

Warning: some vi versions don't support the more esoteric features described in this document. You can edit/redistribute this document, as long as you don't make false claims on original authorship.

Author: Maarten Litmaath <maart@nikhef.nl>
Version: 8

vi Reference Table of Contents

- [Contributions](#)
- [Legenda](#)
- [Move commands](#)
- [Searching](#)
- [Undoing changes](#)
- [Appending text](#)
- [Deleting text](#)
- [Changing text](#)
- [Substitute replacement patterns](#)
- [Remembering text \(yanking\)](#)
- [Commands while in append/change mode](#)
- [Writing, editing other files, and quitting vi](#)
- [Display commands](#)
- [Mapping and abbreviation](#)
- [Switch and shell commands](#)
- [vi startup](#)
- [The most important options](#)

contributions

Rich Salz <rsalz@bbn.com>
Eamonn McManus <emcmanus@cs.tcd.ie>
Diomidis Spinellis <diomidis%ecrcvax.uucp@pyramid.pyramid.com>
Blair P. Houghton <bph@buengc.bu.edu>
Rusty Haddock <{uunet,att,rutgers}!mimsy.umd.edu!fe2o3!rusty>
Panos Tsirigotis <panos@boulder.colorado.edu>
David J. MacKenzie <djm@wam.umd.edu>
Kevin Carothers <kevin@ttidca.tti.com>
Dan Mercer <mercer@ncrcce.StPaul.NCR.COM>
Ze'ev Shtadler <steed@il4cad.intel.com>
Paul Quare <pq@r2.cs.man.ac.uk>
Dave Beyerl <att!ihlpl!db2l>
David C Johnson <johnson@wrs.com>
Lee Sailer <UH2@psuvm.psu.edu>
David Gast <gast@cs.ucla.edu>

legenda

default values	: 1
<*>	: '*' must not be taken literally
[*]	: '*' is optional
^X	: <ctrl>X
<sp>	: space
<cr>	: carriage return
<lf>	: linefeed
<ht>	: horizontal tab
<esc>	: escape
<erase>	: your erase character
<kill>	: your kill character
<intr>	: your interrupt character
<a-z>	: an element in the range
N	: number ('*' = allowed, '-' = not appropriate)
CHAR	: char unequal to <ht> <sp>

WORD : word followed by <ht>|<sp>|<lf>

move commands

N	Command	Meaning
*	h ^H <erase>	<*> chars to the left.
*	j <lf> ^N	<*> lines downward.
*	l <sp>	<*> chars to the right.
*	k ^P	<*> lines upward.
*	\$	To the end of line <*> from the cursor.
-	^	To the first CHAR of the line.
*	-	To the first CHAR <*> - 1 lines lower.
*	-	To the first CHAR <*> lines higher.
*	+ <cr>	To the first CHAR <*> lines lower.
-	0	To the first char of the line.
*		To column <*> (<ht>: only to the endpoint).
*	f<char>	<*> <char>s to the right (find).
*	t<char>	Till before <*> <char>s to the right.
*	F<char>	<*> <char>s to the left.
*	T<char>	Till after <*> <char>s to the left.
*	;	Repeat latest `f' `t' `F' `T' <*> times.
*	,	Idem in opposite direction.
*	w	<*> words forward.
*	W	<*> WORDS forward.
*	b	<*> words backward.
*	B	<*> WORDS backward.
*	e	To the end of word <*> forward.
*	E	To the end of WORD <*> forward.
*	G	Go to line <*> (default EOF).
*	H	To line <*> from top of the screen (home).
*	L	To line <*> from bottom of the screen (last).
-	M	To the middle line of the screen.
*)	<*> sentences forward.
*	(<*> sentences backward.
*	}	<*> paragraphs forward.
*	{	<*> paragraphs backward.
-]]	To the next section (default EOF).
-	[[To the previous section (default begin of file).
-	`<a-z>	To the mark.
-	'<a-z>	To the first CHAR of the line with the mark.
-	``	To the cursor position before the latest absolute jump (of which are examples `/' and `G').
-	''	To the first CHAR of the line on which the cursor was placed before the latest absolute jump.
-	/<string>	To the next occurrence of <string>.
-	?<string>	To the previous occurrence of <string>.
-	n	Repeat latest `/' `?' (next).
-	N	Idem in opposite direction.
-	%	Find the next bracket and go to its match (also with `{' `}' and `[' `]').

searching (see above)

:ta <name>	Search in the tags file[s] where <name> is defined (file, line), and go to it.
^]	Use the name under the cursor in a `:ta' command.
^T	Pop the previous tag off the tagstack and return to its position.
: [x,y]g/<string>/<cmd>	Search globally [from line x to y] for <string> and execute the `ex' <cmd> on each occurrence.
: [x,y]v/<string>/<cmd>	Execute <cmd> on the lines that don't match.

undoing changes

u	Undo the latest change.
U	Undo all changes on a line, while not having moved off it (unfortunately).
:q!	Quit vi without writing.
:e!	Re-edit a messed-up file.

appending text (end with <esc>)

* a	<*> times after the cursor.
* A	<*> times at the end of line.
* i	<*> times before the cursor (insert).
* I	<*> times before the first CHAR of the line
* o	On a new line below the current (open). The count is only useful on a slow terminal.
* O	On a new line above the current. The count is only useful on a slow terminal.
* ><move>	Shift the lines described by <*><move> one shiftwidth to the right.
* >>	Shift <*> lines one shiftwidth to the right.
* ["<a-zA-Z1-9>]p	Put the contents of the (default undo) buffer <*> times after the cursor. A buffer containing lines is put only once, below the current line.
* ["<a-zA-Z1-9>]P	Put the contents of the (default undo) buffer <*> times before the cursor. A buffer containing lines is put only once, above the current line.
* .	Repeat previous command <*> times. If the last command before a `.' command references a numbered buffer, the buffer number is incremented first (and the count is ignored): "lpu.u.u.u.u - `walk through' buffers 1 through 5 "lP.... - restore them

deleting text

Everything deleted can be stored into a buffer. This is achieved by putting a ``' and a letter <a-z> before the delete command. The deleted text will be in the buffer with the used letter. If <A-Z> is used as buffer name, the conjugate buffer <a-z> will be augmented instead of overwritten with the text. The undo buffer always contains the latest change. Buffers <1-9> contain the latest 9 LINE deletions (``1' is most recent).

* x	Delete <*> chars under and after the cursor.
* X	<*> chars before the cursor.
* d<move>	From begin to endpoint of <*><move>.
* dd	<*> lines.
- D	The rest of the line.
* <<move>	Shift the lines described by <*><move> one shiftwidth to the left.
* <<	Shift <*> lines one shiftwidth to the left.
* .	Repeat latest command <*> times.

changing text (end with <esc>)

* r<char>	Replace <*> chars by <char> - no <esc>.
* R	Overwrite the rest of the line, appending change <*> - 1 times.
* s	Substitute <*> chars.
* S	<*> lines.
* c<move>	Change from begin to endpoint of <*><move>.
* cc	<*> lines.
* C	The rest of the line and <*> - 1 next lines.
* =<move>	If the option `lisp' is set, this command will realign the lines described by <*><move> as though they had been typed with the option `ai' set too.
- ~	Switch lower and upper cases (should be an operator, like `c').
* J	Join <*> lines (default 2).
* .	Repeat latest command <*> times (`J' only once).
- &	Repeat latest `ex' substitute command, e.g. `s/wrong/good'.
- :[x,y]s/<p>/<r>/<f>	Substitute (on lines x through y) the pattern <p>

```

| (default the last pattern) with <r>. Useful
| flags <f> are `g' for `global' (i.e. change
| every non-overlapping occurrence of <p>) and
| `c' for `confirm' (type `y' to confirm a
| particular substitution, else <cr>). Instead
| of `/' any punctuation CHAR unequal to <lf>
| can be used as delimiter.

```

substitute replacement patterns

The basic meta-characters for the replacement pattern are `&' and `~'; these are given as `\\&' and `\\~' when nomagic is set. Each instance of `&' is replaced by the characters which the regular expression matched. The meta-character `~' stands, in the replacement pattern, for the defining text of the previous replacement pattern. Other meta-sequences possible in the replacement pattern are always introduced by the escaping character `\\'. The sequence `\\n' (with `n' in [1-9]) is replaced by the text matched by the n-th regular subexpression enclosed between `\\(' and `\\)'. The sequences `\\u' and `\\l' cause the immediately following character in the replacement to be converted to upper- or lower-case respectively if this character is a letter. The sequences `\\U' and `\\L' turn such conversion on, either until `\\E' or `\\e' is encountered, or until the end of the replacement pattern.

remembering text (yanking)

With yank commands you can put `"<a-zA-Z>' before the command, just as with delete commands. Otherwise you only copy to the undo buffer. The use of buffers <a-z> is THE way of copying text to another file; see the `:e <file>' command.

```

* | y<move>          | Yank from begin to endpoint of <*><move>.
* | yy              | <*> lines.
* | Y               | Idem (should be equivalent to `y$' though).
- | m<a-z>          | Mark the cursor position with a letter.

```

commands while in append/change mode

```

^@          | If typed as the first character of the
            | insertion, it is replaced with the previous
            | text inserted (max. 128 chars), after which
            | the insertion is terminated.
^V          | Deprive the next char of its special meaning
            | (e.g. <esc>).
^D          | One shiftwidth to the left, but only if
            | nothing else has been typed on the line.
0^D         | Remove all indentation on the current line
            | (there must be no other chars on the line).
^^D         | Idem, but it is restored on the next line.
^T          | One shiftwidth to the right, but only if
            | nothing else has been typed on the line.
^H | <erase>    | One char back.
^W          | One word back.
<kill>      | Back to the begin of the change on the
            | current line.
<intr>      | Like <esc> (but you get a beep as well).

```

writing, editing other files, and quitting vi

In `:` `ex' commands - if not the first CHAR on the line - `%` denotes the current file, `#` is a synonym for the alternate file (which normally is the previous file). As first CHAR on the line `%` is a shorthand for `1,\$'. Marks can be used for line numbers too: '<a-z>. In the `:w'|`:f'|`:cd'|`:e'|`:n' commands shell meta-characters can be used.

```

:q          | Quit vi, unless the buffer has been changed.
:q!         | Quit vi without writing.

```

<code>^Z</code>	Suspend vi.
<code>:w</code>	Write the file.
<code>:w <name></code>	Write to the file <name>.
<code>:w >> <name></code>	Append the buffer to the file <name>.
<code>:w! <name></code>	Overwrite the file <name>.
<code>:x,y w <name></code>	Write lines x through y to the file <name>.
<code>:wq</code>	Write the file and quit vi; some versions quit even if the write was unsuccessful! Use <code>`ZZ'</code> instead.
<code>ZZ</code>	Write if the buffer has been changed, and quit vi. If you have invoked vi with the <code>`-r'</code> option, you'd better write the file explicitly (<code>`w'</code> or <code>`w!'</code>), or quit the editor explicitly (<code>`q!'</code>) if you don't want to overwrite the file - some versions of vi don't handle the <code>`recover'</code> option very well.
<code>:x [<file>]</code>	Idem [but write to <file>].
<code>:x! [<file>]</code>	<code>`:w! [<file>]'</code> and <code>`:q'</code> .
<code>:pre</code>	Preserve the file - the buffer is saved as if the system had just crashed; for emergencies, when a <code>:w'</code> command has failed and you don't know how to save your work (see vi -r).
<code>:f <name></code>	Set the current filename to <name>.
<code>:cd [<dir>]</code>	Set the working directory to <dir> (default home directory).
<code>:cd! [<dir>]</code>	Idem, but don't save changes.
<code>:e [+<cmd>] <file></code>	Edit another file without quitting vi - the buffers are not changed (except the undo buffer), so text can be copied from one file to another this way. [Execute the <code>`ex'</code> command <cmd> (default <code>`\$'</code>) when the new file has been read into the buffer.] <cmd> must contain no <sp> or <ht>. See vi startup .
<code>:e! [+<cmd>] <file></code>	Idem, without writing the current buffer.
<code>^^</code>	Edit the alternate (normally the previous) file.
<code>:rew</code>	Rewind the argument list, edit the first file.
<code>:rew!</code>	Idem, without writing the current buffer.
<code>:n [+<cmd>] [<files>]</code>	Edit next file or specify a new argument list.
<code>:n! [+<cmd>] [<files>]</code>	Idem, without writing the current buffer.
<code>:args</code>	Give the argument list, with the current file between <code>`['</code> and <code>`]'</code> .

display commands

<code>^G</code>	Give file name, status, current line number and relative position.
<code>^L</code>	Refresh the screen (sometimes <code>^P'</code> or <code>^R'</code>).
<code>^R</code>	Sometimes vi replaces a deleted line by a <code>^@'</code> , to be deleted by <code>^R'</code> (see option redraw).
<code>[*]^E</code>	Expose <*> more lines at bottom, cursor stays put (if possible).
<code>[*]^Y</code>	Expose <*> more lines at top, cursor stays put (if possible).
<code>[*]^D</code>	Scroll <*> lines downward (default the number of the previous scroll; initialization: half a page).
<code>[*]^U</code>	Scroll <*> lines upward (default the number of the previous scroll; initialization: half a page).
<code>[*]^F</code>	<*> pages forward.
<code>[*]^B</code>	<*> pages backward (in older versions <code>^B'</code> only works without count).

If in the next commands the field <wi> is present, the window size will change to <wi>. The window will always be displayed at the bottom of the screen.

<code>[*]z[wi]<cr></code>	Put line <*> at the top of the window (default the current line).
<code>[*]z[wi]+</code>	Put line <*> at the top of the window (default the first line of the next page).
<code>[*]z[wi]-</code>	Put line <*> at the bottom of the window (default the current line).
<code>[*]z[wi]^</code>	Put line <*> at the bottom of the window

[*]z[wi].	(default the last line of the previous page). Put line <*> in the centre of the window (default the current line).
-----------	--

mapping and abbreviation

When mapping take a look at the options `to' and `remap' (below).

:map <string> <seq>	<string> is interpreted as <seq>, e.g. `:map ^C :!cc %^V<cr>' to invoke `cc' (the C compiler) from within the editor (vi replaces `%` with the current file name).
:map	Show all mappings.
:unmap <string>	Deprive <string> of its mapping. When vi complains about non-mapped macros (whereas no typos have been made), first do something like `:map <string> Z', followed by `:unmap <string>' (`Z' must not be a macro itself), or switch to `ex' mode first with `Q'.
:map! <string> <seq>	Mapping in append mode, e.g. `:map! \be begin^V<cr>end;^V<esc>O<ht>'. When in append mode <string> is preceded by `^V', no mapping is done.
:map!	Show all append mode mappings.
:unmap! <string>	Deprive <string> of its mapping (see `:unmap').
:ab <string> <seq>	Whenever in append mode <string> is preceded and followed by a breakpoint (e.g. <sp> or `,'), it is interpreted as <seq>, e.g. `:ab ^P procedure'. A `^V' immediately following <string> inhibits expansion.
:ab	Show all abbreviations.
:unab <string>	Do not consider <string> an abbreviation anymore (see `:unmap').
@<a-z>	Consider the contents of the named register a command, e.g.: o0^D:s/wrong/good/<esc>"zdd Explanation: o - open a new line 0^D - remove indentation :s/wrong/good/ - this input text is an `ex' substitute command <esc> - finish the input "zdd - delete the line just created into register `z' Now you can type `@z' to replace `wrong' with `good' on the current line.
@@	Repeat last register command.

switch and shell commands

Q ^\ <intr><intr>	Switch from vi to `ex'.
:	An `ex' command can be given.
:vi	Switch from `ex' to vi.
:sh	Execute a subshell, back to vi by `^D'.
:[x,y]!<cmd>	Execute a shell <cmd> [on lines x through y; these lines will serve as input for <cmd> and will be replaced by its standard output].
:[x,y]!! [<args>]	Repeat last shell command [and append <args>].
:[x,y]!<cmd> ! [<args>]	Use the previous command (the second `!') in a new command.
[*]!<move><cmd>	The shell executes <cmd>, with as standard input the lines described by <*><move>, next the standard output replaces those lines (think of `cb', `sort', `nroff', etc.).
[*]!<move>!<args>	Append <args> to the last <cmd> and execute it, using the lines described by the current <*><move>.
[*]!!<cmd>	Give <*> lines as standard input to the shell <cmd>, next let the standard output replace those lines.
[*]!!! [<args>]	Use the previous <cmd> [and append <args> to it].
:x,y w !<cmd>	Let lines x to y be standard input for <cmd> (notice the <sp> between the `w' and the `!').

:r!<cmd>	Put the output of <cmd> onto a new line.
:r <name>	Read the file <name> into the buffer.

vi startup

vi [<files>]	Edit the files, start with the first page of the first file.
--------------	---

The editor can be initialized by the shell variable `EXINIT', which looks like:

```
EXINIT='<cmd>|<cmd>|...'
<cmd>: set options
      map ...
      ab ...
export EXINIT (in the Bourne shell)
```

However, the list of initializations can also be put into a file. If this file is located in your home directory, and is named `.exrc' AND the variable `EXINIT' is NOT set, the list will be executed automatically at startup time. However, vi will always execute the contents of a `.exrc' in the current directory, if you own the file. Else you have to give the execute (`source') command yourself:

```
:so file
```

In a `.exrc' file a comment is introduced with a double quote character: the rest of the line is ignored. Exception: if the last command on the line is a `map[!]' or `ab' command or a shell escape, a trailing comment is not recognized, but considered part of the command.

On-line initializations can be given with `vi +<cmd> file', e.g.:

vi +x file	The cursor will immediately jump to line x (default last line).
vi +/<string> file	Jump to the first occurrence of <string>.

You can start at a particular tag with:

vi -t <tag>	Start in the right file in the right place.
-------------	---

Sometimes (e.g. if the system crashed while you were editing) it is possible to recover files lost in the editor by `vi -r file'. A plain `vi -r' command shows the files you can recover. If you just want to view a file by using vi, and you want to avoid any change, instead of vi you can use the `view' or `vi -R' command: the option `readonly' will be set automatically (with `:w!' you can override this option).

the most important options

ai	autoindent - In append mode after a <cr> the cursor will move directly below the first CHAR on the previous line. However, if the option `lisp' is set, the cursor will align at the first argument to the last open list.
aw	autowrite - Write at every shell escape (useful when compiling from within vi).
dir=<string>	directory - The directory for vi to make temporary files (default `/tmp').
eb	errorbells - Beeps when you goof (not on every terminal).
ic	ignorecase - No distinction between upper and lower cases when searching.
lisp	Redefine the following commands: `(', `)` - move backward (forward) over S-expressions `{', `}' - idem, but don't stop at atoms `[', `]' - go to previous (next) line beginning with a `('
list	See option `ai'.
magic	<lf> is shown as `\$', <ht> as `^I'. If this option is set (default), the chars `.',

	<pre> `[' and `*' have special meanings within search and `ex' substitute commands. To deprive such a char of its special function it must be preceded by a `\''. If the option is turned off it's just the other way around. Meta-chars: ^<string> - <string> must begin the line <string>\$ - <string> must end the line . - matches any char [a-z] - matches any char in the range [^a-z] - any char not in the range [<string>] - matches any char in <string> [^<string>] - any char not in <string> <char>* - 0 or more <char>s \<<string> - <string> must begin a word <string>\> - <string> must end a word </pre>
modeline	<p>When you read an existing file into the buffer, and this option is set, the first and last 5 lines are checked for editing commands in the following form:</p> <pre> <sp>vi:set options map ... ab ... !...: </pre> <p>Instead of <sp> a <ht> can be used, instead of `vi' there can be `ex'. Warning: this option could have nasty results if you edit a file containing `strange' modelines.</p>
nu	number - Numbers before the lines.
para=<string>	<p>paragraphs - Every pair of chars in <string> is considered a paragraph delimiter nroff macro (for `{ ' and `} '). A <sp> preceded by a `{ ' indicates the previous char is a single letter macro. `:set para=P\ bp' introduces `.P' and `.bp' as paragraph delimiters. Empty lines and section boundaries are paragraph boundaries too.</p>
redraw	The screen remains up to date.
remap	<p>If on (default), macros are repeatedly expanded until they are unchanged.</p> <p>Example: if `o' is mapped to `A', and `A' is mapped to `I', then `o' will map to `I' if `remap' is set, else it will map to `A'.</p>
report=<*>	Vi reports whenever e.g. a delete or yank command affects <*> or more lines.
ro	<p>readonly - The file is not to be changed.</p> <p>However, `:w!' will override this option.</p>
sect=<string>	sections - Gives the section delimiters (for `[' and `] '); see option `para'. A `{ ' beginning a line also starts a section (as in C functions).
sh=<string>	shell - The program to be used for shell escapes (default `\$SHELL' (default `/bin/sh')).
sw=<*>	shiftwidth - Gives the shiftwidth (default 8 positions).
sm	<p>showmatch - Whenever you append a `)', vi shows its match if it's on the same page; also with `{ ' and `} '. If there's no match at all, vi will beep.</p>
taglength=<*>	The number of significant characters in tags (0 = unlimited).
tags=<string>	The space-separated list of tags files.
terse	Short error messages.
to	<p>timeout - If this option is set, append mode mappings will be interpreted only if they're typed fast enough.</p>
ts=<*>	tabstop - The length of a <ht>; warning: this is only IN the editor, outside of it <ht>s have their normal length (default 8 positions).
wa	writeany - No checks when writing (dangerous).
warn	Warn you when you try to quit without writing.
wi=<*>	window - The default number of lines vi shows.
wm=<*>	<p>wrapmargin - In append mode vi automatically puts a <lf> whenever there is a <sp> or <ht> within <wm> columns from the right margin (0 = don't put a <lf> in the file, yet put it on the screen).</p>
ws	<p>wrapscreen - When searching, the end is considered `stuck' to the begin of the file.</p>

:set <option>	Turn <option> on.
:set no<option>	Turn <option> off.
:set <option>=<value>	Set <option> to <value>.
:set	Show all non-default options and their values.
:set <option>?	Show <option>'s value.
:set all	Show all options and their values.

Converted to html by Jeff Aspinall (aspinall@umich.edu) with help from [Meng Weng Wong's txt2html script](#), 22 June 1994

