vi Editor Pocket Reference

Arnold Robbins, 1st Edition 1999, Copyright © 1999 by O'Reilly, ISBN 1-56592-497-5, Order Number: 4975, 72 pages, US $6.95

For many users, working in the Unix environment means using *vi*, a full-screen text editor available on most Unix systems. Even those who know *vi* often make use of only a small number of its features.

The *vi Editor Pocket Reference* is a companion volume to O'Reilly's newly updated *Learning the vi Editor*, now in its 6th edition, a complete guide to text editing with *vi*. New topics in *Leanring the vi Editor* include coverage of four *vi* clones, *vim*, *elvis*, *nvi* and *vile*, and their extensions, such as multiwindow editing, extended regular expressions, and GUI interfaces.

This small book is a handy reference guide to the information in the larger volume, presenting movement and editing commands, command-line options, and other elements of the vi editor in an easy-to-use tabular format.

Arnold Robbins, an Atlanta native now happily living in Israel, is a professional programmer and technical author, and author of the second edition of O'Reilly's *sed & awk* as well as the new *Learning the vi Editor*. He has been working with Unix systems since 1980. He currently maintains *gawk* (*GNU awk*) and its documentation.

Table of Contents

Introduction

This pocket reference is a compilation to *Learning the via Editor*, by Linda Lamb and Arnold Robbins. It describes the *vi* command-line options, command mode commands, *ex* commands and options, regular expressions and the use of the substitute (`s`) command, and other pertinent information for using *vi*. Also covered are the additional features in the four *vi* clones, *nvi*, *elvis*, *vim* and *vile*.

The Solaris 2.6 version of *vi* served as the "reference" version of *vi* for this pocket reference.

Conventions

The following font conventions are used in this book:

`Courier`

Used for command names, options, and everything to be typed literally

`Courier Italic`

Used for replaceable text within commands

Italic

Used for replaceable text within regualar text, program names, filenames, paths, for emphasis, and new terms when first defined

[...]

Identifies optional text; the brackets are not typed

`CTRL-G`

Indicates a keystroke

1. Command Line Options

Command	Action
`vi file`	Invoke *vi* on *file*
`vi file1 file2`	Invoke *vi* on files sequentially
`view file`	Invoke *vi* on *file* in read-only mode
`vi -R file`	Invoke *vi* on *file* in read-only mode
`vi -r file`	Recover *file* and recent edits after a crash
`vi -t tag`	Look up a *tag* and start editing at its definition
`vi -w n`	Set the windows size to *n*; useful over a slow connection
`vi + file`	Open *file* at last line
`vi +n file`	Open *file* directly at line number *n*
`vi -c command file`	Open *file*, execute *command*, which is usually a search command or line number (POSIX)
`vi +/pattern file`	Open *file* directly at *pattern*
`ex file`	Invoke *ex* on *file*
`ex - file <script`	Invoke *ex* on *file*, taking commands from *script*; surpress informative messages and prompts
`ex -s file <script`	Invoke *ex* on *file*, taking commands from *script*; surpress informative messages and prompts (POSIX)

2. vi Commands

Most vi command follow a general pattern:

```
[command][number]text object
```

or the equivalent form:

```
[number][command]text object
```

Movement Commands

Command	Meaning
Character	
`h, j, k, l`	Left, down, up, right

Text

`w, W, b, B`	Forward, backward by word
`e, E`	End of word
`), (`	Beginning of next, previous sentence
`}, {`	Beginning of next, previous paragraph
`]], [[`	Beginning of next, previous section

Lines

`RETURN`	First nonblank character of next line
`0, $`	First, last position of current line
`^`	First nonblank character of current line
`+, -`	First nonblank character of next, previous line
`n|`	Column *n* of current line
`H`	Top line of screen
`M`	Middle line of screen
`L`	Last line of screen
`nH`	*n* (number) of lines after top line
`nL`	*n* (number) of lines before last line

Scrolling

`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 window
`z`**`RETURN`**	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)

Searches

`/pattern`	Search forward for *pattern*
`?pattern`	Search backward for *pattern*
`n, N`	Repeat last search in the same, opposite direction
`/, ?`	Repeat previous search forward, backward
`fx`	Search forward for character *x* in current line
`Fx`	Search backward for character *x* in current line
`tx`	Search forward to character before *x* in current line
`Tx`	Search backward to character after *x* in current line
`;`	Repeat previous current line search

`'`	Repeat previous current line search in opposite direction

Line Number

`CTRL-G`	Display current line number
`nG`	Move to line number *n*
`G`	Move to last line in file
`:n`	Move to line *n* in file

Marking position

`mx`	Mark current position as *x*
`` ``x ```	Move cursor to mark *x*
`` `` ```	Return to previous mark or context
`'x`	Move to the beginning of line containing mark *x*
`''`	Return to beginning of line containing previous mark

Editing Commands

Command	**Action**

Insert

`i, a`	Insert text before, after cursor
`I, A`	Insert text before beginning, after end of line
`o, O`	Open new line for text below, above cursor

Change

`r`	Replace character
`cw`	Change word
`cc`	Change current line
`cmotion`	Change text between the cursor and the target of *motion*
`C`	Change to end of line
`R`	Type over (overwrite) characters
`s`	Substitute: delete character and insert new text
`S`	Substitute: delete current line and insert new text

Delete, move

`x`	Delete character under cursor
`X`	Delete character before cursor
`dw`	Delete word

`dd`	Delete current line
`dmotion`	Delete text between the cursor and the target of *motion*
`D`	Delete to end of line
`p, P`	Put deleted text after, before cursor
`"np`	Put text from delete buffer number *n* after cursor (for last nine deletions)

Yank

`yw`	Yank (copy) word
`yy`	Yank cureent line
`"ayy`	Yank current line into named buffer *a* (a-z). Uppercase names append text
`ymotion`	Yank text between the cursor and the target of *motion*
`p, P`	Put yanked text after, before cursor
`"aPv`	Put text from buffer *a* before cursor (a-z)

Other Commands

`.`	Repeat last edit command
`u, U`	Undo last edit, restore current line
`J`	Join two lines

ex edit commands

`:d`	Delete lines
`:m`	Move lines
`:co` or `:t`	Copy lines
`:.,$d`	Delete from current line to end of file
`:30,60m0`	Move lines 30 through 60 to top of file
`:.,/pattern/co$`	Copy from current line through line containing *pattern* to end of file

Exit Commands

Command	**Meaning**
`ZZ`	Write (save) and quit file
`:x`	Write (save) and quit file
`:wq`	Write (save) and quit file
`:w`	Write (save) file
`:w!`	Write (save) file, overriding protection
`:30,60w newfile`	Write from line 30 through line 60 as *newfile*
`:30,60w>> file`	Write from line 30 through line 60 and append to *file*
`:w %.new`	Write current buffer named *file* as *file.new*

`:q`	Quit file
`:q!`	Quit file, overriding protection
`Q`	Quit *vi* and invoke *ex*
`:e file2`	Edit *file2* without leaving *vi*
`:n`	Edit next file
`:e!`	Return to version of current file at time of last write (save)
`:e #`	Edit alternate file
`:vi`	Invoke *vi* from *ex*
`:`	Invoke one *ex* command from *vi* editor
`%`	Current filename (substitutes into *ex* command line)
`#`	Alternate filename (substitutes into *ex* command line)

Solaris vi Command-Mode Tag Commands

Command	Action
`^]`	Look up the location of the identifier under the cursor in the *tags* file and move to that location; if tag stacking is enabled, the current location is automatically pushed onto the tag stack
`^T`	Return to the previous location in the tag stack, i.e., pop off one element

Buffer Names

Buffer Names	Buffer Use
`1-9`	The last nine deletions, from most to least recent
`a-z`	Named buffers to use as needed; uppercase letters append to buffer

Buffer and Marking Commands

Command	Meaning
`"b`*command*	Do *command* with buffer *b*
`mx`	Mark current position with *x*
`'x`	Move cursor to first character of line marked by *x*
`` `x ``	Move cursor to character marked by *x*
`` `` ```	Return to exact position of previous mark or context
`''`	Return to beginning of the line of previous mark or context

3. Input Mode Shortcuts

Word Abbreviation

`:ab abbr phrase`

Define *abbr* as an abbreviation for *phrase*.

`:unab abbr`

Remove definition of *abbr*.

Be careful with abbreviation texts that either end with the abbreviation name or contain the abbreviation name in the middle.

Command and Input Mode Maps

`:map x sequence`

Define character(s) *x* as a *sequence* of editing commands.

`:unmap x`

Disable *sequence* defined for *x*

`:map`

List the characters that are currently mapped .

`:map! x sequence`

Define character(s) *x* as a *sequence* of editing commands or text that will be recognized in input mode.

`:unmap! x`

Disable the *sequence* defined for the input mode map *x*.

For both command and input mode maps, the map name *x* can take several forms:

One Character

When you type the character, *vi* executes the associated sequence of commands.

Multiple Characters

All the characters must be txped within one second. The value of `notimeout` changes the behaviour.

`#n` Function key notation: a # followed by a digit *n* represents the sequence of characters sent by the terminal's function number *n*

To enter characters such as Escape (^]) or carriage return (^M) first type a **CTRL-V** (^V).

Executable Buffers

Named buffers provide yet another way to create "macros" - complex command sequences you can repeat with a few keystrokes. Here's how it's done:

1. Type a *vi* command sequence or *ex* command *preceded by a colon*; return to command mode.
2. Delete the text into a named buffer.
3. Execute the buffer with the @ command followed by the buffer letter.

The ex command: @buf-name works similarly.

Some versions treat * identically to @ when used from the *ex* command line. In addition, if the buffer character supplied after the @ or * commands is *, the command is taken from the default (unnamed) buffer.

Automatic Indentation

You enable automatic indentation with the command:

```
:set autoindent
```

Four special input sequences affect automatic indentation:

^T	Add one level of indentation; typed in insert mode
^D	Remove one level of indentation; type in insert mode
^ ^D	Shift the cursor back to the beginning of the line, but only for the current line*
0 ^D	Shift the cursor back to the beginning of the line and reset the current auto-indent level to zero+

* ^ ^D and 0 ^D are not in *elvis* 2.0
+ The nvi 1.79 documentation has these two commands switched, but the program actually behaves as described here.

Two commands can be used for shifting source:

<<	Shift a line left eight spaces
>>	Shift a line right eight spaces

The default shift is the value of shiftwidth, usually eight spaces.

4. Substitution and Regular Expressions

The Substitute Command

The general form of the substitute command is:

```
:[addr1][,addr2]s/old/new/[flags]
```

Omitting the search pattern (`:s//replacement/`) uses the last search or substitution regular expression.

An empty replacement part (`:s/pattern//`) "replaces" the matched text with nothing, effectively deleting it from the line.

Substitution flags

Flag	Meaning
`c`	Confirm each substitution
`g`	Change all occurences of *old* to *new* on each line (globally)
`p`	Print the line after the change is made

It's often useful to combine the substitute command with the *ex* global command, `:g`:

```
:g/Object Oriented/s//Buzzword compliant/g
```

vi Regular Expressions

`.` Matches any *single* character except newline. Remember that spaces are treated as characters.

`*` Matches zero of more (as many as there are) of the single character that immediately precedes it. The `*` can follow a metacharacter, such as `.` or a range in brackets.

`^` When used at the start of a regular expression, `^` requires that the following regular expression be found at the beginning of the line. When not at the beginning of a regular expression, `^` stands for itself.

`$` When used at the end of a regular expression, `$` requires that the following regular expression be found at the end of the line. When not at the end of a regular expression, `^` stands for itself.

`\` Treats the following character as an ordinary character. (Use `\\` to get a literal backslash.)

`[ ]` Matches any *one* of the characters enclosed between the brackets. A range of consecutive characters can be specified by separating the first and last characters in the range with a hyphen..

You can include more than one range inside brackets and specify a mix of ranges and separate characters.

Most metacharacters lose their special meaning inside brackets, so you don't need to escape them if you want to use them as ordinary characters. Within brackets, the three metacharacters you still need to escape are `\ - ]`. (The hypen (-) acquires meaning as a range specifier; to use an actual hypen, you can also place it as the first character inside brackets.)

A caret (^) has special meaning only when it's the first character inside brackets, but in this case, the meaning differs from that of the normal `^` metacharacter. As the first character within brackets, a `^` reverses their sense: the brackets match any one character not in the list. For example, `[^a-z]` matches

any character that is not a lowercase letter.

`\( \)` Saves the pattern enclosed between \(and \) into a special holding space or "hold buffer". up to nine patterns can be saved in this way on a single line.

You can also use the \n notation within a search or substitute string:

```
:s/\(abcd\)\1/alphabet-soup/
```

changes `abcdabcd` into `alphabet-soup`.*

* This works with *vi*, *nvi*, and *vim*, but not with *elvis* 2.0 , *vile* 7.4, or *vile* 8.0.

`\< \>` Matches characters at the beginning (`\<`) or end (`\>`) of a word. The end or beginning of a word determined either by a punctuation mark or by a space. Unlike `\(...\)`, these don't have to be used in matched pairs.

`~` Matches whatever regular expression was used in the *last* search.

POSIX Brackets Expressions

POSIX bracket expressions may contain the following:

Character classes

A POSIX character class consists of keywords bracketed by `[:` and `:]`. The keywords describe different classes of characters such as alphabetic characters, control characters, and so on (see the following table).

Collating symbols

A collating symbol is a multicharacter sequence that should be treated as a unit. It consists of the characters bracketed by `[,` and `,]`.

Equivalence classes

An equivalence class lists a set of characters that should be considered equivalent, such as `e` and `è`. It consists of a named element from the locale, bracketed by `[=` and `=]`.

All three contructs must appear inside square brackets of a bracket expression.

POSIX character classes

Class	Matching Characters
`[:alnum:]`	Alphanumeric characters
`[:alpha:]`	Alphabetic characters
`[:blank:]`	Space and tab characters
`[:cntrl:]`	Control characters

`[:digit:]`	Numeric characters
`[:graph:]`	Printable and visible (nonspace) characters
`[:lower:]`	lowercase characters
`[:print:]`	Printable characters (includes whitespace)
`[:punct:]`	Punctuation characters
`[:space:]`	Whitespace characters
`[:upper:]`	Uppercase characters
`[:xdigit:]`	Hexdecimal digits

Metacharacters Used in Replacement Strings

`\n`	Is replaced with the text matched by the *n*th pattern previously saved by `\(` and `\)`, where *n* is a number from 1 to 9, and previously saved patterns (kept in hold buffers) are counted from the left on the line.
`\`	Treats the following special characters as an ordinary character. To specify a real backslash, type two in a row (`\\`).
`&`	Is replaced with the entire text matched by the search pattern when used in a replacement string. This is useful when you want to avoid retyping text.
`~`	The string found is replaced with the replacement text specified in the last substitute command. This is useful for repeating an edit.
`\u or \l`	Changes the next character in the replacement string to upper- or lowercase, respectively.
`\U or \L, \e or \E`	`\U` and `\L` are similar to `\u` or `\l`, but all following characters are converted to upper- or lowercase until the end of the replacement string or until `\e` or `\E` is reached. If there is no `\e` or `\E`, all characters of the replacement text affected by the `\U` or `\L`.

More Substitution Tricks

- You can instruct *vi* to ignore case by typing: `set ic`.
- A simple `:s` is the same as `:s//~/`.
- `:&` is the same as `:s`. You can follow the `&` with a `g` to make the subsitution globally on the line, and even use it with a line range.
- The **&** key can be used as a *vi* command to perform the `:&` command, i.e., to repeat the last substitution.
- The `:~` command is similar to the `:&` command, but with a subtle difference. The search pattern used is the last regular expression used in any command, not necessarily the one used in the last substitute command.
- Besides the `/` character, you may use any nonalphanumerical, nonwhitespace character as your delimiter, except backslash, double-quote, and the vertical bar (`\`, `"`, and `|`).
- When the `edcompatible` option is enabled, *vi* remembers the flags (`g` for global and `c` for confirmation) used on the last substitute and applies then to the next one.

5. ex Commands

Command Syntax

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

Address Symbols

Address	Inlcudes
`1,$`	All lines in the file
`x,y`	Lines *x* through *y*
`x;y`	Lines *x* through *y*, with current line reset to *x*
`0`	Top of the file
`.`	Current line
`n`	Absolute line number *n*
`$`	Last line
`%`	All lines; same as `1,$`
`x-n`	*n* lines before *x*
`x+n`	*n* lines after *x*
`-[n]`	One or *n* lines previous
`+[n]`	One or *n* lines ahead
`'x`	Line marked with *x*
`''`	Previous mark
`/pat/` or `?pat?`	Ahead or back to line where *pat* matches

Command Option Symbols

Symbol	Meaning
`!`	A variant form of the command
`count`	Repeat the command *count* times
`file`	Filename: `%` is current file, `#` is previous file

Alphabetical List of Commands

Full Name	Command
Abbrev	`ab [string text]`
Append	`[address]a[!]`

	`text` `.`
Args	`ar`
Change	`[address]c[!]` `text` `.`
Copy	`[address]co destination`
Delete	`[address]d [buffer]`
Edit	`e[!][+n] [filename]`
File	`f [filename]`
Global	`[address]g[!]/pattern/[commands]`
Insert	`[address]i[!]` `text` `.`
Join	`[address]j[!][count]`
K (mark)	`[address]k char`
List	`[address]l [count]`
Map	`map char commands`
Mark	`[address]ma char`
Move	`[address]m destination`
Next	`n[!][[+command] filelist]`
Number	`[address]nu [count]`
Open	`[address]o [/pattern]`
Preserve	`pre`
Print	`[address]P [count]` `[address]p [count]`
Put	`[address]pu [char]`
Quit	`q[!]`
Read	`[address]r filename`
Read	`[address]r ! command`
Recover	`rec`
Rewind	`rew[!]`
Set	`set` `set option` `set nooption` `set option=value` `set option?`
Shell	`sh`
Source	`so filename`
Substitute	`[addr]s [/pat/repl/][opts]`
T (to)	`[address]t destination`
Tag	`[address]ta tag`
Unabbreviate	`una word`

Undo	`u`
Unmap	`unm char`
V (global exclude)	`[address]v/pattern/[commands]`
Version	`ve`
Visual	`[address]vi[type][count]`
Visual	`vi [+n] [filename]`
Write	`[address]w[!] [[>>]filename]`
Write	`[address]w !command`
Wq	`wq[!]`
Xit	`x`
Yank	`[address]y[char][count]`
Z (position line)	`[address]z[type][count]` *type* can be one of: `+` Place line at the top of the window (default) `-` Place line at bottom of the window `.` Place line in the center of the window `^` Print the previous window `=` Place line in the center of the window and leave the current line at this line
!	`[address]!command`
= (line number)	`[address]=`
<> (shift)	`[address]<[count]` `[address]>[count]`
Address	`address`
Return (next line)	**`RETURN`**
&	`[address]& [options][count]` repeat substitute
~	`[address]~[count]` Like `&`, but with last used regular expression; for details, see Chapter 6 of *Learning the vi Editor*

6. Initialization and Recovery

Initialization

vi performs the following initialization steps:

1. If the `EXINIT` environment variable exist, execute the commands it contains. Separate multiple commands by a pipe symboll (`|`).

2. If `EXINIT` doesn't exist, look for file *$HOME/.exrc*. If it exists, read and execute it.
3. If either `EXINIT` or *$HOME/.exrc* turns on the exrc option, read and execute the file *./*.exrc, if it exists
4. Execute search or goto command given with `+/pattern` or `+n` command-line options (POSIX: `-c` option).

The *.exrc* files are simple scripts of *ex* commands; they don't need a leading colon. You can put comments in your scripts by starting a line with a double quote (`"`). This is recommended.

Recovery

The commands `ex -r` or `vi -r` list any files you can recover. You then use the command:

```
$ vi -r file
```

to recover a particular *file*.

Even without a crash, you can force the system to preserve your buffer by using the command `:pre` (preserve).

7. vi Options

autoindent	(ai)	noai
autoprint	(ap)	ap
autowrite	(aw)	noaw
beautify	(bf)	nobf
directory	(dir)	*/tmp*
edcompatible		noedcompatible
errorbells	(eb)	errorbells
exrc (ex)	(ex)	noexrc
hardtabs	(ht)	8
ignorecase	(ic)	noic
lisp		nolisp
list		nolist
magic		magic
novice		nonovice
number	(nu)	nonu
open		open
optimize	(opt)	noopt
paragraphs	(para)	IPLPPPQP LIpplpipbp
prompt		prompt
readonly	(ro)	noro
read	(re)	
remap		remap

report		5
scroll		half window
sections	(sect)	SHNHH HU
shell	(sh)	*/bin/sh*
shiftwidth	(sw)	8
showmatch	(sm)	nosm
showmode		noshowmode
slowopen	(slow)	
tabstop	(ts)	8
taglength	(tl)	0
tags		*tags /usr/lib/tags*
tagstack		tagstack
term		(from $TERM)
terse		noterse
timeout	(to)	timeout
ttytype		(from $TERM)
warn		warn
window	(w)	
wrapscan	(ws)	ws
wrapmargin	(wm)	0
writeany	(wa)	nowa

8. Enhanced Tags and Tag Stacks (not in here)

Buy the book!

9. nvi - New vi

Buy the book!

10. elvis

Buy the book!

11. vim - vi Improved

Buy the book!

12. vile - vi Like Emacs

Buy the book!

13. Clone Source and Contact Information

Editor *nvi*
Author Keith Bostic
Email *bostic@bostic.com*
Source *http://www.bostic.com/vi/*

Editor *elvis*
Author Steve Kirkendall
Email *kirkenda@cs.pdx.edu*
Source *ftp://ftp.cs.pdx.edu/pub/elvis/README.html*

Editor *vim*
Author Bram Moolenaar
Email *Bram@vim.org*
Source *http://www.vim.org/*

Editor *vile*
Author Kevin Buettner, Tom Dickley, and Paul Fox
Email *vile-bugs@foxharp.boston.ma.us*
Source *http://www.clark.net/pub/dickey/vile/vile.html*