DCD Release Note 27.0
"vi" cheat sheet

RANDOLPH HERBER

June 7, 1993

Conventions

`g`	`command characters`
...	*your arbitrary text*
#	an optional positive integer
move	a movement
re	a "ed"-type regular expression
x	a single character
a	a single lower-case letter
9	a single digit
q	a single lower-case letter or digit
cr	a carriage control character
bs	a backspace (or $\boxed{h}$) character
esc	a escape character
spc	a space character
$\boxed{x}$	a control-*x* character
`@`	a line mode address (See line modes below)

Negative integers can not be specified. Zero is specified by omitting the integer. For most commands, omitting the integer (specifying zero) results in the same action that specifying 1 would have resulted in; exceptions will be noted.

Screen (or visual) mode

Except when indicated, these commands are not ended with a carriage return, **cr**.

This is the initial mode entered when the editor is started as "`vi`" or "`view`". It is automatically reentered after single line mode. It is entered by the `vi` command from multiple (or full-time) line mode.

movement

Note: movements are used by some editing commands to determine their range.

by character

- `#h` **or** #**bs** **or** #$\boxed{h}$ a character to the left; if numbered, move # characters
- `#j` **or** #$\boxed{J}$ down one line to the same column, if possible; if numbered, move # lines
- `#k` up one line to the same column, if possible; if numbered, move # lines
- `#l` **or** `#`**spc** a character to the right; if numbered, move # characters

by text element

- `#b` the beginning of the word to the left, going to next line if needed; if numbered, move # words
- `#B` the beginning of the word to the left, ignoring punctuation, going to next line if needed; if numbered, move # words
- `#e` the end of the current word; if numbered, move # words
- `#E` the beginning of the current word, ignoring punctuation; if numbered, move # words
- `#w` the beginning of the word to the right, going to next line if needed; if numbered, move # words
- `#W` the beginning of the word to the right, ignoring punctuation, going to next line if needed; if numbered, move # words
- `(` beginning of previous sentence
- `)` beginning of next sentence

`{`	beginning of previous paragraph
`}`	beginning of next paragraph
`[[`	beginning of previous section
`]]`	beginning of next section
`%`	when cursor is on a `(`, `{`, `[`, `)`, `}` or `]`, moves the cursor to the matching character

by lines

`0`	first character on current line
`$`	last character on current line
`^`	first visible character on current line
`#-`	up one line and to the first visible character; if numbered, move # lines
`#+` **or** `#`cr	down one line and to the first visible character; if numbered, move # lines
`#|`	to column # of current line; if unnumbered, move to column 1
`#H`	to line # from top of current screen; if unnumbered, move to top of screen
`#M`	line at middle of current screen; number is ignored
`#L`	to line # from bottom of current screen; if unnumbered, move to bottom of screen

by screens

`#`$\boxed{\texttt{f}}$	scroll a screen towards end of file; if numbered, move # screens
`#`$\boxed{\texttt{b}}$	scroll a screen towards beginning of file; if numbered, move # screens
`#`$\boxed{\texttt{d}}$	scroll a half screen towards end of file; if numbered, move # half screens
`#`$\boxed{\texttt{u}}$	scroll a half screen towards beginning of file; i*xxx*f numbered, move # half screens
`#`$\boxed{\texttt{e}}$	scroll a line towards end of file; if numbered, move # lines
`#`$\boxed{\texttt{y}}$	scroll a line towards beginning of file; if numbered, move # lines
`#z`cr	put line with the cursor at top of screen; if numbered, put line # at top of screen
`#z.`	put line with the cursor at middle of screen; if numbered, put line # at middle of screen
`#z-`	put line with the cursor at bottom of screen; if numbered, put line # at bottom of screen

by search

`/...`cr	search forward (wrap permitted) for text
`?...`cr	search backwards (wrap permitted) for text
`n`	repeat previous search in same direction
`N`	repeat previous search in opposite direction
`f`*x*	scan forward in current line for character *x*
`F`*x*	scan backwards in current line for character *x*
`t`*x*	scan forward in current line for character *x*, cursor to left of target
`T`*x*	scan backwards in current line for character *x*, cursor to right of target
`;`	repeat previous scan in same direction
`,`	repeat previous scan in opposite direction

by line number

$\boxed{\texttt{g}}$	display current line number
`m`*a*	mark current character position with letter *a*
`#G`	go to line number #, last line if # is omitted
`‘‘`	go to previous character position in file
`‘`*x*	go to character position marked *x*
`’’`	go to previous line position in file
`’`*x*	go to line position marked *x*

editing

If the command character is duplicated as the movement, the command applies to the entire line the cursor is on.

by insertion

#i...**esc** insert text to left of cursor; if numbered, insert # copies

#a...**esc** insert text to right of cursor; if numbered, insert # copies

#I...**esc** insert text to left of visible text on line; if numbered, insert # copies

#A...**esc** insert text to right of line; if numbered, insert # copies

#O...**esc** insert one or more lines of text above the current line; the number is ignored

#o...**esc** insert one or more lines of text below the current line; the number is ignored

by replacement

#R...**esc** replace character by character until terminated; if numbered, insert # copies for first replacement

#r*x* replace a single character; if numbered, replace the # characters starting the cursor with *x*.

#c*move*...**esc** replace the range until terminated;

#C...**esc** equivalent to #c$...esc; i.e. replace to end of the current line

#s...**esc** replace # characters starting with the cursor with text until terminated; if unnumbered, replace 1 character

#S...**esc** replace # lines starting with the cursor with text until terminated; if unnumbered, replace 1 line

#~ replace a single character with opposite case if alphabetic, otherwise no change; if numbered, replace the # characters starting the cursor

by deletion or copying

With the exception of the x and X commands, the material is placed in a buffer. It is always put in the deletion buffers numbered from 1 to 9, which are a push-down stack from the number 1 buffer end. If a deletion or copy command is prefixed by a buffer designation, "*a*, a copy is also put in the designated buffer.

If the buffer designation for a deletion or copy is an upper-case letter instead of a lower-case letter, then the deleted matter will be appended to the buffer instead of replacing the buffer contents.

When inserting from a buffer, if the material was deleted or copied in line mode, then the insertion is also in line mode, otherwise the insertion is in character mode.

If a insertion from numbered buffer, p or P is undone, u or U, then following undesignated insertions from buffers will use consecutively higher buffers until buffer 9 is reached. This behavior is terminated by a deletion or copy operation. then

#x delete character under cursor; if numbered, repeat # times — deleted material not put in a deletion buffer

#X delete character before cursor; if numbered, repeat # times — deleted material not put in a deletion buffer

#d*move* delete the range

#D equivalent to #d$...**esc**; i.e. delete to end of the current line

#y*move* copy the range

#Y equivalent to #y$...**esc**; i.e. copy to end of the current line

"qP insert buffer before cursor; if not buffer designated, use buffer 1

"qp insert buffer after cursor; if not buffer designated, use buffer 1

#S...**esc** replace # lines starting with the cursor with text until terminated; if unnumbered, replace 1 line

#~ replace a single character with opposite case if alphabetic, otherwise no change; if numbered, replace the # characters starting the cursor

miscellaneous

#. repeat the previous command # times; if unnumbered, do 1 time; some commands are not repeatable u undo last change

U undo all changes to current lines since arriving to it

#J join next # lines to current line; if unnumbered, 1 line

!*move*...**cr** the line range will be passed as stdin to the operating system command ... and the command's stdout will replace the line range

This can be used, for example, to reformat paragraph, sort lines, obtain the current date and time, etc.

#<< for each of the # lines beginning with the current line, if *shiftwidth* columns of whitespace exists at the beginning of the line, then shift the line *shiftwidth* columns to the left.

See table 1 on page 7.

#<< for each of the # lines beginning with the current line, shift the line *shiftwidth* columns to the right.

See table 1 on page 7.

ZZ write file if it has been modified and terminate editor

l refresh the display

Q switch to full-time line mode

: switch to single line mode to run a single line mode command

Single line mode

This mode is entered from visual mode by prefixing the command with the screen mode command to switch to line mode for a single command, :. This character will not be shown in the model commands below.

Each command will be shown with its default value for addresses.

@ a line mode address

A line mode address may be :

absolute line number

. the current line

$ the last line

/*ldots*/ the next line below the current line containing the regular expression, ...

?*ldots*? the next line above the current line containing the regular expression, ...

'*a* the line marked with *a*.

@-## ## lines above addressed line

@+## ## lines below addressed line

general command formats

a zero address commands, e.g. quit

@a single address commands, e.g. read file

@,@a double address commands, e.g. delete lines

@,@a. triple address commands, e.g. copy lines

An address range of % is equivalent to the address pair 1,$.

When a command expects an address range and receives a single address, the address range is that single line.

movement

@**cr** make the addressed line the current line; the default address is ..

editing

.,.d**cr** delete the addressed line range

.,.m@**cr** move the addressed line range after the specified destination address; source range overlap with destination address is not allowed

.,.t@**cr** copy the addressed line range after the specified destination address; source range overlap with destination address *is* allowed

.,.move @**cr** move the addressed line range after the specified destination address; source range overlap with destination address is not allowed

.,.copy @**cr** copy the specified line range after the specified destination address; source range overlap with destination address *is* allowed

., .yank *a***cr** yank the addressed line range into the buffer named *a*.

.put *a***cr** place the contents of the buffer *a* after the addressed line

., .s/*re*/.../**cr** replace the first occurance of regular expression re on each line of the range with ...

Any printable character can be substituted for the delimiter character; in the model, /.

The ending delimiter can be omitted.

A null re is equivalent the previously used regular expression in either a global search or a substitution command.

If ... is "~", the previously used replacement value in a substitution command is used.

\# in the replacement text refers to the values of the
(...
) demarked submatchs in re.

& in the replacement text refers to the value of the entire substring matched by re.

., .s/*re*/.../g**cr** replace the all occurances of regular expression re on each line of the range with ...

Except that the trailing delimiter can not be omitted, the comments above about the single substitution pertain to this form as well.

., .s/*re*/.../c**cr** confirm with y**cr** or deny with **cr** each possible replacement of the first occurance of regular expression re on each line of the range with ...

Except that the trailing delimiter can not be omitted, the comments above about the single substitution pertain to this form as well.

., .s/*re*/.../gc**cr** confirm with y**cr** or deny with **cr** each possible replacement of the all occurances of regular expression re on each line of the range with ...

Except that the trailing delimiter can not be omitted, the comments above about the single substitution pertain to this form as well.

conditional application

1,$g/*re*/*command***cr** apply *command* to all lines in address range matched by re

In effect, the current line is set to each matched line in turn.

command can be:

.,.s/// change first occurance

.,.s///g change all occurances

.,.p print range

.,.d delete range

Any printable character can be substituted for the delimiter character; in the model, /.

1,$v/*re*/*command***cr** like g//... except that the command is applied to all unmatched lines

miscellaneous

In the shell escapes and input-output commands, % is replaced by the current filename and # is replaced by the alternate (previous) filename.

x**cr** write file if modified and terminate

q**cr** terminate

q!**cr** terminate even if file is modified

1,$n**cr** stop editing current file and start next file, if any

If set autowrite is set, the file will be written as well.

1,$n!**cr** stop editing current file unconditionally (even if modified) and start next file, if any

If set autowrite is set, the file will be written as well.

1,$w**cr** write file

1,$w!**cr** write file unconditionally (limited only by operating system limits)

1,$w *filename***cr** write file *filename*

1,$w! *filename***cr** write file *filename* unconditionally (limited only by operating system limits)

1,$w >> *filename***cr** append to file *filename*

`1,$w! >>` *filename*<u>**cr**</u> append to file *filename* unconditionally (limited only by operating system limits)

`.r`<u>**cr**</u> read in current file after addressed line

`.r` *filename*<u>**cr**</u> read file *filename* after addressed line

`.r!` *command*<u>**cr**</u> place stdout of command *command* after addressed line

`1,$e`<u>**cr**</u> restart edit on current file

`1,$e!`<u>**cr**</u> restart edit on current file unconditionally (limited only by operating system limits)

`1,$e` *filename*<u>**cr**</u> restart edit on file *filename*

`1,$e!` *filename*<u>**cr**</u> restart edit on file *filename* unconditionally (limited only by operating system limits)

`sh`<u>**cr**</u> create a subshell environment

`!`*command*<u>**cr**</u> run operating system command

`map` *x sequence*<u>**cr**</u> define character or terminal function *x* as sequence *sequence* of vi commands active only in command mode

`map!` *x sequence*<u>**cr**</u> define character *x* as sequence *sequence* of vi commands active in both command and input modes

`unmap` *x*<u>**cr**</u> undefine character or terminal function *x*

`ab` *abbr phrase*<u>**cr**</u> define "word" *abbr* as an abbreviation for the phrase *phrase*

`unab` *abbr*<u>**cr**</u> undefine "word" *abbr* as an abbreviation

`set` *option*<u>**cr**</u> set *option*

`set` *option=value*<u>**cr**</u> set *option* to *value*

`set no`*option*<u>**cr**</u> unset *option*

`set`<u>**cr**</u> display user set options

`set all`<u>**cr**</u> display all options

`set` *option?*<u>**cr**</u> display value of *option*

See table 1 on page 7.

Multiple line mode

This is the initial mode entered when the editor is started as "ex". This mode is entered from the screen mode by the command character `Q`.

This mode has all the single line mode commands. It also has a few more.

`.i`<u>**cr**</u>`...`<u>**cr**</u>`.`<u>**cr**</u> before the addressed line, insert the lines of new text, ..., terminated by a line with a single period, ., at the beginning of the line immediately followed by a carriage return

`.a`<u>**cr**</u>`...`<u>**cr**</u>`.`<u>**cr**</u> after the addressed line, insert the lines of new text, ..., terminated by a line with a single period, ., at the beginning of the line immediately followed by a carriage return

`.,.c`<u>**cr**</u>`...`<u>**cr**</u>`.`<u>**cr**</u> replace the addressed range with the lines of new text, ..., terminated by a line with a single period, ., at the beginning of the line immediately followed by a carriage return

`vi`<u>**cr**</u> enter screen mode

Expressions

A expression is a sequence of the following elements.

regular expressions

`.` matches any single character except *new line.*

`*` matches zero or more of the preceeding regular expression

`[...]` matches any one of the characters enclosed within the `[` and the `]` characters

A range of characters in the collating sequence can be specified by naming the lowest and highest characters, in that order, separated by a `-`.

The set of character selected can be inverted with respect to the complete collating sequence less the *new line* character by placing a `^` character immediately after the initial `[`.

If a `]` character needs to be specified, it must be specified immediately after the opening `[` or `[^`.

Table 1: “set” options

Option	Abbr.	Default	Description
autoindent	ai	noai	indent to same position as previous line
autoprint	ap	ap	display changes
autowrite	aw	noaw	automatically write file if changed when going to next file
beautify	bf	nobf	ignore non-printing control characters when reading file
directory	dir	/tmp	where vi puts its work files
edcompatible	*none*	noedcompatible	use ed rules in substitute commands
errorbells	eb	errorbells	ring bell on errors
hardtabs	ht	8	spacing of terminal hardware's tabs
ignorecase	ic	noic	ignore case when searching
lisp	*none*	nolisp	uses lisp indent and movement rules
list	*none*	nolist	show tabs as `^I` and end of line as `$`
magic	*none*	magic	allow searchs with regular expressions
number	nu	nonu	display line numbers
open	*none*	open	allow switching between line and screen modes
paragraphs	para	IPLPPPQPLI-pplpipbp	defines paragraph delimiters
readonly	ro	noro	all writes however caused require `!` form
remap	*none*	remap	allow nested map sequences
report	*none*	5	number of lines modified warning limit
scroll	*none*	one half window	scroll amount
shell	*none*	*environ*	shell program to use for escapes
shiftwidth	sw	8	number of columns to shift at a time
showmatch	sm	nosm	move cursor briefly to matching `(`, `{` or `[` when `)`, `}` or `]` is entered
terse	*none*	noterse	use short error messages
wrapscan	ws	ws	search can wrap past end of file
wrapmargin	wm	0	if non-zero, automatically insert carriage return at first white-space after column specifed
writeany	wa	nowa	makes all writes into `!` forms

If a ^ character needs to be specified, it must not be specified immediately after the opening [.

If a - character needs to be specified, it must be specified either immediately after the opening [or [^ or immediately before the closing].

`^` matches the null string at the beginning of the line

`$` matches the null string at the end of the line

`\` removes the special meaning from the following character or adds a special meaning to the following character

If a character does not have a special meaning in either form, the character remains ordinary.

`\(...\)` identifies a sub-regular expression

These sub-regular expressions are numbered from 1 on the left to at most 9 on the right. Their match values can be used in a substitution command replacement string.

`\<` matches a the first character of a "word" or the beginning of the line.

`\>` matches a the first character after a "word" or the end of the line.

any other character represents itself

replacement expressions

`&` has the value of the substring matched by the entire search regular expression

`\#` has the value of the sub-regular expression which corresponds

`~` is a synonym of the previous replacement expression

`\u` makes the next character uppercase if possible

`\l` makes the next character lowercase if possible

`\U` makes all following characters uppercase if possible, up to either the end of the replacement or a `\e` or `\E`

`\L` makes all following characters lowercase if possible, up to either the end of the replacement or a `\e` or `\E`

any other character represents itself

Command Invocation

`vi` Invoke `vi` with an empty buffer.

`vi` *file* Invoke `vi` with the buffer preloaded with a single file.

`vi` *file* ... Invoke `vi` with the buffer preloaded with the first file. Following files can be processed using the `:n` command.

`vi -R` *file* Invoke `vi` in read-only mode with the buffer preloaded with a single file.

`view` *file* Invoke `vi` in read-only mode with the buffer preloaded with a single file.

`vi -r` *file* Invoke `vi` in recovery mode with the name of a file being processed when processing was abnormally ended.

`vi +` *file* Invoke `vi` with the buffer preloaded with a single file and positioned at the end of the file.

`vi +#` *file* Invoke `vi` with the buffer preloaded with a single file and positioned at line # of the file.

`vi +/`*pattern file* Invoke `vi` with the buffer preloaded with a single file and positioned at the first line matching the regular expression, *pattern*.

Miscellaneous

When additional files are edited using the `:e`, `:e!`, `:n` or `:n!` commands, the lettered save buffers are preserved. This can be useful for moving text from file to file.