i>mv</i>	Shift text left (EDIT)
= <i>mv</i>	Reformat
> <i>mv</i>	Shift text right (EDIT)
? <i>text</i>	Search backward for a given regular expression (MOVE)
@ <i>key</i>	Execute the contents of a cut-buffer as VI commands
<i>cnt</i> A <i>inp</i>	Append at end of the line (EDIT)
<i>cnt</i> B	Move back Word (MOVE)
C <i>inp</i>	Change text from the cursor through the end of the line (EDIT)
D	Delete text from the cursor through the end of the line (EDIT)
<i>cnt</i> E	Move end of Word (MOVE)
<i>cnt</i> F <i>key</i>	Move leftward to a given character (MOVE)
<i>cnt</i> G	Move to line # <i>cnt</i> (default is the bottom line) (MOVE)
<i>cnt</i> H	Move to home row (the line at the top of the screen)
<i>cnt</i> I <i>inp</i>	Insert at the front of the line (after indents) (EDIT)
<i>cnt</i> J	Join lines, to form one big line (EDIT)
K	Look up keyword (EXT)
<i>cnt</i> L	Move to last row (the line at the bottom of the screen)
M	Move to middle row
N	Repeat previous search, but in the opposite direction (MOVE)
<i>cnt</i> O <i>inp</i>	Open up a new line above the current line (EDIT)
P	Paste text before the cursor (EDIT)

Command	Description
Q	Quit to EX mode
R inp	Overtime (EDIT)
cnt S inp	Change lines, like <i>cntcc</i>
cnt T key	Move leftward *almost* to a given character (MOVE)
U	Undo all recent changes to the current line
V	Start marking lines for <i>c/d/y/</>/!/^\</i> (EXT)
cnt W	Move forward <i>cnt</i> Words (MOVE)
cnt X	Delete the character(s) to the left of the cursor (EDIT)
cnt Y	Yank text line(s) (copy them into a cut buffer)
Z Z	Save the file & exit
[[	Move back 1 section (MOVE)
\ mv	Pop-up menu for modifying text (EXT)
]]	Move forward 1 section (MOVE)
^	Move to the front of the current line (after indent) (MOVE)
cnt _	Move to the current line
' key	Move to a marked character (MOVE)
cnt a inp	Insert text after the cursor (EDIT)
cnt b	Move back <i>cnt</i> words (MOVE)
c mv	Change text (EDIT)
d mv	Delete text (EDIT)
cnt e	Move forward to the end of the current word (MOVE)
cnt f key	Move rightward to a given character (MOVE)
g	—
cnt h	Move left (MOVE)
cnt i inp	Insert text at the cursor (EDIT)
cnt j	Move down (MOVE)
cnt k	Move up (MOVE)
cnt l	Move right (MOVE)
m key	Mark a line or character
n	Repeat the previous search (MOVE)
cnt o inp	Open a new line below the current line (EDIT)
p	Paste text after the cursor (EDIT)
q	—
cnt r key	Replace <i>cnt</i> chars by a given character (EDIT)
cnt s inp	Replace <i>cnt</i> chars with text from the user (EDIT)
cnt t key	Move rightward *almost* to a given character (MOVE)
u	Undo the previous edit command
v	Start marking characters for <i>c/d/y/</>/!/^\</i> (EXT)
cnt w	Move forward <i>cnt</i> words (MOVE)
cnt 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
cnt {	Move back <i>cnt</i> paragraphs (MOVE)
cnt 	Move to column <i>cnt</i> (the leftmost column is 1)
cnt }	Move forward <i>cnt</i> paragraphs (MOVE)
cnt ~	Switch a character between uppercase & lowercase (EDIT)
DEL	— (usually mapped to shift-X, so it deletes one character)

count/cnt

Many commands may be preceded by a count. This is a sequence of digits representing a decimal number. For most commands that use a count, the command is repeated

<count> times. The count is always optional, and usually defaults to 1.

key Some commands require two keystrokes. The first key always determines which command is to be executed. The second key is used as a parameter to the command.

mv Some commands (! < > **c d y** =) operate on text between the cursor and some other position. There are three ways that you can specify that other position.

The first way is to follow the command keystroke with a movement command. For example, **dw** deletes a single word. "d3w" and "3dw" both delete three words.

The second way is to type the command keystroke twice. This causes whole lines to be acted upon. For example, >> indents the current line. 3>> indents the current line and the following two lines.

The last way is to move the cursor to one end of the text, type '**v**' or '**V**' to start marking, move the cursor to the other end, and then type the desired command key.

inp Many commands allow the user to interactively enter text. See the discussion of "input mode" in the following section.

(EXT) These commands are extensions — the real *vi* doesn't have them.

(EDIT) These commands affect text, and may be repeated by the "." command.

(MOVE)

These commands move the cursor, and may be used to specify the extent of a member of the "mv" class of commands.

2.1 Input Mode

You can't type text into your file directly from visual command mode. Instead, you must first give a command which will put you into input mode. The commands to do this are A/C/I/O/R/S/a/i/o/s.

The **S/s/C/** commands temporarily place a \$ at the end of the text that they are going to change.

In input mode, all keystrokes are inserted into the text at the cursor's position, except for the following:

^A	insert a copy of the last input text
^D	delete one indent character
^H	(backspace) erase the character before the cursor
^L	redraw the screen
^M	(carriage return) insert a newline (^J, linefeed)
^O	execute next key as a visual command (limited!)
^P	insert the contents of the cut buffer
^R	redraw the screen, like ^L
^T	insert an indent character
^U	backspace to the beginning of the line
^V	insert the following keystroke, even if special
^W	backspace to the beginning of the current word
^Z^Z	write the file & exit ELVIS
^I	(ESCape) exit from input mode, back to command mode

Also, on some systems, ^S may stop output, ^Q may restart output, and ^C may interrupt execution. ^@ (the NUL character) cannot be inserted.

The **R** visual command puts you in overtyping mode, which is a slightly different form of input mode. In overtyping mode, each time you insert a character, one of the old characters is deleted

from the file.

2.2 Arrow Keys in Input Mode

The arrow keys can be used to move the cursor in input mode. (This is an extension; the real *vi* doesn't support arrow keys in input mode.) The <PgUp>, <PgDn>, <Home>, and <End> keys work in input mode, too. The <Delete> key deletes a single character in input mode. The <Insert> key toggles between input mode and replace mode.

The best thing about allowing arrow keys to work in input mode is that as long as you're in input mode, ELVIS seems to have a fairly ordinary user interface. With most other text editors, you are always in either insert mode or replace mode, and you can use the arrow keys at any time to move the cursor. Now, ELVIS can act like that, too. In fact, with the new **inputmode** option and the "control-Z control-Z" input command, you may never have to go into visual command mode for simple edit sessions.

2.3 Digraphs

ELVIS supports digraphs as a way to enter non-ASCII characters. A digraph is a character which is composed of two other characters. For example, an apostrophe and the letter i could be defined as a digraph which is to be stored & displayed as an accented i.

There is no single standard for extended ASCII character sets. ELVIS can be compiled to fill the digraph with values appropriate for either the IBM PC character set, or the LATIN-1 character set used by X windows, or neither. You can view or edit the digraph table via the **:digraph** colon command.

Digraphs will not be recognized until you've entered **:set digraph**.

To actually use a digraph type the first character, then hit <Backspace>, and then type the second character. ELVIS will then substitute the non-ASCII character in their place.

2.4 Abbreviations

ELVIS can expand abbreviations for you. You define an abbreviation with the **:abbr** command, and then whenever you type in the abbreviated form while in input mode, ELVIS will immediately replace it with the long form. COBOL programmers should find this useful. :-)

ELVIS doesn't perform the substitution until you type a non-alphanumeric character to mark the end of the word. If you type a control-V before that non-alphanumeric character, then ELVIS will not perform the substitution.

2.5 Auto-Indent

With the **:set autoindent** option turned on, ELVIS will automatically insert leading whitespace at the beginning of each new line that you type in. The leading whitespace is copied from the preceding line.

To add more leading whitespace, type control-T. To remove some whitespace, type control-D.

If you **:set noautotab**, then the whitespace generated by control-T will always consist of spaces — never tabs. Some people seem to prefer this.

ELVIS' autoindent mode isn't 100% compatible with *vi*'s. In ELVIS, 0^D and ^^D don't work, ^U can wipeout all indentation, and sometimes ELVIS will use a different amount of indentation than *vi* would.

3. Colon Mode Commands

To use colon mode commands, you must switch from visual command mode to colon command mode. The visual mode commands to do this are **:** for a single colon command, or **Q** for many colon mode commands.

In general, command lines begin with 0, 1, or 2 line specifiers, followed by a command name, and perhaps some arguments after that.

Lines which don't access the text, such as **:quit**, don't allow any line specifiers. Other commands, such as **:mark**, only allow a single line specifier. Most commands, though, allow two line specifiers; the command is applied to all lines between the two specified lines, inclusive. The table below indicates how many line specifiers each command allows.

Command names can usually be abbreviated; in the table below, the extra part of command names has is enclosed in square brackets. Square brackets are also used to indicate which arguments are optional.

LINES	COMMAND	ARGUMENTS
	ab[br]	[short] [expanded form]
	an[d]	condition
[line]	a[ppend][!]	
	ar[gs]	[files]
	cc	[files]
	cd[!]	[directory]
[line][,line]	c[hange]	
	chd[ir][!]	[directory]
[line][,line]	co[py]	line
	col[or]	[when] [[“light”] color] [“on” color]
[line][,line]	d[elete]	[“x”]
	dig[raph][!]	[XX [Y]]
	e[dit][!]	[file]
	el[se]	commands
	er[rlist][!]	[errlist]
	f[ile]	[file]
[line][,line]	g[lobal]	/regexp/ command
	if	condition
[line]	i[nsert]	
[line][,line]	j[oin][!]	
[line][,line]	l[ist]	
	mak[e]	[target]
	map[!]	key mapped_to
[line]	ma[rk]	[“x”]
	mk[exrc]	
[line][,line]	m[ove]	line
	n[ext][!]	[files]
	N[ext][!]	
[line][,line]	nu[mber]	
	o[r]	condition
	po[p][!]	
[line][,line]	p[rint]	
[line]	pu[t]	[“x”]
	q[uit][!]	
[line]	r[ead]	file
	rew[ind][!]	

	se[t]	[options]
	so[urce]	file
[line][,line]	s[ubstitute]	/regexp/replacement/[p][g][c]
[line][,line]	t	line
	ta[g][!]	tagname
	th[en]	commands
	una[bbr]	[short]
	u[ndo]	
	unm[ap][!]	key
	ve[rsion]	
[line][,line]	v[lobal]	/regexp/ command
	vi[sual]	[filename]
	wq	
[line][,line]	w[rite][!]	[[>>]file]
	x[it][!]	
[line][,line]	y[ank]	[''x]
[line][,line]	!	command
[line][,line]	<	
[line][,line]	=	
[line][,line]	>	
[line][,line]	&	
[line][,line]	@	''x

3.1 Line Specifiers

Line specifiers are always optional. The first line specifier of most commands usually defaults to the current line. The second line specifier usually defaults to be the same as the first line specifier. Exceptions are **:write**, **:global**, and **:vglobal**, which act on all lines of the file by default, and **!:**, which acts on no lines by default.

If you use the visual **V** command to mark a range of lines, and then use the visual **:** command to execute a single ex command, then the default range affected by the ex command will be the visibly marked text.

Line specifiers consist of an absolute part and a relative part. The absolute part of a line specifier may be either an explicit line number, a mark, a dot to denote the current line, a dollar sign to denote the last line of the file, or a forward or backward search.

An explicit line number is simply a decimal number, expressed as a string of digits.

A mark is typed in as an apostrophe followed by a letter. Marks must be set before they can be used. You can set a mark in visual command mode by typing **m** and a letter, or you can set it in colon command mode via the **mark** command.

A forward search is typed in as a regular expression surrounded by slash characters; searching begins at the default line. A backward search is typed in as a regular expression surrounded by question marks; searching begins at the line before the default line.

If you omit the absolute part, then the default line is used.

The relative part of a line specifier is typed as a "+" or "-" character followed by a decimal number. The number is added to or subtracted from the absolute part of the line specifier to produce the final line number.

As a special case, the **%** character may be used to specify all lines of the file. It is roughly equivalent to saying **1,\$**. This can be a handy shortcut.

Some examples:

:p	print the current line
:37p	print line 37
:’gp	print the line which contains mark g
:/foo/p	print the next line that contains "foo"
:\$p	print the last line of the file
:20,30p	print lines 20 through 30
:1,\$p	print all lines of the file
:%p	print all lines of the file
:/foo/-2,+4p	print 5 lines around the next "foo"

3.2 Text Entry Commands

[line] **a**ppend
[line][,line] **c**hange ["x"]
[line] **i**nsert

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.

3.3 Cut & Paste Commands

[line][,line] **d**elete ["x"]
[line][,line] **y**ank ["x"]
[line] **p**ut ["x"]
[line][,line] **c**opy line
[line][,line] **t**o line
[line][,line] **m**ove 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 **p**ut command inserts text from a cut buffer after the specified line.

The **c**opy 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.

3.4 Display Text Commands

[line][,line] **p**rint
[line][,line] **l**ist
[line][,line] **n**umber

The **p**rint command displays the specified range of lines.

The **number** command displays the lines, with line numbers.

The **list** command also displays them, but it is careful to make control characters visible.

3.5 Global Operations Commands

```
[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 **vg**lobal command is similar, but it searches for lines that *don't* contain the regular expression.

3.6 Line Editing Commands

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

The **j**oin command catenates 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 normal "join" inserts one or two spaces between the lines; the "join!" variation (with a '!') doesn't insert spaces.

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" to sort the lines 'a,'z.

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. The "g" option permits all instances of the regular expression to be found & replaced. (Without "g", only the first occurrence in each line is replaced.) The "c" option asks for confirmation before each substitution.

The **&** command repeats the previous substitution command. Actually, "&" is equivalent to "s/~//" with the same options as last time. It searches for the last regular expression that you specified for any purpose, and replaces it with the the same text that was used in the previous substitution.

3.7 Undo Command

```
undo
```

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

3.8 Configuration & Status Commands

```

map[!] [key mapped_to]
unmap[!] key
abbr [word expanded_form_of_word]
unabbr word
digraph[!] [XX [Y]]
set [options]
mkexrc
[line] mark "x
visual
version
[line][,line] =
file [file]
source file
@ "x
color [when] [{"light"} color] [{"on"} color]

```

The **map** 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 input and replace modes as well. 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. As a special case, if the first argument is a '#' sign followed by a number then ELVIS will map the corresponding function key; for example, ":map #7 dd" will cause the <F7> key to delete a line. Also, on some systems, ":map #7s ..." may map <Shift-F7>, ":map #7c ..." may map <Control-F7>, and ":map #7a ..." may map <Alt-F7>.

The **unmap** command removes key definitions that were made via the map command.

The **abbr** command is used to define/list a table of abbreviations. The table contains both the abbreviated form and the fully spelled-out form. When you're in visual input mode, and you type in the abbreviated form, ELVIS will replace the abbreviated form with the fully spelled-out form. When this command is called without arguments, it lists the table; with two or more arguments, the first argument is taken as the abbreviated form, and the rest of the command line is the fully-spelled out form.

The **unabbr** command deletes entries from the abbr table.

The **digraph** command allows you to display the set of digraphs that ELVIS is using, or add/remove a digraph. To list the set of digraphs, use the digraph command with no arguments. To add a digraph, you should give the digraph command two arguments. The first argument is the two ASCII characters that are to be combined; the second is the non-ASCII character that they represent. The non-ASCII character's most significant bit is automatically set by the digraph command, unless to append a ! to the command name. Removal of a digraph is similar to adding a digraph, except that you should leave off the second argument.

The **set** 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 **mkexrc** command saves the current configuration to a file called **.exrc** in the current directory.

The **mark** 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 **visual** command puts the editor into visual mode. Instead of emulating *ex*, ELVIS will start emulating *vi*.

The **version** 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. You can also use it to change the name of the current file.

The **source** command reads a sequence of colon mode commands from a file, and interprets them.

The **@** command executes the contents of a cut-buffer as EX commands.

The **color** command only works under MS-DOS, or if you have an ANSI-compatible color terminal. It allows you to set the foreground and background colors for different types of text: normal, bold, italic, underlined, standout, pop-up menu, and visible selection. By default, it changes the "normal" colors; to change other colors, the first argument to the **:color** command should be the first letter of the type of text you want. The syntax for the colors themselves is fairly intuitive. For example, **:color light cyan on blue** causes normal text to be displayed in light cyan on a blue background, and **:color b bright white** causes bold text to be displayed in bright white on a blue background. The background color always defaults to the current background color of normal text. Your first **:color** command *must* specify both the foreground and background for normal text.

3.9 Conditional Commands

```
if condition
and condition
or condition
then commands
else commands
```

These commands allow ELVIS to execute a set of commands only if a given condition is valid. The **if**, **and**, and **or** commands set or clear a flag, and the **then** and **else** commands test that flag to decide whether to execute their arguments as commands. This can be handy in **.exrc** files.

A condition can test the following types of values:

filetype asterisk, followed by filename extension

constants either a number or a quoted string

options the name of a **:set** option

termcap fields
a two letter name, enclosed in colons

environment variables
the name, preceded by a dollar sign

The condition can involve either one boolean value, two strings compared for equality ("**=**" or "**==**") or inequality ("**!=**"), or two numbers compared with any comparison operator.

The **if** command sets the conditional flag equal to the results of the condition. The **and** command performs a logical AND of the conditional flag and the new condition. The **or** command performs a logical OR of the conditional flag and the new condition.

The **then** command's arguments are one or more commands. (Commands can be delimited by placing a ' | ' character between them.) The commands are executed if the conditional flag is true, or skipped if it is false. Similarly, **else** executes its arguments only if the conditional flag is false.

For example, on my Linux system the console can handle color commands, but xterms can't. To have colors set on the console but not on an xterm, I added the following to my **.exrc** file...

```
if term="console"
then color yellow on blue | color quit white on blue
```

Note: The **.exrc** file is executed before elvis loads the first file, so you can't test for a specific filename there, or modify text, or adjust cut buffers. A new initialization file, **.exfilerc**, is now supported to fill this need. It resides in your home directory. The **.exfilerc** file is executed after each file is loaded. A typical **.exfilerc** file might look like...

```
if *.c
or *.h
and newfile
then !mkskel %
```

3.10 Multiple File Commands

```
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 **args** command will display this list, or define a new one.

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

The **Next** and **previous** commands (they're really aliases for the same command) switch from the current file to the preceding file in the args list.

The **rewind** command switches from the current file to the first file in the args list.

3.11 Switching Files

```
edit[!] [file]
tag[!] tagname
pop[!]
```

The **edit** 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 **tag** 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.

The **pop** command reverses a tag command. It switches back to the file and line number from which you invoked the tag command. Up to 15 tag commands can be reversed; the

filenames and line numbers are saved on a stack, so you can perform multiple tag commands, and then reverse them with multiple pop commands.

3.12 Working with a Compiler

```
cc [files]
make [target]
errlist[!] [errlist]
```

The **cc** and **make** commands execute your compiler or "make" utility and redirect any error messages into a file called **errlist**. By default, cc is run on the current file. (You should write it before running cc.) The contents of the **errlist** file are then scanned for error messages. If an error message is found, then the cursor is moved to the line where the error was detected, and the description of the error is displayed on the status line.

After you've fixed one error, the **errlist** command will move the cursor to the next error. In visual command mode, hitting ***** will do this, too.

You can also create an **errlist** file from outside of ELVIS, and use **elvis -m** to start ELVIS and have the cursor moved to the first error. Note that you don't need to supply a filename with **elvis -m** because the error messages always say which source file an error is in.

Note: When you use **errlist** repeatedly to fix several errors in a single file, it will attempt to adjust the reported line numbers to allow for lines that you have inserted or deleted. These adjustments are made with the assumption that you will work through the file from the beginning to the end.

3.13 Exit Commands

```
quit[!]
wq
xit
```

The **quit** command exits from the editor without saving your file.

The **wq** command writes your file out, then then exits.

The **xit** command is similar to the **wq** command, except that **xit** won't bother to write your file if you haven't modified it.

3.14 File I/O Commands

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

The **read** command gets text from another file and inserts it after the specified line. It can also read the output of a program; simply precede the program name by a **'!'** and use it in place of the file name.

The **write** 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 lines will be appended to the file. You can send the lines to the standard input of a program by replacing the filename with a **'!'** followed by the command and its arguments.

Note: Be careful not to confuse **":w!filename"** and **":w !command"**. To write to a program, you must have at least one blank before the **'!'**.

3.15 Directory Commands

```
cd [directory]
chdir [directory]
shell
```

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

The **shell** command starts an interactive shell.

3.16 Debugging Commands

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

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

The **debug** command lists statistics for the blocks which contain the specified range of lines. If the **!** is present, then the contents of those blocks is displayed, too.

The **validate** command checks certain variables for internal consistency. Normally it doesn't output anything unless it detects a problem. With the **!**, though, it will always produce *some* output.

4. Regular Expressions

ELVIS uses regular expressions for searching and substitutions. A regular expression is a text string in which some characters have special meanings. This is much more powerful than simple text matching.

4.1 Syntax

ELVIS' regexp package treats the following one- or two-character strings (called meta-characters) in special ways:

`\(subexpression\)`

The `\(` and `\)` metacharacters are used to delimit subexpressions. When the regular expression matches a particular chunk of text, ELVIS will remember which portion of that chunk matched the *subexpression*. The `:s/regexp/newtext/` command makes use of this feature.

`^` The `^` metacharacter matches the beginning of a line. If, for example, you wanted to find "foo" at the beginning of a line, you would use a regular expression such as `^foo/`. Note that `^` is only a metacharacter if it occurs at the beginning of a regular expression; anyplace else, it is treated as a normal character.

`$` The `$` metacharacter matches the end of a line. It is only a metacharacter when it occurs at the end of a regular expression; elsewhere, it is treated as a normal character. For example, the regular expression `/$$/` will search for a dollar sign at the end of a line.

`\<` The `\<` metacharacter matches a zero-length string at the beginning of a word. A word is considered to be a string of 1 or more letters and digits. A word can begin at the beginning of a line or after 1 or more non-alphanumeric characters.

`\>` The `\>` metacharacter matches a zero-length string at the end of a word. A word can end at the end of the line or before 1 or more non-alphanumeric characters. For example, `\<end\>/` would find any instance of the word "end", but would ignore any instances of e-n-d inside another word such as "calendar".

`\=` This matches any zero-length string; i.e., it has no effect on the text that a regular expression will match. However, it has a useful side-effect for the forward visual search command: Instead of leaving the cursor at the front of the matching text, it will leave the cursor at the position that matched the `\=` metacharacter. For example, `/zo\=t^M` will locate the next "zot" and leave the cursor on the 't' instead of the 'z'.

`.` The `.` metacharacter matches any single character.

`[character-list]`

This matches any single character from the *character-list*. Inside the *character-list*, you can denote a span of characters by writing only the first and last characters, with a hyphen between them. If the *character-list* is preceded by a `^` character, then the list is inverted — it will match character that *isn't* mentioned in the list. For example, `/[a-zA-Z]/` matches any letter, and `/[^ ]/` matches anything other than a blank.

`\@` If you are in visual mode, and the cursor is on a word in the edit buffer before you start typing the regular expression, then `\@` will match the word at the cursor. For example, `":map #1 \<\@>^M"` will cause the `<F1>` key to search for the next occurrence of the word under the cursor.

`\{n\}` This is a closure operator, which means that it can only be placed after something that matches a single character. It controls the number of times that the single-character expression should be repeated.

The `\{n\}` operator, in particular, means that the preceding expression should be repeated exactly n times. For example, `/^-{\80\}$/` matches a line of eighty hyphens, and `/^[a-zA-Z]{4}>/` matches any four-letter word.

- `\{n,m\}` This is a closure operator which means that the preceding single-character expression should be repeated between n and m times, inclusive. If the m is omitted (but the comma is present) then m is taken to be infinity. For example, `/"[^"]{3,5}"/` matches any pair of quotes which contains three, four, or five non-quote characters.
- `*` The `*` metacharacter is a closure operator which means that the preceding single-character expression can be repeated zero or more times. It is equivalent to `\{0,\}`. For example, `/.*/` matches a whole line.
- `\+` The `\+` metacharacter is a closure operator which means that the preceding single-character expression can be repeated one or more times. It is equivalent to `\{1,\}`. For example, `/./+` matches a whole line, but only if the line contains at least one character. It doesn't match empty lines.
- `\?` The `\?` metacharacter is a closure operator which indicates that the preceding single-character expression is optional — that is, that it can occur 0 or 1 times. It is equivalent to `\{0,1\}`. For example, `/no[-]?one/` matches "no one", "no-one", or "noone".

Anything else is treated as a normal character which must exactly match a character from the scanned text. The special strings may all be preceded by a backslash to force them to be treated normally.

4.2 Substitutions

The `:s` command has at least two arguments: a regular expression, and a substitution string. The text that matched the regular expression is replaced by text which is derived from the substitution string.

Most characters in the substitution string are copied into the text literally but a few have special meaning:

<code>&</code>	Insert a copy of the original text
<code>~</code>	Insert a copy of the previous replacement text
<code>\1</code>	Insert a copy of that portion of the original text which matched the first set of <code>\(\)</code> parentheses
<code>\2-9</code>	Do the same for the second (etc.) pair of <code>\(\)</code>
<code>\U</code>	Convert all chars of any later <code>&</code> or <code>\#</code> to uppercase
<code>\L</code>	Convert all chars of any later <code>&</code> or <code>\#</code> to lowercase
<code>\E</code>	End the effect of <code>\U</code> or <code>\L</code>
<code>\u</code>	Convert the first char of the next <code>&</code> or <code>\#</code> to uppercase
<code>\l</code>	Convert the first char of the next <code>&</code> or <code>\#</code> to lowercase

These may be preceded by a backslash to force them to be treated normally. If "nomagic" mode is in effect, then `&` and `~` will be treated normally, and you must write them as `\&` and `\~` for them to have special meaning.

4.3 Options

ELVIS has two options which affect the way regular expressions are used. These options may be examined or set via the `:set` command.

The first option is called "[no]magic". This is a boolean option, and it is "magic" (TRUE) by default. While in magic mode, all of the meta-characters behave as described above. In nomagic

mode, only ^ and \$ retain their special meaning.

The second option is called "[no]ignorecase". This is a boolean option, and it is "noignorecase" (FALSE) by default. While in ignorecase mode, the searching mechanism will not distinguish between an uppercase letter and its lowercase form. In noignorecase mode, uppercase and lowercase are treated as being different.

Also, the "[no]wrapscan" option affects searches.

4.4 Examples

This example changes every occurrence of "utilize" to "use":

```
:%s/utilize/use/g
```

This example deletes all whitespace that occurs at the end of a line anywhere in the file. (The brackets contain a single space and a single tab.):

```
:%s/[ ]+$//
```

This example converts the current line to uppercase:

```
:s/*^U&/
```

This example underlines each letter in the current line, by changing it into an "underscore backspace letter" sequence. (The ^H is entered as "control-V backspace".):

```
:s/[a-zA-Z]/_ ^H&/g
```

This example locates the last colon in a line, and swaps the text before the colon with the text after the colon. The first \(\) pair is used to delimit the stuff before the colon, and the second pair delimit the stuff after. In the substitution text, \1 and \2 are given in reverse order to perform the swap:

```
:s/(.*):(.*)/^2:1/
```

5. Options

Options may be set or examined via the colon command **set**. The values of options will affect the operation of later commands.

For convenience, options have both a long descriptive name and a short name which is easy to type. You may use either name interchangeably. I like the short names, myself.

There are three types of options: Boolean, string, and numeric. Boolean options are made TRUE by giving the name of the option as an argument to the **set** command; they are made FALSE by prefixing the name with **no**. For example, **set autoindent** makes the autoindent option TRUE, and **set noautoindent** makes it FALSE. ELVIS also allows boolean options to be toggled by prefixing the name with **neg**. So, **:map g :set neglist^M** will cause the <g> key to alternately toggle the **list** option on and off. (The **neg** prefix is an extension; the real *vi* doesn't support it.)

To change the value of a string or numeric option, pass the **set** command the name of the option, followed by an "=" sign and the option's new value. For example, "set tabstop=8" will give the tabstop option a value of 8. For string options, you may enclose the new value in quotes.

NAMES	TYPE	DEFAULT	MEANING
autoindent, ai	Bool	noai	auto-indent during input
autoprint, ap	Bool	ap	in EX, print the current line
autotab, at	Bool	at	auto-indent allowed to use tabs?
autowrite, aw	Bool	noaw	auto-write when switching files
beautify, bf	Bool	nobf	strip control chars from file?
charattr, ca	Bool	noca	interpret \fX sequences?
cc, cc	Str	cc="cc -c"	name of the C compiler
columns, co	Num	co=80	width of the screen
digraph, dig	Bool	nodig	recognize digraphs?
directory, dir	Str	dir="/usr/tmp"	where tmp files are kept
edcompatible, ed	Bool	noed	remember ":s/" options
equalprg, ep	Bool	ep="fmt"	program to run for = operator
errorbells, eb	Bool	eb	ring bell on error
exrc, exrc	Bool	noexrc	read "./.exrc" file?
exrefresh, er	Bool	er	write lines individually in EX
flash, vbell	Bool	flash	use visible alternative to bell
flipcase, fc	Str	fc=""	non-ASCII chars flipped by ~
hideformat, hf	Bool	hf	hide text formatter commands
ignorecase, ic	Bool	noic	upper/lowercase match in search
inputmode, im	Bool	noim	start vi in insert mode?
keepanon, ka	Bool	noka	keep anonymous buffers?
keytime, kt	Num	kt=2	timeout for mapped key entry
keywordprg, kp	Str	kp="ref"	full pathname of shift-K prog
lines, ln	Num	ln=25	number of lines on the screen
list, li	Bool	noli	display lines in "list" mode
magic, ma	Bool	ma	use regular expression in search
make, mk	Str	mk="make"	name of the "make" program
mesg, ms	Bool	ms	allow messages from other users?
modelines, ml	Bool	noml	are modelines processed?
more, more	Bool	more	pause between messages?
nearscroll, ns	Num	ns=15	when to scroll vs. redraw

NAMES	TYPE	DEFAULT	MEANING
newfile, new	BOOL	new	is current file new?
novice, nov	Bool	novice	set options for ease of use
number, nu	Bool	nonumber	show line numbers
paragraphs, para	Str	para="PPppIPLQP"	names of "paragraph" nroff cmd
prompt, pr	Bool	pr	show ':' prompt in ex mode
readonly, ro	Bool	nor	prevent overwriting of orig file
remap, rem	Bool	remap	allow key maps to call key maps
report, re	Num	re=5	report when 5 or more changes
ruler, ru	Bool	noru	display line / column numbers
scroll, sc	Num	sc=12	scroll amount for ^U and ^D
sections, sect	Str	sect="NHSSE"	names of "section" nroff cmd
shell, sh	Str	sh="/bin/sh"	full pathname of the shell
showmatch, sm	Bool	nosm	show matching ()[]{}
showmode, smd	Bool	nosmd	say when we're in input mode
shiftwidth, sw	Num	sw=8	shift amount for < and >
sidescroll, ss	Num	ss=8	amount of sideways scrolling
sync, sy	Bool	nosy	call sync() often
tabstop, ts	Num	ts=8	width of tab characters
taglength, tl	Num	tl=0	significant chars in tag name
tags, tag	Str	tags="tags" list of tags files	
tagstack, tgs	Bool	tgs	enable tagstack?
term, te	Str	te="\$TERM"	name of the termcap entry
terse, tr	Bool	notr	give shorter error messages
timeout, to	Bool	to	distinguish <esc> from <arrow>?
warn, wa	Bool	wa	warn for ! if file modified
window, wi	Num	wi=24	lines to redraw after long move
wrapmargin, wm	Num	wm=0	wrap long lines in input mode
wrapscan, ws	Bool	ws	at EOF, searches wrap to line 1
writeany, wr	Bool	nowr	allow :w to clobber files

autoindent, ai

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.

autoprint, ap

This option only affects EX mode. If the autoprint option on, and either the cursor has moved to a different line or the previous command modified the file, then ELVIS will print the current line.

autotab, at

This option affects the behavior of the autoindent mode. If autoindent is turned off, then autotab has no effect.

When autotab is turned on, elvis will use a mixture of spaces and tabs to create the proper amount of indentation. This is the default.

When autotab is turned off, elvis will only use spaces for auto-indent. ELVIS will still insert a real tab character when you hit the <Tab> key, though; the autotab option only affects *automatic* indentation.

autowrite, aw

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.

beautify, bf

This option causes all control characters to be deleted from the text file, at the time when you start editing it. If you're already editing a file when you turn on the beautify option, then that file won't be affected.

cc

The **:cc** command runs the C compiler. This option should be set to the name of your compiler.

charattr, ca

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 `\fP` 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, co

This option shows how wide your screen is.

digraph, dig

This option is used to enable/disable recognition of digraphs. The default value is nodigraph, which means that digraphs will not be recognized.

directory, dir

ELVIS stores text in temporary files. This option allows you to control which directory those temporary files will appear in. The default is `/usr/tmp`.

This option can only be set in a `.exrc` file; after that, ELVIS will have already started making temporary files in some other directory, so it would be too late.

edcompatible, ed

This option affects the behavior of the `:s/regexp/text/options` command. It is normally off (**:se noed**) which causes all of the substitution options to be off unless explicitly given.

However, with edcompatible on (**:se ed**), the substitution command remembers which options you used last time. Those same options will continue to be used until you change them. In edcompatible mode, when you explicitly give the name of a substitution option, you will toggle the state of that option.

This all seems very strange to me, but its implementation was almost free when I added the **:&** command to repeat the previous substitution, so there it is.

equalprg, ep

This holds the name & arguments of the external filter program used the the `visual =` operator. The default value is "fmt", so the `=` operator will adjust line breaks in text.

errorbells, eb

ELVIS normally rings a bell when you do something wrong. This option lets you disable the bell.

exrc

This option specifies whether a **.exrc** file in the current directory should be executed. By default, this option is off (**:set noexrc**) which prevents elvis from executing **.exrc** in the current directory. If the **.exrc** file in your home directory turns this option on (**:set exrc**) then the ELVIS will attempt to execute the **.exrc** file in the current directory.

This option exist mainly for security reasons. A mean-spirited person could do something like

```
echo >/tmp/.exrc '!rm -rf $HOME'
```

and then anybody who attempted to edit or view a file in the **/tmp** directory would lose most of their files. With the **exrc** option turned off, this couldn't happen to you.

exrefresh, er

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 visual command mode or input mode.

flash, 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 flash option will be off, and you can't turn it on.

flipcase, fc

The flipcase option allows you to control how the non-ASCII characters are altered by the **~** command.

The string is divided into pairs of characters. When **~** is applied to a non-ASCII character, ELVIS looks up the character in the flipcase string to see which pair it's in, and replaces it by the other character of the pair.

hideformat, hf

Many text formatters require you to embed format commands in your text, on lines that start with a "." character. ELVIS normally displays these lines like any other text, but if the hideformat option is on, then format lines are displayed as blank lines.

ignorecase, ic

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.

`inputmode, im`

This option allows you to have ELVIS start up in insert mode. You can still exit insert mode at any time by hitting the ESC key, as usual. Usually, this option would be set in your `.exrc` file.

`keytime, kt`

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.

On most systems, the setting is the number of tenths of a second to allow between characters. On some other systems, the setting is in whole seconds.

Try to avoid setting “`keytime=1`”. Most systems just count clock beats, so if you tried to read a character shortly before a clock beat, you could allow almost no time at all for reading the characters. For higher `keytime` settings, the difference is less critical.

If your system’s response time is poor, you might want to increase the `keytime`. In particular, I’ve found that when keystrokes must be sent through a network (via X windows, `rlogin`, or `telnet`, for example) the `keytime` should be set to at least 1 second.

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`.

`keepanon, ka`

Normally, ELVIS (and the real `vi`) discard the contents of the anonymous cut buffers whenever you switch from one file to another, but retain the contents of the named cut buffers. Setting this option will cause ELVIS to keep all cut buffers when you switch files.

`keywordprg, kp`

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.

The default value of this option is `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, ln`

This option says how many lines your screen has.

`list, li`

In **`nolist`** mode (the default), ELVIS displays text in a “normal” manner — with

tabs expanded to an appropriate number of spaces, etc.

However, sometimes it is useful to have tab characters displayed differently. In list mode, tabs are displayed as "T", and a "\$" is displayed at the end of each line.

magic, ma

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 (**:set noma**), then all characters except ^ and \$ are treated literally. ^ and \$ retain their special meanings regardless of the setting of **magic**.

make, mk

The **:make** command runs your "make" program. This option defines the name of your "make" program.

mesg

With the real *vi*, **:set nomesg** would prevent other users from sending you messages. ELVIS ignores it, though.

modelines, ml

ELVIS supports modelines. Modelines are lines near the beginning or end of your text file which contain **ex:yowza:**, where *yowza* is any EX command. A typical *yowza* would be something like "set ts=5 ca kp=spell wm=15". Other text may also appear on a modeline, so you can place the **ex:yowza:** in a comment:

```
/* ex:set sw=4 ai: */
```

Normally these lines are ignored, for security reasons, but if you have **set modelines** in your **.exrc** file then *yowza* is executed.

nearscroll, ns

The line that contains the cursor will always be on the screen. If you move the cursor to a line that isn't on the screen, then ELVIS will either scroll (if the cursor's line is nearly on the screen already) or redraw the screen completely with the cursor's line centered (if the cursor line is *not* near the screen already).

This option allows you to control elvis' idea of "near". A value of 15 is typical. A value of 1 would cause elvis to scroll no more than one line. A value of 0 disables scrolling.

newfile, new

The **newfile** option is an unsettable boolean option. Its value is automatically set to FALSE when you start editing a file that already exists, or TRUE if the file doesn't exist yet. This can be handy in **.exfilerc** initialization files.

novice, nov

The command **:set novice** is equivalent to **:set nomagic report=1 showmode**.

number, nu

The **number** option causes ELVIS to display line numbers at the start of each line. The numbers are not actually part of the text; when the file is written out, it will be written without line numbers.

paragraphs, pa

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.

more

When ELVIS must display a sequence of messages at the bottom line of the screen in visual mode, it normally pauses after all but the last one, so you have time to read them all.

If you turn off the **more** option, then ELVIS will not pause. This means you can only read the last message, but it is usually the most important one anyway.

prompt, pr

If you **:set noprompt**, then ELVIS will no longer emit a ':' when it expects you to type in an *ex* command. This is slightly useful if you're using an astonishingly slow machine, but the rest of us can just ignore this one.

readonly, ro

Normally, ELVIS will let you write back any file to which you have write permission. If you don't 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 don't 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 can't 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.

remap

The **:map** command allows you to convert one key sequence into another. The **remap** option allows you to specify what should happen if portions of that other sequence are also in the map table. If **remap** is on, then those portions will also be mapped, just as if they had been typed on the keyboard. If **remap** is off, then the matching portions will not be mapped.

For example, if you enter the commands **:map A B** and **:map B C**, then when **remap** is on, A will be converted to C. But when **remap** is off, A will be converted only to B.

report, re

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.

ruler, ru

This option is normally off. If you turn it on, then ELVIS will constantly display the line / column numbers of the cursor, at the bottom of the screen.

scroll, sc

The **^U** and **^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. If you invoke **^U** or **^D** with a count argument (for example, "33^D") then this option's value is set to the count.

sections, se

The **[[** and **]]** commands move the cursor backward or forward in increments 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, sh

When ELVIS forks a shell (perhaps for the **:!** or **:shell** commands) this is the program that is used as a shell. This is **"/bin/sh"** (**\$OPENNT_ROOT/bin/sh**) by default, unless you have set the **SHELL** (or **COMSPEC**, for MS-DOS) environment variable, in which case the default value is copied from the environment.

shiftwidth, sw

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.

showmatch, sm

With **showmatch** set, in input mode every time you hit one of **)]**, ELVIS will momentarily move the cursor to the matching **([**.

showmode, smd

In visual mode, it is easy to forget whether you're in the visual command mode or input/replace mode. Normally, the **showmode** option is off, and you haven't a clue as to which mode you're in. If you turn the **showmode** option on, though, a little message will appear in the lower right-hand corner of your screen, telling you which mode you're in.

sidescroll, ss

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, sy

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 the OS 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 (the default), ELVIS lets the operating system control when data is written to the disk. This is generally faster.

In **sync** mode (which is the default for MS-DOS, AmigaDos, and Atari TOS), ELVIS forces all changes out to disk every time you make a change. This is generally safer, but slower. It can also be a rather rude thing to do on a multi-

user system.

tabstop, ts

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

taglength, tl

This option allows you to specify how many characters of a tag's name must match when performing tag lookup. As a special case, **:set taglength=0** means that all characters of a tag's name must match.

Note: some configurations of ELVIS don't support this option.

tags, tag

If your version of elvis is compiled with `-DINTERNAL_TAGS`, then this is a space-delimited list of tags files. When you tell elvis to look up a tag, it searches through each file in turn until it finds the tag.

If your version of elvis is compiled without `-DINTERNAL_TAGS`, then you can achieve the same effect via an environment variable called *TAGPATH*. *TAGPATH*'s value is a colon-delimited list of file or directory names. (For some operating systems, including MS-DOS, the list is delimited by semicolons instead of colons.)

tagstack

This option allows you to disable the tagstack. I can't think of any reason why you would want to do that.

term, te

This read-only option shows the name of the termcap entry that ELVIS is using for your terminal.

terse, tr

The real *vi* uses this option to select longer vs. shorter error messages. ELVIS has only one set of error messages, though, so this option has no effect.

timeout, to

The command **:set notimeout** is equivalent to **:set keytime=0**, and **:set timeout** is equivalent to **:set keytime=1**. This affects the behavior of the <Esc> key. See the discussion of the **keytime** option for more information.

warn, wa

If you have modified a file but not yet written it back to disk, then ELVIS will normally print a warning before executing a **!cmd** command. However, in **nowarn** mode, this warning is not given.

ELVIS also normally prints a message after a successful search that wrapped at EOF. The **[no]warn** option can also disable this warning.

window, wi

This option controls how many lines are redrawn after a long move.

On fast terminals, this is usually set to the number of rows that the terminal can display, minus one. This causes the entire screen to be filled with text around the cursor.

On slow terminals, you may wish to reduce this value to about 7 or so. That way, if you're doing something like repeatedly hitting 'n' to search for each occurrence of some string and trying to find a particular occurrence, then you don't need to wait as long for ELVIS to redraw the screen after each search.

wrapmargin, wm

Normally (with `wrapmargin=0`) ELVIS will let you type in extremely long lines, if you wish.

However, with **wrapmargin** set to something other than 0 (`wrapmargin=10` is nice), ELVIS will automatically cause long lines to be "wrapped" on a word break for lines come too close to the right-hand margin. For example: On an 80-column screen, **:set wm=10** will cause lines to wrap when their length exceeds 70 columns.

wrapscan, ws

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.

writeln, wr

With **writeln** turned off, elvis will prevent you from accidentally overwriting a file. For example, if "foo" exists then `:w foo` will fail. If you turn on the **writeln** option, then `:w foo` will work.

Regardless of the setting of **writeln**, though, `:w! foo` will work. The `!` forces the **w** command to write the file unless the operating system won't allow it.

6. Initialization

Many features of ELVIS are configurable at runtime. There are commands for assigning actions to keys (:**map**), defining abbreviations (:**abbr**), non-ASCII keying sequences (:**digraph**), setting screen colors (:**color**), and miscellaneous other options (:**set**).

All of these commands can be issued interactively. Experienced *vi* users generally prefer to have some options set every time they run they execute ELVIS, and ELVIS has ways to support this.

6.1 Start-up Initialization

When ELVIS starts, it executes the following algorithm in an attempt to locate initialization commands:

1. If this version of ELVIS supports a system-wide initialization file and that file exists,
Interpret that file's contents as a series of "ex" commands
2. If the *EXINIT* environment variable is set
Interpret the value of *EXINIT* as an "ex" command line.
Else if the home directory contains a file named **.exrc**
Interpret that file's contents as a series of "ex" commands
3. If the **exrc** option is set, and the current directory contains a file named **.exrc**
Interpret that file's contents as a series of "ex" commands
4. If a tag was specified via a **-t** command line argument,
Execute a tag look-up, and load file if successful
5. If no tag was specified, or the specified tag wasn't found,
Load the first file named on the command line, or start empty buffer.
6. If a command was specified via **+command** or **-c command**
Execute the given command.

Note that most of this initialization occurs before the first file is loaded. Consequently, commands which examine or change the edit buffer can't be used there. Only **+command** or **-c command** is executed after the text file has been loaded.

On systems where **.exrc** is an invalid filename, the file is called **ELVIS.RC** instead. Also, the home directory is the directory named by the *HOME* environment variable; on DOS and a few other systems, if *HOME* is unset then ELVIS will use the directory which contains the executable file (**ELVIS.EXE**) as your home directory.

6.2 File Initialization

Loading a file, too, can cause commands to be executed. Each time any file is loaded into the edit buffer, the following algorithm is used to locate file-specific initialization commands.

1. Fill the edit buffer with the file's contents, and set various options and variables accordingly.
2. If the home directory contains a file named **.exfilerc**
Interpret the contents of that file as a series of "ex" commands.
3. If the **modelines** option is set,
Search the first 5 & last 5 lines of the text for lines which contain "ex:<command>:" or "vi:<command>:", and interpret any <command> as an "ex" command line.

On systems where **.exfilerc** is an invalid filename, the file is called **EXFILE.RC** instead.

6.3 The **:mkexrc** Command

ELVIS has a special command, **:mkexrc** [*filename*], to help you create **.exrc** files. It creates a file which sets all nonstandard options, maps, and so on.

By default, the created file's name will be **.exrc** in the current directory. You can either add **:set exrc** to the **.exrc** file in your home directory to force ELVIS to read this new **.exrc** in your current directory, or you can move this new **.exrc** file into your home directory.

Alternatively, you can supply an explicit filename as an argument to **:mkexrc**.

Afterward, you may wish to edit the created file. For example, some options may be conditional; the **:mkexrc** file doesn't distinguish between options which were set unconditionally from those that were set in a text file's modelines or other conditional context.

Warning: the **:mkexrc** command will happily overwrite any file that you tell it to, if your operating system permits.

6.4 Other Techniques

ELVIS has commands for conditional execution, but the standard *vi* doesn't. If you often use the real *vi*, you may want to avoid ELVIS's extensions. To have a terminal-dependent initialization file, you can add **":so \$HOME/.exrc.\$TERM"** to the end of your **.exrc** file, and then create files with names like **.exrc.vt100** and **.exrc.ansi** in your home directory which contain the terminal-dependent commands.

Another good technique is to write a shell-script "wrapper" around ELVIS/*vi*. Here's one of my favorites. It uses to locate files containing a given regular expression, and then starts *vi* on those files with the cursor positioned on the first occurrence in the first file. I call this script "vg".

```
#!/bin/sh
case "$#" in
  0)          echo "usage: vg regexp [files]..." >&2; exit;;
  1)          set -- "$1" *.[ch];;
  esac

regexp="$1"
shift

vi +/"$regexp" 'grep -l "$regexp" "$@"'
```

7. Cut Buffers

When ELVIS deletes text, it stores that text in a cut buffer. This happens in both visual mode and EX mode. There is no practical limit to how much text a cut buffer can hold.

There are 36 cut buffers: 26 named buffers ("a through "z), 9 anonymous buffers ("1 through "9), and 1 extra cut buffer (".).

In EX mode, the **:move** and **:copy** commands use a cut buffer to temporarily hold the text to be moved/copied.

7.1 Putting Text into a Cut Buffer

In visual mode, text is copied into a cut buffer when you use the **d**, **y**, **c**, **C**, **s**, or **x** commands. There are also a few others.

By default, the text goes into the "1 buffer. The text that used to be in "1 gets shifted into "2, "2 gets shifted into "3, and so on. The text that used to be in "9 is lost. This way, the last 9 things you deleted are still accessible.

You can also put the text into a named buffer — "a through "z. To do this, you should type the buffer's name (two keystrokes: a double-quote and a lowercase letter) before the command that will cut the text. When you do this, "1 through "9 are not affected by the cut.

You can append text to one of the named buffers. To do this, type the buffer's name in uppercase (a double-quote and an uppercase letter) before the **d/y/c/** command.

The "." buffer is special. It isn't affected by the **d/y/c/** command. Instead, it stores the text that you typed in the last time you were in input mode. It is used to implement the . visual command, and ^A in input mode.

In EX mode (also known as colon mode), the **:delete**, **:change**, and **:yank** commands all copy text into a cut buffer. Like the visual commands, these EX commands normally use the "1 buffer, but you can use one of the named buffers by giving its name after the command. For example,

```
:20,30y a
```

will copy lines 20 through 30 into cut buffer "a.

You can't directly put text into the "." buffer, or the "2 through "9 buffers.

7.2 Pasting from a Cut Buffer

There are two styles of pasting: line-mode and character-mode. If a cut buffer contains whole lines (from a command like **dd**) then line-mode pasting is used; if it contains partial lines (from a command like **dw**) then character-mode pasting is used. The EX commands always cut whole lines.

Character-mode pasting causes the text to be inserted into the line that the cursor is on.

Line-mode pasting inserts the text on a new line above or below the line that the cursor is on. It doesn't affect the cursor's line at all.

In visual mode, the **p** and **P** commands insert text from a cut buffer. Uppercase **P** will insert it before the cursor, and lowercase **p** will insert it after the cursor. Normally, these commands will paste from the "1 buffer, but you can specify any other buffer to paste from. Just type its name (a double-quote and another character) before you type the **P** or **p**.

In EX mode, the **(pu)t** command pastes text after a given line. To paste from a buffer other than "1, enter its name after the command.

7.3 Macros

The contents of a named cut buffer can be executed as a series of `ex/vi` commands.

To put the instructions into the cut buffer, you must first insert them into the file, and then delete them into a named cut buffer.

To execute a cut buffer's contents as EX commands, you should give the EX command `@` and the name of the buffer. For example, `:@z` will execute "z as a series of EX commands.

To execute a cut buffer's contents as visual commands, you should give the visual command `@` and the letter of the buffer's name. The visual `@` command is different from the EX `@` command. They interpret the cut buffer's contents differently.

The visual `@` command can be rather finicky. Each character in the buffer is interpreted as a keystroke. If you load the instructions into the cut buffer via a `zdd` command, then the newline character at the end of the line will be executed just like any other character, so the cursor would be moved down 1 line. If you don't want the cursor to move down 1 line at the end of each `@z` command, then you should load the cut buffer by saying `0"zD` instead.

Although cut buffers can hold any amount of text, ELVIS can only *execute* small buffers. The size limit is roughly 1000 characters, for either EX macros or VI macros. If a buffer is too large to execute, an error message is displayed.

You can't nest `:@` commands. You can't run `:@` commands from your `.exrc` file, or any other `:source` file either. Similarly, you can't run a `:source` command from within an `@` command. Hopefully, these restrictions will be lifted in a later version.

7.4 The Effect of Switching Files

When ELVIS first starts up, all cut buffers are empty. When you switch to a different file (via the `:n` or `:e` commands perhaps) the 27 named buffers ("a through "z, and ".) retain their text. By default, the 9 anonymous cut buffers ("1 through "9) are emptied, but the `:set keepanon` option allows you to cause them to be retained, too.

8. Differences Between ELVIS & BSD vi/ex

ELVIS is not 100% compatible with the real *vi/ex*. ELVIS has many small extensions, some omissions, and a few features which are implemented in a slightly different manner.

8.1 Extensions

Save Configuration

The **:mkexrc** command saves the current **:set**, **:map**, **:ab**, **:color**, and **:digraph** configurations in the **.exrc** file in your current directory.

Previous File

The **:N** or **:prev** command moves backwards through the args list.

Center Current Row

In visual command mode, the (lowercase) **zz** command will center the current line on the screen, like "z=".

Changing Repeat Count

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.

Previous Text

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).

Keyword Lookup

In visual command mode, 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.

Increment/Decrement

In visual command mode, you can move the cursor onto a number and then hit **##** or **#+** to increment that number by 1. To increment it by a larger amount, type in the increment value before hitting the initial **#**. The number can also be decremented or set by hitting **#-** or **#=**, respectively.

Input Mode

You can backspace past the beginning of the line.

The arrow keys work in input mode.

If you type control-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 control-@ on the real *vi*, except that control-A really works.)

Control-P will insert the contents of the cut buffer.

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

The **~T** and **~D** keys can adjust the indent of a line no matter where the cursor happens to be in that line.

You can save your file and exit ELVIS directly from input mode by hitting control-Z twice.

ELVIS supports digraphs as a way to enter non-ASCII characters.

Start in Input Mode

If you **:set inputmode** in your **.exrc** file, then ELVIS will start up in input mode instead of visual command mode.

Visible Fonts

With **:set charattr**, ELVIS can display "backslash-f" style character attributes on the screen as you edit. The following example shows the recognized attributes:

```
normal \fBboldface\fR \fIitalics\fR \fUunderlined\fR normal
```

NOTE: you must compile ELVIS without the **-DNO_CHARATTR** flag for this to work.

File Syncing

After a crash, you can usually recover the altered form of the file from the temporary file that ELVIS uses — unless the temporary file was corrupted.

On systems that use a delayed-write cache, when ELVIS tries to write to the temporary file, the information might still be in RAM instead of on the disk. A power failure at that time would cause the in-RAM information to be lost. A call such as *sync()* will force all such information to disk.

MS-DOS and Atari TOS don't write a file's length to disk until that file is closed. Consequently, the temporary file would appear to be 0 bytes long if power failed when we were editing. To avoid this problem, a *sync()* function has been written which will close the temporary file and then immediately reopen it.

Cursor Shape

ELVIS changes the shape of the cursor to indicate which mode you're in, if your terminal's termcap entry includes the necessary capabilities.

Hide nroff Lines

The **:set hideformat** option hides nroff format control lines. (They are displayed on the screen as blank lines.)

Compiler Interface

ELVIS is clever enough to parse the error messages emitted by many compilers. To use this feature, you should collect your compiler's error messages into a file called **errlist**; ELVIS will read this file, determine which source file caused the error messages, start editing that file, move the cursor to the line where the error was detected, and display the error message on the status line. Nifty!

Visible Text Selection

In visual command mode, **'v'** starts visibly selecting characters and **'V'** starts visibly selecting whole lines. The character or line where the cursor is located becomes one endpoint of the selection. You can then use the standard cursor movement commands to move the cursor to the other endpoint, and then press one of the operator commands (**c/d/y/</>/!/=/**). The operator will then immediately be applied to the selected text.

Pop-up Menu Operator

The **'\'** key is a new operator, similar in operation to the **c/d/y/</>/!/** operators. It conjures up a menu, from which you can select any of the other operators plus a few other common commands.

Preset Filter Operator

The **'='** key is another new operator. It is similar to the **'!'** operator, except that while **'!'** asks you to type in a filter command each time, **'='** assumes it should always run the command stored in the *equalprg* option.

keepanon

Normally ELVIS discards the anonymous cut buffers when you switch files, just like the real vi. The keepanon option allows anonymous cut buffers to be retained.

Move to a Given Percentage

The '%' movement key can now accept an optional count. Without a count, the '%' key still moves to a matching parenthesis like it always did. With a count somewhere between 1 and 100, though, it moves the cursor to approximately a given percentage of the way through the file. For example, typing "50%" will move the cursor to the middle of the file. In regular expressions, several new forms of closure operators are supported: `\{n\}`, `\{n,m\}`, `\+`, and `\?`. Also, `'\@'` matches the word at the cursor, and `'\='` causes the cursor to be left someplace other than the start of the matching text.

8.2 Omissions

The replace mode is a hack. It doesn't 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. So is the **-r** flag. I've never had a good reason to use **:preserve**, and since **:recover** is used so rarely I 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, I figured.

LISP support is missing. However, the = key is still an operator that reformats lines of text. By default, it reformats lines by sending them through the *fmt* filter, but you could write your own LISP beautifier and configure *elvis* to use it. Key mappings could take care of most other differences. Auto-indent is the only thing that is irrecoverably lost.

Autoindent mode acts a little different from the real vi, anyway. It doesn't handle `^D` or `0D` correctly. On the other hand, it *does* allow `^D` and `^T` to be used anywhere in the line, to adjust the indentation for the whole line.

9. Termcap

ELVIS uses fairly standard termcap fields for most things. I invented the cursor shape names and some of the function key names, but other than that there should be few surprises.

9.1 Required Numeric Fields

:co#:	number of columns on the screen (chars per line)
:li#:	number of lines on the screen

On many systems, ELVIS has other ways to find out how many rows and columns your screen can show, so these values might not be very relevant. If these numbers aren't given in your termcap entry, and ELVIS can't find the screen size any other way, then it will default to 80x24.

9.2 Required String Fields

:ce=:	clear to end-of-line
:cm=:	move the cursor to a given row/column
:up=:	move the cursor up one line

If these fields are missing, then ELVIS will still run fairly well in "ex" mode, but "vi" mode requires these capabilities as an absolute minimum.

9.3 Boolean Fields

:am:	auto margins - wrap when char is written in last column?
:xn:	brain-damaged auto margins - newline ignored after wrap
:pt:	physical tabs?

9.4 Optional String Fields

:al=:	insert a blank row on the screen
:cl=:	home the cursor & clear the screen
:dl=:	delete a row from the screen
:cd=:	clear to end of display
:ei=:	end insert mode
:ic=:	insert a blank character
:im=:	start insert mode
:dc=:	delete a character
:sr=:	scroll reverse (insert row at top of screen)
:vb=:	visible bell
:ks=:	keypad enable
:ke=:	keypad disable
:ti=:	terminal initialization string, to start full-screen mode
:te=:	terminal termination, to end full-screen mode

9.5 Optional Strings Received from the Keyboard

:kd=:	sequence sent by the <down arrow> key
:kl=:	sequence sent by the <left arrow> key
:kr=:	sequence sent by the <right arrow> key
:ku=:	sequence sent by the <up arrow> key
:kP=:	sequence sent by the <PgUp> key
:kN=:	sequence sent by the <PgDn> key
:kh=:	sequence sent by the <Home> key
:kH=:	sequence sent by the <End> key
:kI=:	sequence sent by the <Insert> key

Originally, termcap didn't have any names for the <PgUp>, <PgDn>, <Home>, and <End> keys. Although the capability names shown in the table above are the most common, they are *not*

universal. SCO Xenix uses :PU=:PD=:HM=:EN=: for those keys. Also, if the four arrow keys happen to be part of a 3x3 keypad, then the five non-arrow keys may be named :K1=: through :K5=, so an IBM PC keyboard may be described using those names instead. ELVIS can find any of these names.

9.6 Optional Strings Sent by Function Keys

:k1=:...:k9=:k0=:	codes sent by <F1> through <F10> keys
:s1=:...:s9=:s0=:	codes sent by <Shift F1> ... <Shift F10>
:c1=:...:c9=:c0=:	codes sent by <Ctrl F1> ... <Ctrl F10>
:a1=:...:a9=:a0=:	codes sent by <Alt F1> ... <Alt F10>

Note that :k0=: is used to describe the <F10> key. Some termcap documents recommend :ka=: or even :k=: for describing the <F10> key, but ELVIS doesn't support that.

Also, the :s1=:..., :c1=:..., and :a1=:... codes are very non-standard. The terminfo library doesn't support them. Consequently, if you're using the terminfo library then you might as well add -DNO_SHIFT_FKEY to your CFLAGS setting.

9.7 Optional Fields That Describe Character Attributes

:so=:se=:	start/end standout mode (We don't care about :sg#:=)
:us=:ue=:	start/end underlined mode
:md=:me=:	start/end boldface mode
:as=:ae=:	start/end alternate character set (italics)
:ug#:	visible gap left by :us=:ue=:md=:me=:as=:ae=:

9.8 Optional Fields That Affect the Cursor's Shape

The :cQ=: string is used by ELVIS immediately before exiting to undo the effects of the other cursor shape strings. If :cQ=: is not given, then all other cursor shape strings are ignored.

:cQ=:	normal cursor
:cX=:	cursor used for reading EX command
:cV=:	cursor used for reading VI commands
:cI=:	cursor used during VI input mode
:cR=:	cursor used during VI replace mode

If the capabilities above aren't given, then ELVIS will try to use the following values instead.

:ve=:	normal cursor, used as :cQ=:cX=:cI=:cR=:
:vs=:	gaudy cursor, used as :cV=:

9.9 An Example

Here's the termcap entry I use on my Minix-ST system.

```
mx|minix|minixst|ansi:\
:is=\E[0~:co#80:li#25:bs:pt:\
:cm=\E[%i%d;%dH:up=\E[A:do=^J:nd=\E[C:sr=\EM:\
:cd=\E[J:ce=\E[K:cl=\E[H\E[J:\
:al=\E[L:dl=\E[M:ic=\E[@:dc=\E[P:im=:ei=:\
:so=\E[7m:se=\E[m:us=\E[4m:ue=\E[m:\
:md=\E[1m:me=\E[m:as=\E[1;3m:ae=\E[m:\
:ku=\E[A:kd=\E[B:kr=\E[C:kl=\E[D:\
:k1=\E[1~:k2=\E[2~:k3=\E[3~:k4=\E[4~:k5=\E[5~:\
:k6=\E[6~:k7=\E[17~:k8=\E[18~:k9=\E[19~:k0=\E[20~:
```

10. Environment Variables

ELVIS examines several environment variables when it starts up. The values of these variables are used internally for a variety of purposes. You don't need to define all of these; on most systems, ELVIS only requires *TERM* to be defined. On AmigaDOS, MS-DOS or TOS systems, even that is optional.

TERM, TERMCAP

TERM tells ELVIS the name of the termcap entry to use. *TERMCAP* may contain either the entire termcap entry, or the full pathname of the termcap file to search through.

If your version of ELVIS is using tnytcap instead of the full termcap library, then the value of *TERMCAP* can't be the name of a file; it can only be undefined, or contain the entire termcap entry. In the termcap entry, tnytcap will convert \E to an <Esc> character, but other backslash escapes (\b, \r, etc.) or carat escapes (^[, ^M, etc.) will not be converted to control characters. Instead, you should embed the actual control character into the string.

TMP, TEMP

These only work for AmigaDOS, MS-DOS and Atari TOS. Either of these variables may be used to set the "directory" option, which controls where temporary files are stored. If you define them both, then *TMP* is used, and *TEMP* is ignored.

LINES, COLUMNS

The termcap entry for your terminal should specify the size of your screen. If you're using a windowing interface, then there is an *ioctl()* call which will provide the size of the window; the *ioctl()* values will override the values in the termcap entry. The *LINES* and *COLUMNS* environment variables (if defined) will override either of these sources. They, in turn, can be overridden by a **:set** command.

Normally, the *LINES* and *COLUMNS* variables shouldn't need to be defined.

EXINIT

This variable's value may contain one or more colon-mode commands, which will be executed instead of the **.exrc** file in your home directory. before interactive editing begins.

To put more than one command in *EXINIT*, you can separate the commands with either a newline or a '|' character.

SHELL, COMSPEC

You can use *COMSPEC* in MS-DOS, or *SHELL* in any other system, to specify which shell should be used for executing commands and expanding wildcards.

HOME

This variable should give the full pathname of your home directory. ELVIS needs to know the name of your home directory so it can locate the **.exrc** file there.

TAGPATH

This variable is used by the program. It contains a list of directories that might contain a relevant **tags** file. Under AmigaDOS, MS-DOS or Atari TOS, the names of the directories should be separated by semicolons (";"). Under other operating systems, the names should be separated by colons (":").

If you don't define *TAGPATH*, then will use a default list which includes the current directory and a few other likely places. See the definition of **DEFTAGPATH** at the start of **ref.c** for an accurate list.

11. Questions & Answers

1. How can I make ELVIS run faster under DOS?

There are several things you can do. The first thing to do is get a good screen driver such as NANSI.SYS. This can speed up screen redrawing by as much as a factor of eight! The DOS-specific part of section 12 tells you how to do this.

You might also consider reducing the size of the blocks that ELVIS uses. You'll need to recompile ELVIS to do this. The default BLKSIZE is 2048 bytes for the DOS version of ELVIS, which means that for each keystroke that you insert, ELVIS must shift an average of about 1000 bytes. That's a lot to ask from a little old 5MHz 8088. A BLKSIZE of 512 bytes might be more appropriate.

A "write-back" disk cache can help. DOS 5.0 and Windows 3.0 come with one of these, called SMARTDRV.EXE. I suggest you add "SMARTDRV C+" to your AUTOEXEC.BAT file.

If you're *really* desperate for more speed, you might want to make ELVIS store its temporary files on a RAM disk. However, this limits the size of the file you can edit, and it eliminates any chance you may have had to recover your work after a power failure or system crash, but it might be worth it; you decide. To do this, add `:set dir=R:\fp` (or whatever your RAM disk's name is) to the **elvis.rc** file.

Next, consider turning off the **sync** option. When the sync option is turned on, ELVIS will close the temporary file and reopen it after every change, in order to force DOS to update the file's directory entry. If you put `:set nosync` into the **elvis.rc** file, then ELVIS will only close the file when you start editing a different text file, or when you're exiting ELVIS. Consequently, there is no chance that you'll be able to recover your changes after a power failure... so if you're going to this, then you might as well store the temp files on the RAM disk, too.

2. Why isn't ELVIS reading my .exrc (or ELVIS.RC) file?

For security reasons, ELVIS doesn't normally look for a **.exrc** file in the current directory. A mean-spirited person could lay a trap by placing a dangerous command in a **.exrc** file in some public directory.

If you know for a fact that no such person has ever used your computer, then you can make elvis process the **.exrc** file from the current directory by adding `set exrc` to either your *EXINIT* environment variable or to the **.exrc** file in your home directory.

3. What is my home directory?

On INTERIX systems, each user is given a private "home" directory. Among other uses, this is used for storing personal configuration files for use with various programs. INTERIX stores the pathname of your home directory in an environment variable called *HOME*.

DOS doesn't have home directories, but if you explicitly set *HOME* to the name of some directory, then ELVIS will still look for its **ELVIS.RC** file there. If *HOME* is unset, then ELVIS will look for **ELVIS.RC** in the same directory where **ELVIS.EXE** was found.

4. **Where's the <Esc> key on a DEC keyboard?**

I don't know. Maybe the <F11> key? Maybe Control-3? You could always use **:map!** to make some other key act like the <Esc> key. If all else fails, try <Control-[>.

5. **Is there a way to show which keys do what?**

Yes. The command **:map** will show what each key does in command mode, and **:map!** (with an exclamation mark) shows what each key does in input mode.

The table is divided into three columns: the key's label, the characters that it sends, and the characters that ELVIS pretends you typed.

6. **How can I make ELVIS display long lines like the real vi?**

You can't yet. The next version of ELVIS should support this, though.

7. **I can't recover my text [under MS-DOS or Atari TOS]. According to the directory listing, the temporary file is 0 bytes long. What went wrong?**

MS-DOS and TOS only update a file's directory entry when the file is closed. If the system crashes while the file is still open, then the file's length is stored as 0 bytes. The **:set sync** option is supposed to prevent this; you probably turned it off in the interest of speed, right?

Under MS-DOS [I don't know about TOS], you should delete the empty temporary file, and then run **CHKDSK/F**. This *might* find the data that belonged in the empty file, and place it in a new file with a name like "000001.CHK" — something like that. You can then try to extract the text from that temporary file by giving the command **elvprsv -R 000001.chk**. If you're lucky, then this might recover your text.

8. **What is the most current version of ELVIS?**

Each version of ELVIS that is released to the public has a version number of the form "number point number". As I write this, the most current version of ELVIS is 1.8pl3.

The intermediate steps between one release and the next are labeled with the *next* version number, with a letter appended. For example, after 1.4 was released, I started working on 1.5a. I am currently working on 2.0a. When ELVIS reaches a stable state, I'll call it 2.0 and release it.

Sometimes a beta-test version of ELVIS will be available via anonymous FTP from m2xenix.psg.com, in the directory "pub/elvis/beta".

9. **I only got executables, but now I want the source code. Where can I get it?**

If you have access to the Internet, then you should be able to fetch it from one of the public archives such as **plains.nodak.edu**. It is accessible via anonymous FTP, or via an email server named "archive-server@plains.nodak.edu". Elvis is located in the directory "/pub/Minix/all.contrib".

It is also available from the C Users' Group, in volume #365. As I write this, they are asking \$4 per disk plus \$3.50 per order in the US, and ELVIS requires three disks; this is subject to change. Their phone number is (913) 841-1631, and their address is:

The C Users' Group
1601 W. 23rd Street, #200
Lawrence KS 66046-2743

10. Is this shareware, or public domain, or what?

It is not public domain; it is copyrighted by me, Steve Kirkendall. However, this particular version is freely redistributable, in either source form or executable form. (I would prefer that you give copies away for free, complete with the full source code... but I'm not going to force you.)

It is not shareware; you aren't expected to send me anything. You can use it without guilt.

It is not "copylefted." I hold a copyright, but currently I have not added any of the usual restrictions that you would find on copylefted software. If people start doing really obnoxious things to ELVIS, then I will start adding restrictions to subsequent versions, but earlier versions won't be affected. (So far, everybody has been pretty good about this so no restrictions have been necessary.)

11. Can I reuse parts of your source code?

Yes. Please be careful, though, to make sure that the code really is mine. Some of the code was contributed by other people, and I don't have the authority to give you permission to use it. The author's name can be found near the top of each source file. If it says "Steve Kirkendall" then you may use it; otherwise, you'd better contact the author first.

Please don't remove my name from the source code. If you modify the source, please make a note of that fact in a comment near the top of the source code. And, finally, please mention my name in your documentation.

12. Can ELVIS work with non-ASCII files?

ELVIS is 8-bit clean. This means that ELVIS will allow you to edit files that use a European extended ASCII character set. However, some terminals are not 8-bit clean; they treat characters in the range 0x80-0x9f as control characters. ELVIS expects all characters above 0x7f to be treated as normal displayable characters, so on these terminals ELVIS may produce a scrambled display.

ELVIS can't edit binary files because it can't handle the NUL character, and because of line-length limitations.

ELVIS has also modified to work with 16-bit character sets, but that modification is not part of the standard ELVIS distribution. Yongguang Zhang (ygz@cs.purdue.edu) has created a Chinese version of ELVIS that uses 16-bit characters and runs under cxterm (Chinese X-term) on X-windows systems. Junichiro Itoh (itojun@foretune.co.jp) has modified ELVIS to edit Japanese text under MS-DOS.

NAME

elvis, ex, vi, view, input - The editor

SYNOPSIS

elvis [*flags*] [*+cmd*] [*files...*]

DESCRIPTION

Elvis is a text editor which emulates *vi* / *ex*.

On systems which pass the program name as an argument, such as Minix, you may also install *elvis* under the names *ex*, *vi*, *view*, and *input*. These extra names would normally be links to *elvis*; see the *ln* shell command.

When *elvis* is invoked as *vi*, it behaves exactly as though it was invoked as "*elvis*". However, if you invoke *elvis* as *view*, then the readonly option is set as though you had given it the **-R** flag. If you invoke *elvis* as *ex*, then *elvis* will start up in the colon command mode instead of the visual command mode, as though you had given it the **-e** flag. If you invoke *elvis* as *input* or *edit*, then *elvis* will start up in input mode, as though the **-i** flag was given.

OPTIONS

- r** To the real *vi*, this flag means that a previous edit should be recovered. *Elvis*, though, has a separate program, called *elvrec*, for recovering files. When you invoke *elvis* with **-r**, *elvis* will tell you to run *elvrec*.
- R** This sets the "readonly" option, so you won't accidentally overwrite a file.
- s** This set the "safer" option, which disables many potentially harmful commands. It has not been rigorously proven to be absolutely secure, however.
- t tag** This causes *elvis* to start editing at the given tag.
- m [file]** *Elvis* will search through *file* for something that looks like an error message from a compiler. It will then begin editing the source file that caused the error, with the cursor sitting on the line where the error was detected. If you don't explicitly name a *file*, then "errlist" is assumed.
- e** *Elvis* will start up in colon command mode.
- v** *Elvis* will start up in visual command mode.
- i** *Elvis* will start up in input mode.
- w winsize**
Sets the "window" option's value to *winsize*.
- +command**
If you use the *+command* parameter, *elvis* executes *command* after loading the first file. A typical example would be "*elvis* +237 foo", which would cause *elvis* to start editing foo and then move directly to line 237. *Command* must be a colon command.
- c command**
The SysV version of the *+* option. *elvis* executes *command* after loading the first file. A typical example would be "*elvis* -c 237 foo", which would cause *elvis* to start editing foo and then move directly to line 237. *Command* must be a colon

command.

FILES

Elvis makes use of the following files:

/tmp/elv*

During editing, *elvis* stores text in a temporary file. This file will usually be stored in the **/tmp** directory, and the first three characters will be "elv". For other systems, the temporary files may be stored someplace else; see the version-specific section of the documentation.

tags This is the database used by the **:tags** command and the **-t** option. It is usually created by the *ctags* program.

\$HOME/.exrc

A file called **.exrc** in your home directory is executed as a series of *ex* commands. A file by the same name may be executed in the current directory, too. On systems where ".exrc" is an invalid file name, the initialization file is called "elvis.rc" instead.

ENVIRONMENT

TERM This is the name of your terminal's entry in the termcap or terminfo database. The list of legal values varies from one system to another.

TERMCAP

Optional. If your system uses termcap, and the **TERMCAP** variable is unset, then *elvis* will read your terminal's definition from **/etc/termcap**. If **TERMCAP** is set to the full pathname of a file (starting with a '/') then *elvis* will look in the named file instead of **/etc/termcap**. If **TERMCAP** is set to a value which doesn't start with a '/', then its value is assumed to be the full termcap entry for your terminal.

TERMINFO

Optional. If your system uses terminfo, and the **TERMINFO** variable is unset, then *elvis* will read your terminal's definition from the database in the **/usr/lib/terminfo** database. If **TERMINFO** is set, then its value is used as the database name to use instead of **/usr/lib/terminfo**.

LINES, COLUMNS

Optional. These variables, if set, will override the screen size values given in the termcap / terminfo for your terminal. On windowing systems such as X, *elvis* has other ways of determining the screen size, so you should probably leave these variables unset.

EXINIT Optional. This variable can hold EX commands which will be executed instead of the .exrc file in your home directory.

SHELL Optional. The **SHELL** variable sets the default value for the "shell" option, which determines which shell program is used to perform wildcard expansion in file names, and also which is used to execute filters or external programs. The default value is **\$INTERIX_ROOT/bin/sh**.

Note: Under MS-DOS, this variable is called **COMSPEC** instead of **SHELL**.

HOME This variable should be set to the name of your home directory. *elvis* looks for its initialization file there; if **HOME** is unset then the initialization file will not be executed.

TAGPATH

Optional. This variable is used by the "ref" program, which is invoked by the shift-K, control-], and :tag commands. See "ref" for more information.

TMP, TEMP These optional environment variables are only used in some of *elvis*. They allow you to supply a directory name to be used for storing temporary files.

SEE ALSO

ctags, ref, elvprsv, elvrec

Elvis - A Clone of Vi/Ex, the complete *elvis* documentation.

BUGS

There is no LISP support. Certain other features are missing, too.

Auto-indent mode is not quite compatible with the real *vi*. Among other things, O^D and ^~D don't do what you might expect.

Long lines are displayed differently. The real *vi* wraps long lines onto multiple rows of the screen, but *elvis* scrolls sideways.

AUTHOR

Steve Kirkendall
kirkenda@cs.pdx.edu

Many other people have worked to port *elvis* to various operating systems. To see who deserves credit, run the **:version** command from within *elvis*, or look in the system-specific section of the complete documentation.

NAME

elvprsv – Preserve the the modified version of a file after a crash.

SYNOPSIS

```
elvprsv ["-why elvis died"] /tmp/filename...  
elvprsv -R /tmp/filename...
```

DESCRIPTION

elvprsv preserves your edited text after *elvis* dies. The text can be recovered later, via the *elvprsv* program.

For systems with an autostartup, you should never need to run this program from the command line. It is run automatically when *elvis* is about to die, and it should be run (via **/etc/rc**) when the computer is booted. **THAT'S ALL!**

For systems such as MS-DOS or VMS, you can either use *elvprsv* in the same way (by running it from your AUTOEXEC.BAT file), or you can run it separately with the **-R** flag to recover the files in one step.

If you're editing a file when *elvis* dies (due to a bug, system crash, power failure, etc.) then *elvprsv* will preserve the most recent version of your text. The preserved text is stored in a special directory; it does *not* overwrite your text file automatically. (If the preservation directory hasn't been set up correctly, then *elvprsv* will simply send you a mail message descrining how to manually run *elvprsv*.)

elvprsv will send mail to any user whose work it preserves, if your operating system normally supports mail.

FILES

/tmp/elv*

The temporary file that *elvis* was using when it died.

/usr/preserve/p*

The text that is preserved by *elvprsv*.

/usr/preserve/Index

A text file which lists the names of all preserved files, and the names of the **/usr/preserve/p*** files which contain their preserved text.

BUGS

Due to the permissions on the **/usr/preserve** directory, *elvprsv* must be run as superuser on systems which have a superuser. This is accomplished by making the *elvprsv* executable be owned by "root" and turning on its "set user id" bit.

If you're editing a nameless buffer when *elvis* dies, then *elvprsv* will pretend that the file was named "foo".

AUTHOR

Steve Kirkendall
kirkenda@cs.pdx.edu

NAME

elvrec - Recover the modified version of a file after a crash

SYNOPSIS

elvrec [*preservedfile* [*newfile*]]

DESCRIPTION

If you're editing a file when *elvis* dies, the system crashes, or power fails, the most recent version of your text will be preserved. The preserved text is stored in a special directory; it does NOT overwrite your text file automatically.

The *elvrec* program (called *virec* on INTERIX) locates the preserved version of a given file, and writes it over the top of your text file -- or to a new file, if you prefer. The recovered file will have nearly all of your changes.

To see a list of all recoverable files, run *elvrec* with no arguments.

(Note: if you haven't set up a directory for file preservation, then *elvis*' you'll have to manually run the *elvprsv* program instead of *elvrec*.)

FILES

/usr/preserve/p* The text that was preserved when *elvis* died.

/usr/preserve/Index A text file which lists the names of all preserved files, and the names of the **/usr/preserve/p*** files which contain their preserved text.

BUGS

elvrec is very picky about filenames. You must tell it to recover the file using exactly the same pathname as when you were editing it. The simplest way to do this is to go into the same directory that you were editing, and invoke *elvrec* with the same filename as *elvis*. If that doesn't work, then try running *elvrec* with no arguments, to see exactly which pathname it is using for the desired file.

Due to the permissions on the **/usr/preserve** directory, *elvrec* must be run as superuser on systems which have a superuser. This is accomplished by making the *elvrec* executable be owned by "root" and setting its "set user id" bit.

If you're editing a nameless buffer when *elvis* dies, then *elvrec* will pretend that the file was named "foo".

AUTHOR

Steve Kirkendall
kirkenda@cs.pdx.edu