The vi, ex, and Vim Editors: Appendix A - Learning the vi and Vim Editors, Seventh Edition

by Linda Lamb, Arnold Robbins, Elbert Hannah

This excerpt is from **Learning the vi and Vim Editors, Seventh Edition**.

The standard guide for vi since 1986, this book has been expanded to include detailed information on vim, the leading vi clone that includes extra features for both beginners and power users. You learn text editing basics and advanced tools for both editors, such as writing macros and scripts to extend the editor, power tools for programmers, multi-window editing -- all in the easy-to-follow style that has made this book a classic.

This appendix summarizes the standard features of **vi** in quick-reference format. Commands entered at the colon (known as **ex** commands because they date back to the original creation of that editor) are included, as well as the most popular Vim features.

This appendix presents the following topics:

- Command-line syntax
- Review of **vi** operations
- Alphabetical list of keys in command mode
- **vi** commands
- **vi** configuration
- **ex** basics
- Alphabetical summary of **ex** commands

Command-Line Syntax

The three most common ways of starting a **vi** session are:

```
vi [options] file
vi [options] +num file
vi [options] +/pattern file
```

You can open *file* for editing, optionally at line *num* or at the first line matching *pattern*. If no *file* is specified, **vi** opens with an empty buffer.

Command-Line Options

Because **vi** and **ex** are the same program, they share the same options. However, some options only make sense for one version of the program. Options specific to Vim are so marked:

+[*num*]

Start editing at line number *num*, or the last line of the file if *num* is omitted.

+/ *pattern*

Start editing at the first line matching *pattern*. (For **ex**, this fails if `nowrapscan` is set in your `.exrc` startup file, since **ex** starts editing at the last line of a file.)

+? *pattern*

Start editing at the last line matching *pattern*.

-b

Edit the file in binary mode. {Vim}

-c *command*

Run the given **ex** command upon startup. Only one `-c` option is permitted for **vi**; Vim accepts up to 10. An older form of this option, `+command`, is still supported.

--cmd *command*

Like `-c`, but execute the command before any resource files are read. {Vim}

-C

Solaris **vi**: same as `-x`, but assume the file is encrypted already.

Vim: start the editor in **vi**-compatible mode.

-d

Run in diff mode. Works like **vimdiff**. {Vim}

-D

Debugging mode for use with scripts. {Vim}

-e

Run as **ex** (line-editing rather than full-screen mode).

-h

Print help message, then exit. {Vim}

-i *file*

Use the specified *file* instead of the default (`~/.viminfo`) to save or restore Vim's state. {Vim}

-l

Enter Lisp mode for running Lisp programs (not supported in all versions).

-L

List files that were saved due to an aborted editor session or system crash (not supported in all versions). For Vim, this option is the same as `-r`.

-m

Start the editor with the `write` option turned off so that the user cannot write to files. {Vim}

-M

Do not allow text in files to be modified. {Vim}

-n

Do not use a swap file; record changes in memory only. {Vim}

--noplugin

Do not load any plug-ins. {Vim}

-N

Run Vim in a non-**vi**-compatible mode. {Vim}

-o[*num*]

Start Vim with *num* open windows. The default is to open one window for each file. {Vim}

-O[*num*]

Start Vim with *num* open windows arranged horizontally (split vertically) on the screen. {Vim}

-r [*file*]

Recovery mode; recover and resume editing on *file* after an aborted editor session or system crash. Without *file*, list files available for recovery.

-R

Edit files in read-only mode.

-s

Silent; do not display prompts. Useful when running a script. This behavior also can be set through the older `-` option. For Vim, applies only when used together with `-e`.

-s *scriptfile*

Read and execute commands given in the specified *scriptfile* as if they were typed in from the keyboard. {Vim}

-S *commandfile*

Read and execute commands given in *commandfile* after loading any files for editing specified on the command line. Shorthand for `vim -c 'source *commandfile*'`. {Vim}

-t *tag*

Edit the file containing *tag*, and position the cursor at its definition.

-T *type*

Set the option terminal type. This value overrides the $TERM environment variable. {Vim}

-u *file*

Read configuration information from the specified resource file instead of the default `.vimrc` resource file. If the *file* argument is `NONE`, Vim will read no resource files, load no plug-ins, and run in compatible mode. If the argument is `NORC`, it will read no resource files, but it will load plug-ins. {Vim}

-v

Run in full-screen mode (default for **vi**).

--version

Print version information, then exit. {Vim}

-V[*num*]

Verbose mode; print messages about what options are being set and what files are being read or written. You can set a level of verbosity to increase or decrease the number of messages received. The default value is 10 for high verbosity. {Vim}

-w *rows*

Set the window size so *rows* lines at a time are displayed; useful when editing over a slow dial-up line (or long distance Internet connection). Older versions of **vi** do not permit a space between the option and its argument. Vim does not support this option.

-W *scriptfile*

Write all typed commands from the current session to the specified *scriptfile*. The file created can be used with the `-s` command. {Vim}

-x

Prompt for a key that will be used to try to encrypt or decrypt a file using **crypt** (not supported in all versions).[74]

-y

Modeless **vi**; run Vim in insert mode only, without a command mode. This is the same as invoking Vim as **evim**. {Vim}

-Z

Start Vim in restricted mode. Do not allow shell commands or suspension of the editor. {Vim}

Although most people know **ex** commands only by their use within **vi**, the editor also exists as a separate program and can be invoked from the shell (for instance, to edit files as part of a script). Within **ex**, you can enter the `vi` or `visual` command to start **vi**. Similarly, within **vi**, you can enter `Q` to quit the **vi** editor and enter **ex**.

You can exit **ex** in several ways:

`:x`	Exit (save changes and quit).
`:q!`	Quit without saving changes.
`:vi`	Enter the `vi` editor.

Review of vi Operations

This section provides a review of the following:

- **vi** modes
- Syntax of **vi** commands
- Status-line commands

Command Mode

Once the file is opened, you are in command mode. From command mode, you can:

- Invoke insert mode
- Issue editing commands
- Move the cursor to a different position in the file
- Invoke **ex** commands
- Invoke a Unix shell
- Save the current version of the file
- Exit **vi**

Insert Mode

In insert mode, you can enter new text in the file. You normally enter insert mode with the `i` command. Press the **ESC** key to exit insert mode and return to command mode. The full list of commands that enter insert mode is provided later in the section the section called “Insert Commands”.

Syntax of vi Commands

In **vi**, editing commands have the following general form:

`[`*`n`*`] `*`operator`*` [`*`m`*`]`*`motion`*

The basic editing *operators* are:

`c`	Begin a change.
`d`	Begin a deletion.
`y`	Begin a yank (or copy).

If the current line is the object of the operation, the *motion* is the same as the operator: `cc`, `dd`, `yy`. Otherwise, the editing operators act on objects specified by cursor-movement commands or pattern-matching commands. (For example, `cf.` changes up to the next period.) *n* and *m* are the number of times the operation is performed, or the number of objects the operation is performed on. If both *n* and *m* are specified, the effect is *n* × *m*.

An object of operation can be any of the following text blocks:

word

Includes characters up to a whitespace character (space or tab) or punctuation mark. A capitalized object is a variant form that recognizes only whitespace.

sentence

Up to `.`, `!`, or `?`, followed by two spaces.

paragraph

Up to the next blank line or paragraph macro defined by the `para=` option.

section

Up to the next **nroff/troff** section heading defined by the `sect=` option.

motion

Up to the character or other text object as specified by a motion specifier, including pattern searches.

Examples

`2cw`	Change the next two words.
`d}`	Delete up to the next paragraph.
`d^`	Delete back to the beginning of the line.

`5yy`	Copy the next five lines.
`y]]`	Copy up to the next section.
`cG`	Change to the end of the edit buffer.

More commands and examples may be found in the section the section called “Changing and deleting text” later in this appendix.

Visual mode (Vim only)

Vim provides an additional facility, “visual mode.” This allows you to highlight blocks of text, which then become the object of edit commands such as deletion or saving (yanking). Graphical versions of Vim allow you to use the mouse to highlight text in a similar fashion. See the earlier section the section called “Visual Mode Motion” for more information.

`v`	Select text in visual mode one character at a time.
`V`	Select text in visual mode one line at a time.
CTRL-V	Select text in visual mode in blocks.

Status-Line Commands

Most commands are not echoed on the screen as you input them. However, the status line at the bottom of the screen is used to edit these commands:

`/`	Search forward for a pattern.
`?`	Search backward for a pattern.
`:`	Invoke an **ex** command.
`!`	Invoke a Unix command that takes as its input an object in the buffer and replaces it with output from the command. You type a motion command after the `!` to describe what should be passed to the Unix command. The command itself is entered on the status line.

Commands that are entered on the status line must be entered by pressing the **ENTER** key. In addition, error messages and output from the **CTRL-G** command are displayed on the status line.

vi Commands

vi supplies a large set of single-key commands when in command mode. Vim supplies additional multikey commands.

Movement Commands

Some versions of **vi** do not recognize extended keyboard keys (e.g., arrow keys, page up, page down, home, insert, and delete); some do. All versions, however, recognize the keys in this section. Many users of **vi** prefer to use these keys, as it helps them keep their fingers on the home row of the keyboard. A number preceding a command repeats the movement. Movement commands are also used after an operator. The operator works on the text that is moved.

Character

`h`, `j`, `k`, `l`	Left, down, up, right (←, ↓, ↑, →)
Spacebar	Right
BACKSPACE	Left
CTRL-H	Left

Text

`w`, `b`	Forward, backward by “word” (letters, numbers, and underscores make up words).
`W`, `B`	Forward, backward by “WORD” (only whitespace separates items).
`e`	End of word.
`E`	End of WORD.
`ge`	End of previous word. {Vim}

`gE`	End of previous WORD. {Vim}
`)`, `(`	Beginning of next, current sentence.
`}`, `{`	Beginning of next, current paragraph.
`]]`, `[[`	Beginning of next, current section.
`][`, `[ ]`	End of next, current section. {Vim}

Lines

Long lines in a file may show up on the screen as multiple lines. (They *wrap* around from one screen line to the next.) Although most commands work on the lines as defined in the file, a few commands work on lines as they appear on the screen. The Vim option `wrap` allows you to control how long lines are displayed.

`0`, `$`	First, last position of current line.
`^`, `_`	First nonblank character of current line.
`+`, `-`	First nonblank character of next, previous line.
ENTER	First nonblank character of next line.
num `|`	Column *num* of current line.
`g0`, `g$`	First, last position of screen line. {Vim}
`g^`	First nonblank character of screen line. {Vim}
`gm`	Middle of screen line. {Vim}
`gk`, `gj`	Move up, down one screen line. {Vim}
`H`	Top line of screen (Home position).
`M`	Middle line of screen.
`L`	Last line of screen.
num `H`	*num* lines after top line.
num `L`	*num* lines before last line.

Screens

CTRL-F, **CTRL-B**	Scroll forward, backward one screen.
CTRL-D, **CTRL-U**	Scroll down, up one-half screen.
CTRL-E, **CTRL-Y**	Show one more line at bottom, top of screen.
`z` **ENTER**	Reposition line with cursor to top of screen.
`z.`	Reposition line with cursor to middle of screen.
`z-`	Reposition line with cursor to bottom of screen.
CTRL-L	Redraw screen (without scrolling).
CTRL-R	**vi**: redraw screen (without scrolling).
	Vim: redo last undone change.

Searches

`/ pattern`	Search forward for *pattern*. End with **ENTER**.
`/ pattern` `/+ num`	Go to line *num* after *pattern*. Forward search for *pattern*.
`/ pattern` `/- num`	Go to line *num* before *pattern*. Forward search for *pattern*.
`? pattern`	Search backward for *pattern*. End with **ENTER**.
`? pattern` `?+ num`	Go to line *num* after *pattern*. Backward search for *pattern*.
`? pattern` `?- num`	Go to line *num* before *pattern*. Backward search for *pattern*.
`:noh`	Suspend search highlighting until next search. {Vim}
`n`	Repeat previous search.
`N`	Repeat search in opposite direction.
`/`	Repeat previous search forward.
`?`	Repeat previous search backward.
`*`	Search forward for word under cursor. Matches only exact words. {Vim}
`#`	Search backward for word under cursor. Matches only exact words. {Vim}
`g*`	Search backward for word under cursor. Matches the characters of this word when embedded in a longer word. {Vim}
`g#`	Search backward for word under cursor. Matches the characters of this word when embedded in a longer word. {Vim}
`%`	Find match of current parenthesis, brace, or bracket.
`f x`	Move cursor forward to *x* on current line.
`F x`	Move cursor backward to *x* on current line.
`t x`	Move cursor forward to character before *x* in current line.
`T x`	Move cursor backward to character after *x* in current line.
`,`	Reverse search direction of last `f`, `F`, `t`, or `T`.
`;`	Repeat last `f`, `F`, `t`, or `T`.

Line numbering

CTRL-G	Display current line number.
`gg`	Move to first line in file. {Vim}
num `G`	Move to line number *num*.
`G`	Move to last line in file.
`:` *num*	Move to line number *num*.

Marks

`m x`	Place mark *x* at current position.
`` ` x ``	(Backquote.) Move cursor to mark *x*.
`' x`	(Apostrophe.) Move to start of line containing *x*.
`` `` ```	(Backquotes.) Return to position before most recent jump.
`''`	(Apostrophes.) Like preceding, but return to start of line.
`'"`	(Apostrophe quote.) Move to position when last editing the file. {Vim}
`` `[, `] ``	(Backquote bracket.) Move to beginning/end of previous text operation. {Vim}
`'[, ']`	(Apostrophe bracket.) Like preceding, but return to start of line where operation occurred. {Vim}
`` `. ``	(Backquote period.) Move to last change in file. {Vim}
`'.`	(Apostrophe period.) Like preceding, but return to start of line. {Vim}
`'0`	(Apostrophe zero.) Position where you last exited Vim. {Vim}
`:marks`	List active marks. {Vim}

Insert Commands

`a`	Append after cursor.
`A`	Append to end of line.
`c`	Begin change operation.
`C`	Change to end of line.
`gI`	Insert at beginning of line. {Vim}
`i`	Insert before cursor.
`I`	Insert at beginning of line.
`o`	Open a line below cursor.
`O`	Open a line above cursor.
`R`	Begin overwriting text.
`s`	Substitute a character.
`S`	Substitute entire line.
`ESC`	Terminate insert mode.

The following commands work in insert mode:

BACKSPACE	Delete previous character.
DELETE	Delete current character.
TAB	Insert a tab.
CTRL-A	Repeat last insertion. {Vim}

CTRL-D	Shift line left to previous shiftwidth. {Vim}
CTRL-E	Insert character found just below cursor. {Vim}
CTRL-H	Delete previous character (same as backspace).
CTRL-I	Insert a tab.
CTRL-K	Begin insertion of multikeystroke character.
CTRL-N	Insert next completion of the pattern to the left of the cursor. {Vim}
CTRL-P	Insert previous completion of the pattern to the left of the cursor. {Vim}
CTRL-T	Shift line right to next shiftwidth. {Vim}
CTRL-U	Delete current line.
CTRL-V	Insert next character verbatim.
CTRL-W	Delete previous word.
CTRL-Y	Insert character found just above cursor. {Vim}
CTRL-[	(ESC) Terminate insert mode.

Some of the control characters listed in the previous table are set by **stty**. Your terminal settings may differ.

Edit Commands

Recall that `c`, `d`, and `y` are the basic editing operators.

Changing and deleting text

The following list is not exhaustive, but it illustrates the most common operations:

`cw`	Change word.
`cc`	Change line.
`c$`	Change text from current position to end-of-line.
`C`	Same as `c$`.
`dd`	Delete current line.
num`dd`	Delete *num* lines.
`d$`	Delete text from current position to end-of-line.
`D`	Same as `d$`.
`dw`	Delete a word.
`d}`	Delete up to next paragraph.
`d^`	Delete back to beginning of line.
`d/` *pat*	Delete up to first occurrence of pattern.
`dn`	Delete up to next occurrence of pattern.
`df` *x*	Delete up to and including *x* on current line.

`dt` *x*	Delete up to (but not including) *x* on current line.
`dL`	Delete up to last line on screen.
`dG`	Delete to end of file.
`gqap`	Reformat current paragraph to `textwidth`. {Vim}
`g~w`	Switch case of word. {Vim}
`guw`	Change word to lowercase. {Vim}
`gUw`	Change word to uppercase. {Vim}
`p`	Insert last deleted or yanked text after cursor.
`gp`	Same as `p`, but leave cursor at end of inserted text. {Vim}
`gP`	Same as `P`, but leave cursor at end of inserted text. {Vim}
`]p`	Same as `p`, but match current indention. {Vim}
`[p`	Same as `P`, but match current indention. {Vim}
`P`	Insert last deleted or yanked text before cursor.
`r` *x*	Replace character with *x*.
`R` *text*	Replace with new *text* (overwrite), beginning at cursor. **ESC** ends replace mode.
`s`	Substitute character.
`4s`	Substitute four characters.
`S`	Substitute entire line.
`u`	Undo last change.
CTRL-R	Redo last change. {Vim}
`U`	Restore current line.
`x`	Delete current cursor position.
`X`	Delete back one character.
`5X`	Delete previous five characters.
`.`	Repeat last change.
`~`	Reverse case and move cursor right.
CTRL-A	Increment number under cursor. {Vim}
CTRL-X	Decrement number under cursor. {Vim}

Copying and moving

Register names are the letters `a`–`z`. Uppercase names append text to the corresponding register.

`Y`	Copy current line.
`yy`	Copy current line.

`" x yy`	Copy current line to register *x*.
`ye`	Copy text to end of word.
`yw`	Like `ye`, but include the whitespace after the word.
`y$`	Copy rest of line.
`" x dd`	Delete current line into register *x*.
`" x d`	Delete into register *x*.
`" x p`	Put contents of register *x*.
`y]]`	Copy up to next section heading.
`J`	Join current line to next line.
`gJ`	Same as `J`, but without inserting a space. {Vim}
`:j`	Same as `J`.
`:j!`	Same as `gJ`.

Saving and Exiting

Writing a file means overwriting the file with the current text.

`ZZ`	Quit **vi**, writing the file only if changes were made.
`:x`	Same as `ZZ`.
`:wq`	Write file and quit.
`:w`	Write file.
`:w` *`file`*	Save copy to *file*.
`:` *`n`* `,` *`m`* `w` *`file`*	Write lines *n* to *m* to new *file*.
`:` *`n`* `,` *`m`* `w >>` *`file`*	Append lines *n* to *m* to existing *file*.
`:w!`	Write file (overriding protection).
`:w!` *`file`*	Overwrite *file* with current text.
`:w %.` *`new`*	Write current buffer named *file* as *`file`*`.new`.
`:q`	Quit **vi** (fails if changes were made).
`:q!`	Quit **vi** (discarding edits).
`Q`	Quit **vi** and invoke **ex**.
`:vi`	Return to **vi** after `Q` command.
`%`	Replaced with current filename in editing commands.
`#`	Replaced with alternate filename in editing commands.

Accessing Multiple Files

`:e` *`file`*	Edit another *file*; current file becomes alternate.
`:e!`	Return to version of current file at time of last write.
`:e +` *`file`*	Begin editing at end of *file*.
`:e +` *`num file`*	Open *file* at line *num*.
`:e #`	Open to previous position in alternate file.
`:ta` *`tag`*	Edit file at location *tag*.
`:n`	Edit next file in the list of files.
`:n!`	Force next file.
`:n` *`files`*	Specify new list of *files*.
`:rewind`	Edit first file in the list.
CTRL-G	Show current file and line number.
`:args`	Display list of files to be edited.
`:prev`	Edit previous file in the list of files.

Window Commands (Vim)

The following table lists common commands for controlling windows in Vim. See also the `split`, `vsplit`, and `resize` commands in the later section the section called “Alphabetical Summary of ex Commands”. For brevity, control characters are marked in the following list by `^`.

`:new`	Open a new window.
`:new` *`file`*	Open *file* in a new window.
`:sp` [*`file`*]	Split the current window. With *file*, edit that file in the new window.
`:sv` [*`file`*]	Same as `:sp`, but make new window read-only.
`:sn` [*`file`*]	Edit next file in file list in new window.
`:vsp` [*`file`*]	Like `:sp`, but split vertically instead of horizontally.
`:clo`	Close current window.
`:hid`	Hide current window, unless it is the only visible window.
`:on`	Make current window the only visible one.
`:res` *`num`*	Resize window to *num* lines.
`:wa`	Write all changed buffers to their files.
`:qa`	Close all buffers and exit.
`^W s`	Same as `:sp`.
`^W n`	Same as `:new`.
`^W ^`	Open new window with alternate (previously edited) file.
`^W c`	Same as `:clo`.

`^W o`	Same as `:only`.
`^W j`, `^W k`	Move cursor to next/previous window.
`^W p`	Move cursor to previous window.
`^W h`, `^W l`	Move cursor to window on left/right of screen.
`^W t`, `^W b`	Move cursor to window on top/bottom of screen.
`^W K`, `^W B`	Move current window to top/bottom of screen.
`^W H`, `^W L`	Move current window to far left/right of screen.
`^W r`, `^W R`	Rotate windows down/up.
`^W +`, `^W -`	Increase/decrease current window size.
`^W =`	Make all windows same height.

Interacting with the System

`:r` *`file`*	Read in contents of *file* after cursor.
`:r !` *`command`*	Read in output from *command* after current line.
`:` *`num`* `r !` *`command`*	Like previous, but place after line *num* (0 for top of file).
`:!` *`command`*	Run *command*, then return.
`!` *`motion command`*	Send the text covered by *motion* to Unix *command*; replace with output.
`:` *`n`* `,` *`m`* `!` *`command`*	Send lines *n*–*m* to *command*; replace with output.
`num` `!!` *`command`*	Send *num* lines to Unix *command*; replace with output.
`:!!`	Repeat last system command.
`:sh`	Create subshell; return to editor with *EOF*.
CTRL-Z	Suspend editor, resume with **fg**.
`:so` *`file`*	Read and execute **ex** commands from *file*.

Macros

`:ab` *`in out`*	Use *in* as abbreviation for *out* in insert mode.
`:unab` *`in`*	Remove abbreviation for *in*.
`:ab`	List abbreviations.
`:map` *`string sequence`*	Map characters *string* as *sequence* of commands. Use `#1`, `#2`, etc., for the function keys.
`:unmap` *`string`*	Remove map for characters *string*.
`:map`	List character strings that are mapped.

`:map! string sequence`	Map characters *string* to input mode *sequence*.
`:unmap! string`	Remove input mode map (you may need to quote the characters with **CTRL-V**).
`:map!`	List character strings that are mapped for input mode.
`q x`	Record typed characters into register specified by letter *x*. If letter is uppercase, append to register. {Vim}
`q`	Stop recording. {Vim}
`@ x`	Execute the register specified by letter *x*. Use `@@` to repeat the last `@` command.

In **vi**, the following characters are unused in command mode and can be mapped as user-defined commands:

Letters

`g`, `K`, `q`, `V`, and `v`

Control keys

`^A`, `^K`, `^O`, `^W`, `^X`, `^_`, and `^\`

Symbols

`_`, `*`, `\`, `=`, and `#`

Tip

The `=` is used by **vi** if Lisp mode is set. Different versions of **vi** may use some of these characters, so test them before using.

Vim does not use `^K`, `^_`, `_`, or `\`.

Miscellaneous Commands

`<`	Shift text described by following motion command left by one shiftwidth. {Vim}
`>`	Shift text described by following motion command right by one shiftwidth. {Vim}
`<<`	Shift line left one shiftwidth (default is eight spaces).
`>>`	Shift line right one shiftwidth (default is eight spaces).
`>}`	Shift right to end of paragraph.
`<%`	Shift left until matching parenthesis, brace, or bracket. (Cursor must be on the matching symbol.)
`==`	Indent line in C-style, or using program specified in `equalprg` option. {Vim}
`g`	Start many multiple character commands in Vim.
`K`	Look up word under cursor in manpages (or program defined in `keywordprg`). {Vim}
`^O`	Return to previous jump. {Vim}
`^Q`	Same as `^V`. {Vim} (On some terminals, resume data flow.)
`^T`	Return to the previous location in the tag stack. (Solaris **vi**, Vim, **nvi**, **elvis**, and **vile**.)
`^]`	Perform a tag lookup on the text under the cursor.
`^\`	Enter **ex** line-editing mode.
`^^`	(Caret key with Ctrl key pressed.) Return to previously edited file.

vi Configuration

This section describes the following:

- The `:set` command
- Options available with `:set`
- Example `.exrc` file

The :set Command

The `:set` command allows you to specify options that change characteristics of your editing environment. Options may be put in the `~/.exrc` file or set during a **vi** session.

The colon does not need to be typed if the command is put in `.exrc`:

`:set` *x*	Enable Boolean option *x*; show value of other options.
`:set no` *x*	Disable option *x*.
`:set` *x* `=` *value*	Give *value* to option *x*.
`:set`	Show changed options.
`:set all`	Show all options.
`:set` *x* `?`	Show value of option *x*.

Appendix B, *Setting Options* provides tables of `:set` options for Solaris **vi**, Vim, **nvi**, **elvis**, and **vile**. Please see that appendix for more information.

Example .exrc File

In an `ex` script file, comments start with the double quote character. The following lines of code are an example of a customized `.exrc` file:

```
set nowrapscan                  " Searches don't wrap at end of file
set wrapmargin=7                " Wrap text at 7 columns from right margin
set sections=SeAhBhChDh nomesg  " Set troff macros, disallow message
map q :w^M:n^M                  " Alias to move to next file
map v dwElp                     " Move a word
ab ORA O'Reilly Media, Inc.     " Input shortcut
```

Tip

The `q` alias isn't needed for Vim, which has the `:wn` command. The `v` alias would hide the Vim command `v`, which enters character-at-a-time visual mode operation.

ex Basics

The **ex** line editor serves as the foundation for the screen editor **vi**. Commands in **ex** work on the current line or on a range of lines in a file. Most often, you use **ex** from within **vi**. In **vi**, **ex** commands are preceded by a colon and entered by pressing **ENTER**.

You can also invoke **ex** on its own—from the command line—just as you would invoke **vi**. (You could execute an **ex** script this way.) Or you can use the **vi** command `Q` to quit the **vi** editor and enter **ex**.

Syntax of ex Commands

To enter an **ex** command from **vi**, type:

`:[`*address*`] command [`*options*`]`

An initial `:` indicates an **ex** command. As you type the command, it is echoed on the status line. Execute the command by pressing the **ENTER** key. *Address* is the line number or range of lines that are the object of *command*. *Options* and *addresses* are described later. **ex** commands are described in the later section the section called "Alphabetical Summary of ex Commands".

You can exit **ex** in several ways:

`:x`	Exit (save changes and quit).
`:q!`	Quit without saving changes.
`:vi`	Switch to the **vi** editor on the current file.

Addresses

If no address is given, the current line is the object of the command. If the address specifies a range of lines, the format is:

x`,`*y*

where *x* and *y* are the first and last addressed lines (*x* must precede *y* in the buffer). *x* and *y* each may be a line number or a symbol. Using `;` instead of `,` sets the current line to *x* before interpreting *y*. The notation `1,$` addresses all lines in the file, as does `%`.

Address Symbols

`1,$`	All lines in the file.
x `,` *y*	Lines *x* through *y*.
x `;` *y*	Lines *x* through *y*, with current line reset to *x*.

`0`	Top of file.
`.`	Current line.
num	Absolute line number *num*.
`$`	Last line.
`%`	All lines; same as `1,$`.
x `-` *n*	*n* lines before *x*.
x `+` *n*	*n* lines after *x*.
`-`[*num*]	One or *num* lines previous.
`+`[*num*]	One or *num* lines ahead.
`'` *x*	(Apostrophe.) Line marked with *x*.
`''`	(Apostrophe apostrophe.) Previous mark.
`/` *pattern* `/`	Forward to line matching *pattern*.
`?` *pattern* `?`	Backward to line matching *pattern*.

See Chapter 6, *Global Replacement* for more information on using patterns.

Options

`!`

Indicates a variant form of the command, overriding the normal behavior. The `!` must come immediately after the command.

count

The number of times the command is to be repeated. Unlike in **vi** commands, *count* cannot precede the command, because a number preceding an **ex** command is treated as a line address. For example, `d3` deletes three lines, beginning with the current line; `3d` deletes line 3.

file

The name of a file that is affected by the command. `%` stands for the current file; `#` stands for the previous file.

Alphabetical Summary of ex Commands

ex commands can be entered by specifying any unique abbreviation. In the following list of reference entries, the full name appears as the heading of the reference entry, and the shortest possible abbreviation is shown in the syntax line below it. Examples are assumed to be typed from **vi**, so they include the `:` prompt.

Name

abbreviate

Synopsis

```
ab [string text]
```

Define *string* when typed to be translated into *text*. If *string* and *text* are not specified, list all current abbreviations.

Examples

Note: `^M` appears when you type `^V` followed by **ENTER**.

```
:ab ora O'Reilly Media, Inc.
:ab id Name:^MRank:^MPhone:
```

Name

append

Synopsis

```
[address] a[!]
text
.
```

Append new *text* at specified *address*, or at present address if none is specified. Add a `!` to toggle the `autoindent` setting that is used during input. That is, if `autoindent` was enabled, `!` disables it. Enter new text after entering the command. Terminate input of new text by entering a line consisting of just a period.

Example

```
:aBegin appending to current line
Append this line
and this line too.
.Terminate input of text to append
```

Name

args

Synopsis

```
ar
args file ...
```

Print the members of the argument list (files named on the command line), with the current argument printed in brackets (`[ ]`).

The second syntax is for Vim, which allows you to reset the list of files to be edited.

Name

bdelete

Synopsis

```
[num] bd[!] [num]
```

Unload buffer *num* and remove it from the buffer list. Add a `!` to force removal of an unsaved buffer. The buffer may also be specified by filename. If no buffer is specified, remove the current buffer. {Vim}

Name

buffer

Synopsis

```
[num] b[!] [num]
```

Begin editing buffer *num* in the buffer list. Add a `!` to force a switch from an unsaved buffer. The buffer may also be specified by filename. If no buffer is specified, continue editing the current buffer. {Vim}

Name

buffers

Synopsis

```
buffers[!]
```

Print the members of the buffer list. Some buffers (e.g., deleted buffers) will not be listed. Add `!` to show unlisted buffers. `ls` is another abbreviation for this command. {Vim}

Name

cd

Synopsis

```
cd dir
chdir dir
```

Change the current directory within the editor to *dir*.

Name

center

Synopsis

```
[address] ce [width]
```

Center the line within the specified *width*. If *width* is not specified, use `textwidth`. {Vim}

Name

change

Synopsis

```
[address] c[!]
text
.
```

Replace the specified lines with *text*. Add a `!` to switch the `autoindent` setting during input of *text*. Terminate input by entering a line consisting of just a period.

Name

close

Synopsis

```
clo[!]
```

Close current window unless it is the last window. If buffer in window is not open in another window, unload it from memory. This command will not close a buffer with unsaved changes, but you may add `!` to hide it instead. {Vim}

Name

copy

Synopsis

```
[address] co destination
```

Copy the lines included in *address* to the specified *destination* address. The command `t` (short for "to") is a synonym for `copy`.

Example

```
:1,10 co 50Copy first 10 lines to just after line 50
```

Name

delete

Synopsis

```
[address] d [register] [count]
```

Delete the lines included in *address*. If *register* is specified, save or append the text to the named register. Register names are the lowercase letters `a`–`z`. Uppercase names append text to the corresponding register. If *count* is specified, delete that many lines.

Examples

```
:/Part I/,/Part II/-1dDelete to line above "Part II"
:/main/+d               Delete line below "main"
:.,$d xDelete from this line to last line into register x
```

Name

edit

Synopsis

```
e[!] [+num] [filename]
```

Begin editing on *filename*. If no *filename* is given, start over with a copy of the current file. Add a `!` to edit the new file even if the current file has not been saved since the last change. With the *+num* argument, begin editing on line *num*. Alternatively, *num* may be a pattern, of the form */pattern*.

Examples

```
:e fileEdit file in current editing buffer
:e +/^Index #       Edit alternate file at pattern match
:e!Start over again on current file
```

Name

file

Synopsis

```
f [filename]
```

Change the filename for the current buffer to *filename*. The next time the buffer is written, it will be written to file *filename*. When the name is changed, the buffer's "not edited" flag is set, to indicate that you are not editing an existing file. If the new filename is the same as a file that already exists on the disk, you will need to use `:w!` to overwrite the existing file. When specifying a filename, the `%` character can be used to indicate the current filename. A `#` can be used to indicate the alternate filename. If no *filename* is specified, print the current name and status of the buffer.

Example

```
:f %.new
```

Name

fold

Synopsis

```
address fo
```

Fold the lines specified by *address*. A fold collapses several lines on the screen into one line, which can later be unfolded. It doesn't affect the text of the file. {Vim}

Name

foldclose

Synopsis

```
[address] foldc[!]
```

Close folds in the specified *address*, or at the present address if none is specified. Add a `!` to close more than one level of folds. {Vim}

Name

foldopen

Synopsis

```
[address] foldo[!]
```

Open folds in the specified *address*, or at the present address if none is specified. Add a `!` to open more than one level of folds. {Vim}

Name

global

Synopsis

```
[address] g[!]/pattern/[commands]
```

Execute *commands* on all lines that contain *pattern* or, if *address* is specified, on all lines within that range. If *commands* are not specified, print all such lines. Add a `!` to execute *commands* on all lines *not* containing *pattern*. See also `v`, later in this list.

Examples

```
:g/Unix/pPrint all lines containing “Unix”
:g/Name:/s/tom/Tom/Change “tom” to “Tom” on all lines containing “Name:”
```

Name

hide

Synopsis

```
hid
```

Close current window unless it is the last window, but do not remove the buffer from memory. This command is safe to use on an unsaved buffer. {Vim}

Name

insert

Synopsis

```
[address] i[!]
text
.
```

Insert *text* at line before the specified *address*, or at present address if none is specified. Add a `!` to switch the `autoindent` setting during input of *text*. Terminate input of new text by entering a line consisting of just a period.

Name

join

Synopsis

```
[address] j[!] [count]
```

Place the text in the specified range on one line, with whitespace adjusted to provide two space characters after a period (.), no space characters before a `)`, and one space character otherwise. Add a `!` to prevent whitespace adjustment.

Example

```
:1,5j!Join first five lines, preserving whitespace
```

Name

jumps

Synopsis

```
ju
```

Print jump list used with **CTRL-I** and **CTRL-O** commands. The jump list is a record of most movement commands that skip over multiple lines. It records the position of the cursor before each jump. {Vim}

Name

k

Synopsis

```
[address] k char
```

Same as `mark`; see `mark` later in this list.

Name

left

Synopsis

`[address] le [count]`

Left-align lines specified by *address*, or current line if no address is specified. Indent lines by *count* spaces. {Vim}

Name

list

Synopsis

`[address] l [count]`

Print the specified lines so that tabs display as `^I` and the ends of lines display as `$`. `l` is like a temporary version of `:set list`.

Name

map

Synopsis

`map[!] [string commands]`

Define a keyboard macro named *string* as the specified sequence of *commands*. *string* is usually a single character or the sequence **#***num*, the latter representing a function key on the keyboard. Use a `!` to create a macro for input mode. With no arguments, list the currently defined macros.

Examples

```
:map K dwwPTranspose two words
:map q :w^M:n^M        Write current file; go to next
:map! + ^[bi(^[ea)Enclose previous word in parentheses
```

Tip

Vim has `K` and `q` commands, which the example aliases would hide.

Name

mark

Synopsis

`[address] ma char`

Mark the specified line with *char*, a single lowercase letter. Same as `k`. Return later to the line with `'x` (apostrophe plus *x*, where *x* is the same as *char*). Vim also uses uppercase and numeric characters for marks. Lowercase letters work the same as in **vi**. Uppercase letters are associated with filenames and can be used between multiple files. Numbered marks, however, are maintained in a special `viminfo` file and cannot be set using this command.

Name

marks

Synopsis

`marks [chars]`

Print list of marks specified by *chars*, or all current marks if no *chars* specified. {Vim}

Example

```
:marks abcPrint marks a, b, and c
```

Name

mkexrc

Synopsis

`mk[!] file`

Create an `.exrc` file containing `set` commands for changed **ex** options and key mappings. This saves the current option settings, allowing you to restore them later. {Vim}

Name

move

Synopsis

`[address] m destination`

Move the lines specified by *address* to the *destination* address.

Example

```
:.,/Note/m /END/Move text block to after line containing “END”
```

Name

new

Synopsis

```
[count] new
```

Create a new window *count* lines high with an empty buffer. {Vim}

Name

next

Synopsis

```
n[!] [[+num] filelist]
```

Edit the next file from the command-line argument list. Use `args` to list these files. If *filelist* is provided, replace the current argument list with *filelist* and begin editing on the first file. With the *+num* argument, begin editing on line *num*. Alternatively, *num* may be a pattern, of the form */pattern*.

Example

```
:n chap*Start editing all “chapter” files
```

Name

nohlsearch

Synopsis

```
noh
```

Temporarily stop highlighting all matches to a search when using the `hlsearch` option. Highlighting is resumed with the next search. {Vim}

Name

number

Synopsis

```
[address] nu [count]
```

Print each line specified by *address*, preceded by its buffer line number. Use `#` as an alternate abbreviation for `number`. *count* specifies the number of lines to show, starting with *address*.

Name

only

Synopsis

```
on [!]
```

Make the current window be the only one on the screen. Windows open on modified buffers are not removed from the screen (hidden), unless you also use the `!` character. {Vim}

Name

open

Synopsis

```
[address] o [/pattern/]
```

Enter open mode (**vi**) at the lines specified by *address*, or at the lines matching *pattern*. Exit open mode with `Q`. Open mode lets you use the regular **vi** commands, but only one line at a time. It can be useful on slow dial-up lines (or on very distant Internet **ssh** connections).

Name

preserve

Synopsis

```
pre
```

Save the current editor buffer as though the system were about to crash.

Name

previous

Synopsis

`prev[!]`

Edit the previous file from the command-line argument list. {Vim}

Name

print

Synopsis

`[address] p [count]`

Print the lines specified by *address*. *count* specifies the number of lines to print, starting with *address*. `P` is another abbreviation.

Example

`:100;+5p` *Show line 100 and the next 5 lines*

Name

put

Synopsis

`[address] pu [char]`

Place previously deleted or yanked lines from the named register specified by *char*, to the line specified by *address*. If *char* is not specified, the last deleted or yanked text is restored.

Name

qall

Synopsis

`qa[!]`

Close all windows and terminate the current editing session. Use `!` to discard changes made since the last save. {Vim}

Name

quit

Synopsis

`q[!]`

Terminate the current editing session. Use `!` to discard changes made since the last save. If the editing session includes additional files in the argument list that were never accessed, quit by typing `q!` or by typing `q` twice. Vim closes the editing window only if there are still other windows open on the screen.

Name

read

Synopsis

`[address] r filename`

Copy the text of *filename* after the line specified by *address*. If *filename* is not specified, the current filename is used.

Example

`:0r $HOME/data` *Read file in at top of current file*

Name

read

Synopsis

`[address] r !command`

Read the output of shell *command* into the text after the line specified by *address*.

Example

`:$r !spell %` *Place results of spellchecking at end of file*

Name

recover

Synopsis

`rec [file]`

Recover *file* from the system save area.

Name

redo

Synopsis

`red`

Restore last undone change. Same as **CTRL-R**. {Vim}

Name

resize

Synopsis

`res [[±]num]`

Resize current window to be *num* lines high. If `+` or `-` is specified, increase or decrease the current window height by *num* lines. {Vim}

Name

rewind

Synopsis

`rew[!]`

Rewind the argument list and begin editing the first file in the list. Add a `!` to rewind even if the current file has not been saved since the last change.

Name

right

Synopsis

`[address] ri [width]`

Right-align lines specified by *address*, or current line if no address is specified, to column *width*. Use `textwidth` option if no *width* is specified. {Vim}

Name

sbnext

Synopsis

`[count] sbn [count]`

Split the current window and begin editing the *count* next buffer from the buffer list. If no count is specified, edit the next buffer in the buffer list. {Vim}

Name

sbuffer

Synopsis

`[num] sb [num]`

Split the current window and begin editing buffer *num* from the buffer list in the new window. The buffer to be edited may also be specified by filename. If no buffer is specified, open the current buffer in the new window. {Vim}

Name

set

Synopsis

`se parameter1 parameter2 ...`

Set a value to an option with each *parameter*, or if no *parameter* is supplied, print all options that have been changed from their defaults. For Boolean options, each *parameter* can be phrased as *option* or `no`*option*; other options can be assigned with the syntax *option*=*value*. Specify `all` to list current settings. The form `set` *option*`?` displays the value of *option*. See the tables that list `set` options in Appendix B, *Setting Options*.

Examples

```
:set nows wm=10
:set all
```

Name

shell

Synopsis

`sh`

Create a new shell. Resume editing when the shell terminates.

Name

snext

Synopsis

`[count] sn [[+num] filelist]`

Split the current window and begin editing the next file from the command-line argument list. If *count* is provided, edit the *count* next file. If *filelist* is provided, replace the current argument list with *filelist* and begin editing the first file. With the *+n* argument, begin editing on line *num*. Alternately, *num* may be a pattern of the form */pattern*. {Vim}

Name

source

Synopsis

`so file`

Read (source) and execute **ex** commands from *file*.

Example

```
:so $HOME/.exrc
```

Name

split

Synopsis

`[count] sp [+num] [filename]`

Split the current window and load *filename* in the new window, or the same buffer in both windows if no file is specified. Make the new window *count* lines high, or if *count* is not specified, split the window into equal parts. With the *+n* argument, begin editing on line *num*. *num* may also be a pattern of the form */pattern*. {Vim}

Name

sprevious

Synopsis

`[count] spr [+num]`

Split the current window and begin editing the previous file from the command-line argument list in the new window. If *count* is specified, edit the *count* previous file. With the *+num* argument, begin editing on line *num*. *num* may also be a pattern of the form */pattern*. {Vim}

Name

stop

Synopsis

`st`

Suspend the editing session. Same as **CTRL-Z**. Use the shell **fg** command to resume the session.

Name

substitute

Synopsis

`[address] s [/pattern/replacement/] [options] [count]`

Replace the first instance of *pattern* on each of the specified lines with *replacement*. If *pattern* and *replacement* are omitted, repeat last substitution. *count* specifies the number of lines on which to substitute, starting with *address*. (Spelling out the command name does not work in Solaris **vi**.)

Options

c	Prompt for confirmation before each change.
g	Substitute all instances of *pattern* on each line (global).
p	Print the last line on which a substitution was made.

Examples

```
:1,10s/yes/no/gSubstitute on first 10 lines
:%s/[Hh]ello/Hi/gc           Confirm global substitutions
:s/Fortran/\U&/ 3            Uppercase “Fortran” on next three lines
:g/^[0-9][0-9]*/s//Line &:/For every line beginning with one or more digits, a
```

Name

suspend

Synopsis

`su`

Suspend the editing session. Same as **CTRL-Z**. Use the shell **fg** command to resume the session.

Name

sview

Synopsis

`[`*`count`*`] sv [+`*`num`*`] [`*`filename`*`]`

Same as the `split` command, but set the `readonly` option for the new buffer. {Vim}

Name

t

Synopsis

`[`*`address`*`] t `*`destination`*

Copy the lines included in *address* **t**o the specified *destination* address. `t` is equivalent to `copy`.

Example

`:%t$`*`Copy the file and add it to the end`*

Name

tag

Synopsis

`[`*`address`*`] ta `*`tag`*

In the `tags` file, locate the file and line matching *tag* and start editing there.

Example

Run **ctags**, then switch to the file containing *myfunction*:

```
:!ctags *.c
:tagmyfunction
```

Name

tags

Synopsis

`tags`

Print list of tags in the tag stack. {Vim}

Name

unabbreviate

Synopsis

`una `*`word`*

Remove *word* from the list of abbreviations.

Name

undo

Synopsis

`u`

Reverse the changes made by the last editing command. In **vi** the undo command will undo itself, redoing what you undid. Vim supports multiple levels of undo. Use `redo` to redo an undone change in Vim.

Name

unhide

Synopsis

`[count] unh`

Split screen to show one window for each active buffer in the buffer list. If specified, limit the number of windows to *count*. {Vim}

Name

unmap

Synopsis

`unm[!] string`

Remove *string* from the list of keyboard macros. Use `!` to remove a macro for input mode.

Name

v

Synopsis

`[address] v/pattern/[command]`

Execute *command* on all lines *not* containing *pattern*. If *command* is not specified, print all such lines. `v` is equivalent to `g!`. See `global`, earlier in this list.

Example

```
:v/#include/dDelete all lines except “#include” lines
```

Name

version

Synopsis

`ve`

Print the editor’s current version number and date of last change.

Name

view

Synopsis

`vie[[+num] filename]`

Same as `edit`, but set file to `readonly`. When executed in **ex** mode, return to normal or visual mode. {Vim}

Name

visual

Synopsis

`[address] vi [type] [count]`

Enter visual mode (**vi**) at the line specified by *address*. Return to **ex** mode with `Q`. *type* can be one of `-`, `^`, or `.` (see the `z` command, later in this section). *count* specifies an initial window size.

Name

visual

Synopsis

`vi [+num] file`

Begin editing *file* in visual mode (**vi**), optionally at line *num*. Alternately, *num* may be a pattern, of the form `/`*pattern*. {Vim}

Name

vsplit

Synopsis

`[count] vs [+num] [filename]`

Same as the `split` command, but split the screen vertically. The *count* argument can be used to specify a width for the new window. {Vim}

Name

wall

Synopsis

`wa[!]`

Write all changed buffers with filenames. Add `!` to force writing of any buffers marked `readonly`. {Vim}

Name

wnext

Synopsis

`[count] wn[!] [[+num] filename]`

Write current buffer and open next file in argument list, or the *count* next file if specified. If *filename* is specified, edit it next. With the *+num* argument, begin editing on line *num*. *num* may also be a pattern of the form */pattern*. {Vim}

Name

wq

Synopsis

`wq[!]`

Write and quit the file in one action. The file is always written. The `!` flag forces the editor to write over any current contents of *file*.

Name

wqall

Synopsis

`wqa[!]`

Write all changed buffers and quit the editor. Add `!` to force writing of any buffers marked `readonly`. `xall` is another alias for this command. {Vim}

Name

write

Synopsis

`[address] w[!] [[>>] file]`

Write lines specified by *address* to *file*, or write full contents of buffer if *address* is not specified. If *file* is also omitted, save the contents of the buffer to the current filename. If *>> file* is used, append lines to the end of the specified *file*. Add a `!` to force the editor to write over any current contents of *file*.

Examples

```
:1,10w name_listCopy first 10 lines to file name_list
:50w >> name_listNow append line 50
```

Name

write

Synopsis

`[address] w !command`

Write lines specified by *address* to *command*.

Example

```
:1,66w !pr -h myfile | lpPrint first page of file
```

Name

X

Synopsis

`X`

Prompt for an encryption key. This can be preferable to `:set key`, as typing the key is not echoed to the console. To remove an encryption key, just reset the `key` option to an empty value. {Vim}

Name

xit

Synopsis

`x`

Write the file if it was changed since the last write, and then quit.

Name

yank

Synopsis

```
[address] y [char] [count]
```

Place lines specified by *address* in named register *char*. Register names are the lowercase letters a–z. Uppercase names append text to the corresponding register. If no *char* is given, place lines in the general register. *count* specifies the number of lines to yank, starting with *address*.

Example

```
:101,200 ya aCopy lines 100–200 to register “a”
```

Name

z

Synopsis

```
[address] z [type] [count]
```

Print a window of text with the line specified by *address* at the top. *count* specifies the number of lines to be displayed.

Type

+

Place specified line at the top of the window (default).

-

Place specified line at the bottom of the window.

.

Place specified line in the center of the window.

^

Print the previous window.

=

Place specified line in the center of the window and leave the current line at this line.

Name

&

Synopsis

```
[address] & [options] [count]
```

Repeat the previous substitute (s) command. *count* specifies the number of lines on which to substitute, starting with *address*. *options* are the same as for the substitute command.

Examples

```
:s/Overdue/Paid/Substitute once on current line
:g/Status/&Redo substitution on all “Status” lines
```

Name

@

Synopsis

```
[address] @ [char]
```

Execute contents of register specified by *char*. If *address* is given, move cursor to the specified address first. If *char* is @, repeat the last @ command.

Name

=

Synopsis

```
[address] =
```

Print the line number of the line indicated by *address*. The default is the line number of the last line.

Name

!

Synopsis

```
[address] !command
```

Execute Unix *command* in a shell. If *address* is specified, use the lines contained in *address* as standard input to *command*, and replace those lines with the output and error output. (This is called *filtering* the text through the *command*.)

Examples

```
:!lsList files in the current directory
:11,20!sort -fSort lines 11–20 of current file
```

Name

< >

Synopsis

```
[address] < [count]
  or
[address] > [count]
```

Shift lines specified by *address* either left (`<`) or right (`>`). Only leading spaces and tabs are added or removed when shifting lines. *count* specifies the number of lines to shift, starting with *address*. The `shiftwidth` option controls the number of columns that are shifted. Repeating the `<` or `>` increases the shift amount. For example, `:>>>` shifts three times as much as `:>`.

Name

~

Synopsis

```
[address] ~ [count]
```

Replace the last-used regular expression (even if from a search, and not from an `s` command) with the replacement pattern from the most recent `s` (substitute) command. This is rather obscure; see Chapter 6, *Global Replacement* for details.

Name

address

Synopsis

```
        address
```

Print the lines specified in *address*.

Name

ENTER

Synopsis

Print the next line in the file. (For **ex** only, not from the `:` prompt in **vi**.)

[74] The **crypt** command's encryption is weak. Don't use it for serious secrets.