VSCodeVim / **Vim**

Branch: **master** ▾ Vim / **ROADMAP.md** Find file Copy path

max-sixty Add ctrl-w o to Roadmap (#4668) 324a605 on Mar 25

19 contributors

```
592 lines (505 sloc)   62.2 KB
```

Raw Blame History

Key

✔️ - command done

✔️ ⭐ - command done with VS Code specific customization

⚠️ - some variations of the command are not supported

🏃 - work in progress

⬇️ - command is low priority; open an issue (or thumbs up the relevant issue) if you want to see it sooner

❌ - command impossible with current VSCode API

🔢 - command accepts numeric prefix

Roadmap

These are the big Vim features, put generally in the order in which we plan to implement them.

Status	Command
✔️	Normal Mode
✔️	Insert Mode
✔️	Visual Mode
✔️	Visual Line Mode
✔️	Number Prefixes
✔️	. Operator
✔️	Searching with / and ?
✔️	Correct Undo/Redo
⚠️	Command Remapping
⚠️	Marks
✔️	Text Objects
✔️	Visual Block Mode
✔️	Replace Mode

Status	Command
✔️	Multiple Select Mode
⚠️	Macros
⚠️	Buffer/Window/Tab

Now follows an exhaustive list of every known Vim command that we could find.

Custom commands

- `gh` - show the hover tooltip.
- `gb` - add an additional cursor at the next place that matches `*`.

Left-right motions

Status	Command	Description
✔️	🔢 h	left (also: CTRL-H, BS, or Left key)
✔️	🔢 l	right (also: Space or Right key)
✔️	0	to first character in the line (also: Home key)
✔️	^	to first non-blank character in the line

Status	Command	Description
✔️	🔢 $	to the last character in the line (N-1 lines lower) (also: End key)
✔️	g0	to first character in screen line (differs from "0" when lines wrap)
✔️	g^	to first non-blank character in screen line (differs from "^" when lines wrap)
✔️	🔢 g$	to last character in screen line (differs from "$" when lines wrap)
✔️	gm	to middle of the screen line
✔️	🔢 |	to column N (default: 1)
✔️	🔢 f{char}	to the Nth occurrence of {char} to the right
✔️	🔢 F{char}	to the Nth occurrence of {char} to the left
✔️	🔢 t{char}	till before the Nth occurrence of {char} to the right
✔️	🔢 T{char}	till before the Nth occurrence of {char} to the left
✔️	🔢 ;	repeat the last "f", "F", "t", or "T" N times
✔️	🔢 ,	repeat the last "f", "F", "t", or "T" N times in opposite direction

Up-down motions

Status	Command	Description

Status	Command	Description
✔️	🔢 k	up N lines (also: CTRL-P and Up)
✔️	🔢 j	down N lines (also: CTRL-J, CTRL-N, NL, and Down)
✔️	🔢 -	up N lines, on the first non-blank character
✔️	🔢 +	down N lines, on the first non-blank character (also: CTRL-M and CR)
✔️	🔢 _	down N-1 lines, on the first non-blank character
✔️	🔢 G	goto line N (default: last line), on the first non-blank character
✔️	🔢 gg	goto line N (default: first line), on the first non-blank character
✔️	🔢 %	goto line N percentage down in the file; N must be given, otherwise it is the `%` command
✔️	🔢 gk	up N screen lines (differs from "k" when line wraps)
✔️	🔢 gj	down N screen lines (differs from "j" when line wraps)

Text object motions

Status	Command	Description
✔️	🔢 w	N words forward
✔️	🔢 W	N blank-separated WORDs forward

Status	Command	Description
✔️	🔢 e	N words forward to the end of the Nth word
✔️	🔢 E	N words forward to the end of the Nth blank-separated WORD
✔️	🔢 b	N words backward
✔️	🔢 B	N blank-separated WORDs backward
✔️	🔢 ge	N words backward to the end of the Nth word
✔️	🔢 gE	N words backward to the end of the Nth blank-separated WORD
✔️	🔢)	N sentences forward
✔️	🔢 (	N sentences backward
✔️	🔢 }	N paragraphs forward
✔️	🔢 {	N paragraphs backward
✔️	🔢]]	N sections forward, at start of section
✔️	🔢 [[	N sections backward, at start of section
✔️	🔢][	N sections forward, at end of section
✔️	🔢 []	N sections backward, at end of section
✔️	🔢 [(	N times back to unclosed '('

Status	Command	Description
✔️	🔢 [{	N times back to unclosed '{'
⬇️	🔢 [m	N times back to start of method (for Java)
⬇️	🔢 [M	N times back to end of method (for Java)
✔️	🔢])	N times forward to unclosed ')'
✔️	🔢]}	N times forward to unclosed '}'
⬇️	🔢]m	N times forward to start of method (for Java)
⬇️	🔢]M	N times forward to end of method (for Java)
⬇️	🔢 [#	N times back to unclosed "#if" or "#else"
⬇️	🔢]#	N times forward to unclosed "#else" or "#endif"
⬇️	🔢 [*	N times back to start of a C comment "/*"
⬇️	🔢]*	N times forward to end of a C comment "*/"

Pattern searches

Status	Command	Description	Note
✔️ ⭐	🔢 `/{pattern}`	search forward for the Nth	Currently we only support JavaScript Regex

Status	Command	Description	Note
	`[/[offset]]<CR>`	occurrence of {pattern}	but not Vim's in-house Regex engine.
✔️ ⭐	🔢 `?{pattern}[?[offset]]<CR>`	search backward for the Nth occurrence of {pattern}	Currently we only support JavaScript Regex but not Vim's in-house Regex engine.
⚠️	🔢 `/<CR>`	repeat last search, in the forward direction	{count} is not supported.
⚠️	🔢 `?<CR>`	repeat last search, in the backward direction	{count} is not supported.
✔️	🔢 n	repeat last search	
✔️	🔢 N	repeat last search, in opposite direction	
✔️	🔢 *	search forward for the identifier under the cursor	
✔️	🔢 #	search backward for the identifier under the cursor	
✔️	🔢 g*	like "*", but also find partial matches	
✔️	🔢 g#	like "#", but also find partial matches	

Status	Command	Description	Note
✔️	gd	goto local declaration of identifier under the cursor	
⬇️	gD	goto global declaration of identifier under the cursor	

Marks and motions

Status	Command	Description
✔️	m{a-zA-Z}	mark current position with mark {a-zA-Z}
✔️	`{a-z}	go to mark {a-z} within current file
✔️	`{A-Z}	go to mark {A-Z} in any file
✔️	`{0-9}	go to the position where Vim was previously exited
✔️	``	go to the position before the last jump
⬇️	`"	go to the position when last editing this file
✔️	`[	go to the start of the previously operated or put text
✔️	'[	go to the start of the previously operated or put text
✔️	`]	go to the end of the previously operated or put text

Status	Command	Description
✔️	']	go to the end of the previously operated or put text
⬇️	`<	go to the start of the (previous) Visual area
⬇️	`>	go to the end of the (previous) Visual area
✔️	`.	go to the position of the last change in this file
✔️	'.	go to the position of the last change in this file
⬇️	'{a-zA-Z0-9[]'"<>.}	same as `, but on the first non-blank in the line
⬇️	:marks	print the active marks
✔️	🔢 CTRL-O	go to Nth older position in jump list
✔️	🔢 CTRL-I	go to Nth newer position in jump list
⬇️	:ju[mps]	print the jump list

Various motions

Status	Command	Description
✔️	%	find the next brace, bracket, comment, or "#if"/ "#else"/"#endif" in this line and go to its match

Status	Command	Description
✔️	🔢 H	go to the Nth line in the window, on the first non-blank
✔️	M	go to the middle line in the window, on the first non-blank
✔️	🔢 L	go to the Nth line from the bottom, on the first non-blank
⬇️	🔢 go	go to Nth byte in the buffer
⬇️	:[range]go[to][off]	go to [off] byte in the buffer

Using tags

The following are all marked low priority because VSCode has very good support for tags with Goto Symbol. Try it from the command palette if you haven't yet!

Status	Command	Description
⬇️	:ta[g][!] {tag}	jump to tag {tag}
⬇️	:[count]ta[g][!]	jump to [count]'th newer tag in tag list
⬇️	CTRL-]	jump to the tag under cursor, unless changes have been made
⬇️	:ts[elect][!] [tag]	list matching tags and select one to jump to
⬇️	:tj[ump][!] [tag]	jump to tag [tag] or select from list when there are multiple matches

Status	Command	Description
⬇️	:lt[ag][!] [tag]	jump to tag [tag] and add matching tags to the location list
⬇️	:tagsa	print tag list
⬇️	🔢 CTRL-T	jump back from Nth older tag in tag list
⬇️	:[count]po[p][!]	jump back from [count]'th older tag in tag list
⬇️	:[count]tn[ext][!]	jump to [count]'th next matching tag
⬇️	:[count]tp[revious][!]	jump to [count]'th previous matching tag
⬇️	:[count]tr[ewind][!]	jump to [count]'th matching tag
⬇️	:tl[ast][!]	jump to last matching tag
⬇️	:pt[ag] {tag}	open a preview window to show tag {tag}
⬇️	CTRL-W }	like CTRL-] but show tag in preview window
⬇️	:pts[elect]	like ":tselect" but show tag in preview window
⬇️	:ptj[ump]	like ":tjump" but show tag in preview window
⬇️	:pc[lose]	close tag preview window
⬇️	CTRL-W z	close tag preview window`

Scrolling

Status	Command	Description
✔️	🔢 CTRL-E	window N lines downwards (default: 1)
✔️	🔢 CTRL-D	window N lines Downwards (default: 1/2 window)
✔️	🔢 CTRL-F	window N pages Forwards (downwards)
✔️	🔢 CTRL-Y	window N lines upwards (default: 1)
✔️	🔢 CTRL-U	window N lines Upwards (default: 1/2 window)
✔️	🔢 CTRL-B	window N pages Backwards (upwards)
✔️	z CR or zt	redraw, current line at top of window
✔️	z. or zz	redraw, current line at center of window
✔️	z- or zb	redraw, current line at bottom of window

These only work when 'wrap' is off:

Status	Command	Description	Note
✔️ ⭐	🔢 zh	scroll screen N characters to the right	In Code, the cursor wil always move when you run this command, whether the horizontal scrollbar moves or not.
✔️ ⭐	🔢 zl	scroll screen N characters to the left	As above

Status	Command	Description	Note
✔️ ⭐	🔢 zH	scroll screen half a screenwidth to the right	As above
✔️ ⭐	🔢 zL	scroll screen half a screenwidth to the left	As above

Inserting text

Status	Command	Description
✔️	🔢 a	append text after the cursor (N times)
✔️	🔢 A	append text at the end of the line (N times)
✔️	🔢 i	insert text before the cursor (N times) (also: Insert)
✔️	🔢 I	insert text before the first non-blank in the line (N times)
✔️	🔢 gI	insert text in column 1 (N times)
✔️	gi	insert at the end of the last change
✔️	🔢 o	open a new line below the current line, append text (N times)
✔️	🔢 O	open a new line above the current line, append text (N times)

in Visual block mode:

Status	Command	Description
✔️	I	insert the same text in front of all the selected lines
✔️	A	append the same text after all the selected lines

Insert mode keys

leaving Insert mode:

Status	Command	Description
✔️	Esc	end Insert mode, back to Normal mode
✔️	CTRL-C	like Esc, but do not use an abbreviation
✔️	CTRL-O {command}	execute {command} and return to Insert mode

moving around:

Status	Command	Description
✔️	cursor keys	move cursor left/right/up/down
✔️	shift-left/right	one word left/right
✔️	shift-up/down	one screenful backward/forward

Status	Command	Description
✔️	End	cursor after last character in the line
✔️	Home	cursor to first character in the line

Special keys in Insert mode

Status	Command	Description	Note
⬇️	CTRL-V {char}..	insert character literally, or enter decimal byte value	
⚠️	NL or CR or CTRL-M or CTRL-J	begin new line	CTRL-M and CTRL-J are not supported
✔️	CTRL-E	insert the character from below the cursor	
✔️	CTRL-Y	insert the character from above the cursor	
✔️ ⭐	CTRL-A	insert previously inserted text	We apply previously document change made in previous Insert session and we only apply changes that happen under cursor
✔️ ⭐	CTRL-@	insert previously inserted text	As above

Status	Command	Description	Note
		and stop Insert mode	
✔️	CTRL-R {0-9a-z%#:.-="}	insert the contents of a register	
✔️	CTRL-N	insert next match of identifier before the cursor	
✔️	CTRL-P	insert previous match of identifier before the cursor	
⬇️	CTRL-X ...	complete the word before the cursor in various ways	
✔️	BS or CTRL-H	delete the character before the cursor	
✔️	Del	delete the character under the cursor	
✔️	CTRL-W	delete word before the cursor	
✔️	CTRL-U	delete all entered characters in the current line	
✔️	CTRL-T	insert one shiftwidth of indent in front of the current line	

Status	Command	Description	Note
✔️	CTRL-D	delete one shiftwidth of indent in front of the current line	
⬇️	0 CTRL-D	delete all indent in the current line	
⬇️	^ CTRL-D	delete all indent in the current line, restore indent in next line	

Digraphs

Status	Command	Description
✔️	:dig[raphs]	show current list of digraphs
⬇️	:dig[raphs] {char1}{char2} {number} ...	add digraph(s) to the list

Special inserts

Status	Command	Description
⚠️	:r [file]	insert the contents of [file] below the cursor
⚠️	:r! {command}	insert the standard output of {command} below the cursor

Deleting text

Status	Command	Description
✔️	🔢 x	delete N characters under and after the cursor
✔️	🔢 Del	delete N characters under and after the cursor
✔️	🔢 X	delete N characters before the cursor
✔️	🔢 d{motion}	delete the text that is moved over with {motion}
✔️	{visual}d	delete the highlighted text
✔️	🔢 dd	delete N lines
✔️	🔢 D	delete to the end of the line (and N-1 more lines)
✔️	🔢 J	join N-1 lines (delete EOLs)
✔️	{visual}J	join the highlighted lines
✔️	🔢 gJ	like "J", but without inserting spaces
✔️	{visual}gJ	like "{visual}J", but without inserting spaces
✔️	:[range]d [x]	delete [range] lines [into register x]

Copying and moving text

Status	Command	Description
✔️	"{char}	use register {char} for the next delete, yank, or put
✔️	"*	use register `*` to access system clipboard
✔️	:reg	show the contents of all registers
✔️	:reg {arg}	show the contents of registers mentioned in {arg}
✔️	🔢 y{motion}	yank the text moved over with {motion} into a register
✔️	{visual}y	yank the highlighted text into a register
✔️	🔢 yy	yank N lines into a register
✔️	🔢 Y	yank N lines into a register
✔️	🔢 p	put a register after the cursor position (N times)
✔️	🔢 P	put a register before the cursor position (N times)
✔️	🔢]p	like p, but adjust indent to current line
✔️	🔢 [p	like P, but adjust indent to current line
✔️	🔢 gp	like p, but leave cursor after the new text
✔️	🔢 gP	like P, but leave cursor after the new text

Changing text

Status	Command	Description	Note
✔️	🔢 r{char}	replace N characters with {char}	
⬇️	🔢 gr{char}	replace N characters without affecting layout	
✔️ ⭐	🔢 R	enter Replace mode (repeat the entered text N times)	{count} is not supported
⬇️	🔢 gR	enter virtual Replace mode: Like Replace mode but without affecting layout	
✔️	{visual}r{char}	in Visual block, visual, or visual line modes: Replace each char of the selected text with {char}	

(change = delete text and enter Insert mode)

Status	Command	Description
✔️	🔢 c{motion}	change the text that is moved over with {motion}
✔️	{visual}c	change the highlighted text
✔️	🔢 cc	change N lines
✔️	🔢 S	change N lines

Status	Command	Description
✔️	🔢 C	change to the end of the line (and N-1 more lines)
✔️	🔢 s	change N characters
✔️	{visual}c	in Visual block mode: Change each of the selected lines with the entered text
✔️	{visual}C	in Visual block mode: Change each of the selected lines until end-of-line with the entered text
✔️	{visual}~	switch case for highlighted text
✔️	{visual}u	make highlighted text lowercase
✔️	{visual}U	make highlighted text uppercase
✔️	g~{motion}	switch case for the text that is moved over with {motion}
✔️	gu{motion}	make the text that is moved over with {motion} lowercase
✔️	gU{motion}	make the text that is moved over with {motion} uppercase
✔️	{visual}g?	perform rot13 encoding on highlighted text
✔️	g?{motion}	perform rot13 encoding on the text that is moved over with {motion}
✔️	🔢 CTRL-A	add N to the number at or after the cursor
✔️	🔢 CTRL-X	subtract N from the number at or after the cursor
✔️	🔢 <{motion}	move the lines that are moved over with {motion} one shiftwidth left

Status	Command	Description
✔️	🔢 <<	move N lines one shiftwidth left
✔️	🔢 >{motion}	move the lines that are moved over with {motion} one shiftwidth right
✔️	🔢 >>	move N lines one shiftwidth right
✔️	🔢 gq{motion}	format the lines that are moved over with {motion} to 'textwidth' length
⬇️	:[range]ce[nter] [width]	center the lines in [range]
⬇️	:[range]le[ft][indent]	left-align the lines in [range] (with [indent])
⬇️	:[ranee]ri[ght][width]	right-align the lines in [range]

Complex changes

Status	Command	Description	Note
⬇️	🔢 `!{motion}{command}<CR>`	filter the lines that are moved over through {command}	
⬇️	🔢 `!!{command}<CR>`	filter N lines through {command}	
⬇️	`{visual}!{command}<CR>`	filter the highlighted lines through {command}	

Status	Command	Description	Note
⬇️	`:[range]! {command}<CR>`	filter [range] lines through {command}	
✔️	🔢 ={motion}	filter the lines that are moved over through 'equalprg'	
✔️	🔢 ==	filter N lines through 'equalprg'	
✔️	{visual}=	filter the highlighted lines through 'equalprg'	
✔️ ⭐ ⚠️	:[range]s[ubstitute]/{pattern}/{string}/[g][c]	substitute {pattern} by {string} in [range] lines; with [g], replace all occurrences of {pattern}; with [c], confirm each replacement	Currently we only support JavaScript Regex and only options `gi` are implemented
⬇️	:[range]s[ubstitute][g][c]	repeat previous ":s" with new range and options	
⬇️	&	Repeat previous ":s" on current line without options	
⬇️	:[range]ret[ab][!] [tabstop]	set 'tabstop' to new value and adjust white space accordingly	

Visual mode

Status	Command	Description
✔️	v	start highlighting characters or stop highlighting
✔️	V	start highlighting linewise or stop highlighting
✔️	CTRL-V	start highlighting blockwise or stop highlighting
✔️	o	exchange cursor position with start of highlighting
✔️	gv	start highlighting on previous visual area

Text objects (only in Visual mode or after an operator)

Status	Command	Description
✔️	🔢 aw	Select "a word"
✔️	🔢 iw	Select "inner word"
✔️	🔢 aW	Select "a WORD"
✔️	🔢 iW	Select "inner WORD"
✔️	🔢 as	Select "a sentence"
✔️	🔢 is	Select "inner sentence"
✔️	🔢 ap	Select "a paragraph"

Status	Command	Description
✓	🔢 ip	Select "inner paragraph"
✓	🔢 a], a[	select '[' ']' blocks
✓	🔢 i], i[	select inner '[' ']' blocks
✓	🔢 ab, a(, a)	Select "a block" (from "[(" to "])")
✓	🔢 ib, i), i(	Select "inner block" (from "[(" to "])")
✓	🔢 a>, a<	Select "a <> block"
✓	🔢 i>, i<	Select "inner <> block"
✓	🔢 aB, a{, a}	Select "a Block" (from "[{" to "]}")
✓	🔢 iB, i{, i}	Select "inner Block" (from "[{" to "]}")
✓	🔢 at	Select "a tag block" (from <aaa> to </aaa>)
✓	🔢 it	Select "inner tag block" (from <aaa> to </aaa>)
✓	🔢 a'	Select "a single quoted string"
✓	🔢 i'	Select "inner single quoted string"
✓	🔢 a"	Select "a double quoted string"
✓	🔢 i"	Select "inner double quoted string"

Status	Command	Description
✔️	🔢 a`	Select "a backward quoted string"
✔️	🔢 i`	Select "inner backward quoted string"

Repeating commands

Status	Command	Description	Note
✔️ ⭐	🔢 .	repeat last change (with count replaced with N)	Content changes that don't happen under cursor can not be repeated.
✔️	q{a-z}	record typed characters into register {a-z}	
⬇️	q{A-Z}	record typed characters, appended to register {a-z}	
✔️	q	stop recording	
✔️	🔢 @{a-z}	execute the contents of register {a-z} (N times)	
✔️	🔢 @@	repeat previous @{a-z} (N times)	
⬇️	:@{a-z}	execute the contents of register {a-z} as an Ex command	

Status	Command	Description	Note
⬇️	:@@	repeat previous :@{a-z}	
⬇️	:[range]g[lobal]/{pattern}/[cmd]	execute Ex command cmd on the lines within [range] where {pattern} matches	
⬇️	:[range]g[lobal]!/{pattern}/[cmd]	execute Ex command cmd on the lines within [range] where {pattern} does NOT match	
⬇️	:so[urce] {file}	read Ex commands from {file}	
⬇️	:so[urce]! {file}	read Vim commands from {file}	
⬇️	:sl[eep][sec]	don't do anything for [sec] seconds	
⬇️	🔢 gs	goto Sleep for N seconds	

options

Status	Command	Description	Note
⬇️	:se[t]	show all modified options	
⬇️	:se[t] all	show all non-termcap options	
⬇️	:se[t] termcap	show all termcap options	

Status	Command	Description	Note
✔️	:se[t] {option}	set boolean option (switch it on), show string or number option	
✔️	:se[t] no{option}	reset boolean option (switch it off)	
✔️	:se[t] inv{option}	invert boolean option	
✔️	:se[t] {option}={value}	set string/number option to {value}	
✔️	:se[t] {option}+={value}	append {value} to string option, add {value} to number option	
✔️ ⭐	:se[t] {option}-={value}	remove {value} to string option, subtract {value} from number option	We don't support string option here.
✔️	:se[t] {option}?	show value of {option}	
⬇️	:se[t] {option}&	reset {option} to its default value	
⬇️	:setl[ocal]	like ":set" but set the local value for options that have one	
⬇️	:setg[lobal]	like ":set" but set the global value of a local option	
⬇️	:fix[del]	set value of 't_kD' according to value of 't_kb'	
⬇️	:opt[ions]	open a new window to view and set options, grouped by functionality, a one line explanation and links to the help	

Since the list is too long, now we just put those already supported options here.

Status	Command	Default Value	Description
✓	tabstop (ts)	4. we use Code's default value `tabSize` instead of Vim	number of spaces that <Tab> in file uses
✓	hlsearch (hls)	false	When there is a previous search pattern, highlight all its matches.
✓	ignorecase (ic)	true	Ignore case in search patterns.
✓	smartcase (scs)	true	Override the 'ignorecase' option if the search pattern contains upper case characters.
✓	iskeyword (isk)	`@,48-57,_,128-167,224-235`	keywords contain alphanumeric characters and '_'. If there is no user setting for `iskeyword` , we use `editor.wordSeparators` properties.
✓	scroll (scr)	20	Number of lines to scroll with CTRL-U and CTRL-D commands.
✓	expandtab (et)	True. we use Code's default value `insertSpaces` instead of Vim	use spaces when <Tab> is inserted
✓	autoindent	true	Keep indentation when doing `cc` or `S` in normal mode to replace a line.

Undo/Redo commands

Status	Command	Description	Note
✔️	🔢 u	undo last N changes	Current implementation may not cover every case perfectly.
✔️	🔢 CTRL-R	redo last N undone changes	As above.
✔️	U	restore last changed line	

External commands

Status	Command	Description
⬇️	:sh[ell]	start a shell
⬇️	:!{command}	execute {command} with a shell
⬇️	K	lookup keyword under the cursor with 'keywordprg' program (default: "man")

Ex ranges

Status	Command	Description	Note
✔️	,	separates two line numbers	
✔️ ⭐	;	idem, set cursor to the first line number before interpreting the second one	The cursor movement is not included.

Status	Command	Description	Note
✔️	{number}	an absolute line number	
✔️	.	the current line	
✔️	$	the last line in the file	
✔️	%	equal to 1,$ (the entire file)	
✔️	*	equal to '<,'> (visual area)	
✔️	't	position of mark t	
⬇️	/{pattern}[/]	the next line where {pattern} matches	
⬇️	?{pattern}[?]	the previous line where {pattern} matches	
✔️	+[num]	add [num] to the preceding line number (default: 1)	
✔️	-[num]	subtract [num] from the preceding line number (default: 1)	

Editing a file

Status	Command	Description	Note
✔️ ⭐	:e[dit] {file}	Edit {file}.	We will open file in a new Tab of current Grouped Editor instead of opening in current tab.

Multi-window commands

Status	Command	Description	Note
✔️ ⭐	:e[dit] {file}	Edit {file}.	We will open file in a new Tab of current Grouped Editor instead of opening in current tab.
✔️ ⭐	<ctrl-w> hl	Switching between windows.	As we don't have the concept of Window in VS Code, we are mapping these commands to switching between Grouped Editors.
✔️	:sp {file}	Split current window in two.	
✔️ ⭐	:vsp {file}	Split vertically current window in two.	
✔️	<ctrl-w> s	Split current window in two.	
✔️ ⭐	<ctrl-w> v	Split vertically current window in two.	
✔️ ⭐	<ctrl-w> o	Close other editor groups.	
✔️	:new	Create a new window horizontally and start editing an empty file in it.	
✔️ ⭐	:vne[w]	Create a new window vertically and start editing an empty file in it.	

Tabs

Status	Command	Description	Note
✔️	:tabn[ext] 🔢	Go to next tab page or tab page {count}. The first tab page has number one.	
✔️	{count}<C-PageDown>, {count}gt	Same as above	
✔️	:tabp[revious] 🔢	Go to the previous tab page. Wraps around from the first one to the last one.	
✔️	:tabN[ext] 🔢	Same as above	
✔️	{count}<C-PageUp>, {count}gT	Same as above	
✔️	:tabfir[st]	Go to the first tab page.	
✔️	:tabl[ast]	Go to the last tab page.	
✔️	:tabe[dit] {file}	Open a new tab page with an empty window, after the current tab page	
⬇️	:[count]tabe[dit], :[count]tabnew	Same as above	[count] is not supported.
✔️	:tabnew {file}	Open a new tab page with an empty	

Status	Command	Description	Note
		window, after the current tab page	
⬇️	:[count]tab {cmd}	Execute {cmd} and when it opens a new window open a new tab page instead.	
✔️ ⭐	:tabc[lose][!] 🔢	Close current tab page or close tab page {count}.	Code will close tab directly without saving.
✔️ ⭐	:tabo[nly][!]	Close all other tab pages.	`!` is not supported, Code will close tab directly without saving.
✔️	:tabm[ove][n]	Move the current tab page to after tab page N.	
⬇️	:tabs	List the tab pages and the windows they contain.	You can always use Code's built-in shortcut: `cmd/ctrl+p`
⬇️	:tabd[o] {cmd}	Execute {cmd} in each tab page.	

Folding

Fold methods

The folding method can be set with the 'foldmethod' option. This is currently not possible as we are relying on Code's Fold logic.

Fold commands

Pretty much everything fold-related is blocked by this issue.

Status	Command	Description
⬇️	zf{motion} or {Visual}zf	Operator to create a fold.
⬇️	zF	Create a fold for [count] lines. Works like "zf".
⬇️	zd	Delete one fold at the cursor.
⬇️	zD	Delete folds recursively at the cursor.
⬇️	zE	Eliminate all folds in the window.
✔️	zo	Open one fold under the cursor.When a count is given, that many folds deep will be opened.
✔️	zO	Open all folds under the cursor recursively.
✔️	zc	Close one fold under the cursor. When a count is given, that many folds deep are closed.
✔️	zC	Close all folds under the cursor recursively.
✔️	za	When on a closed fold: open it. When on an open fold: close it and set 'foldenable'.
⬇️	zA	When on a closed fold: open it recursively. When on an open fold: close it recursively

Status	Command	Description
		and set 'foldenable'.
⬇️	zv	View cursor line: Open just enough folds to make the line in which the cursor is located not folded.
⬇️	zx	Update folds: Undo manually opened and closed folds: re-apply 'foldlevel', then do "zv": View cursor line.
⬇️	zX	Undo manually opened and closed folds
⬇️	zm	Fold more: Subtract one from 'foldlevel'.
✔️	zM	Close all folds: set 'foldlevel' to 0. 'foldenable' will be set.
⬇️	zr	Reduce folding: Add one to 'foldlevel'.
✔️	zR	Open all folds. This sets 'foldlevel' to highest fold level.
⬇️	zn	Fold none: reset 'foldenable'. All folds will be open.
⬇️	zN	Fold normal: set 'foldenable'. All folds will be as they were before.
⬇️	zi	Invert 'foldenable'.
⬇️	[z	Move to the start of the current open fold.
⬇️	]z	Move to the end of the current open fold.
⬇️	zj	Move downwards to the start of the next fold.

Status	Command	Description
⬇️	zk	Move upwards to the end of the previous fold.

Fold options

Currently we don't support any fold option and we are following Code configurations.