ISTITUTO NAZIONALE DI FISICA NUCLEARE

Sezione di Bologna

INFN/TC-92/16
15 Maggio 1992

E. Ugolini:

GUIDE TO VI AND EX EDITORS

Servizio Documentazione
dei Laboratori Nazionali di Frascato

Guide to VI and EX editors

E. Ugolini

INFN – Sezione di Bologna

Abstract

The following document is a comprehensive description of vi commands with ex extensions to introduce UNIX visual text editing for generic users and expecially for system administrators who are forced to use vi during system setup and emergency recovery.

1 Introduction

UNIX has a number of editors to process the contents of text files. There are line ditors, such as `ed` and `ex` , which display a line at a time of the file on the screen, and there are screen editors, such as `vi` , `emacs` and `edt`, which display a part of the file on our terminal screen.

The most useful standard text editor is `vi` , that is found on each UNIX system with the same features. With `vi` we can scroll the page, move the cursor, delete lines, insert chars, and more.

The following report is intended as guide to `vi` usage and addresses people with a basic UNIX knowledge. `vi` belongs to the editor class that works by context, i.e. a command mode and a editing mode. Commands apply to specific actions as delete text, while to type text insert/append mode is entered by a command and exited by an escape.

As all UNIX products `vi` is case sensitive. Commands are usually one char and the same char can have different meaning depending on context. One important feature of `vi` is the capability to interact with the shell in any moment and to switch back and forth to `ex` to make use of a more complete editing environment.

The following documentation describes the complete set of `vi` and `ex` commands in tabular form with examples for the most complex situations.

`vi` commands can be composed in a way similar to option specification in UNIX. If a command acts on a char or a line prefixing the command with a number expands the action to the next n elements:

`x`	delete one char
`10x`	delete 10 chars

in the same way different commands when typed in sequence produce a combined action, thus:

`x`	delete one char
`p`	put from local buffer
`xp`	transpose char
`Y`	cut line
`Yp`	duplicate line

2 Starting and Ending a Session

`vi` *filename*	open a file with name *filename* in `vi` mode
`view` *filename*	open a file with name *filename* readonly
`ex` *filename*	open a file with name *filename* in `ex` mode
`ex -R` *filename*	open a file with name *filename* readonly
`vi -r` *filename*	recover editor file after a system crash in `vi` mode
`ex -r` *filename*	recover editor file after a system crash in `ex` mode
	indicate blank lines on screen (not a command)
`ZZ   :x   :wq<ret>`	write contents to file and exit; if file is readonly, command aborts, we can save file with new name (see later)
`:q<ret> :q!<ret>`	quit session without storing text
`:w<ret>`	save text without exiting
`<ctrl>L`	refresh screen

`vi` means visual, `ex` means external (line editor), `w` means write, `q` means quit.
The `vi` calling sequence is:
`vi [ -t tag ] [ +command ] [ -l ] [ -r ] [ -wn ]  name...`
The meaning of the options is:

`-t` *tag*	Specifies a list of tag files. The tag files are preceded by a backslash (\) and are separated by spaces. The tag option should always be the first entry.
`+command`	Tells the editor to begin by executing the specified command. A useful example would be `+/pattern` to search for a pattern.
`-l`	Sets the showmatch and lisp options for editing LISP code.
`-r` *name*	Retrieves the last saved version of the named file in the event of an editor or system crash. If no file is specified, a list of saved files is produced.
`-wn`	Sets the default window size to n. This option is useful for starting in a small window on dialups.

3 Cursor and Display Control

We can move cursor with `arrows` or `n arrow`.

`0` `|`	`n|`	move cursor to col 0 or *n* on current line
`^`		move cursor to first non-blank char on current line
`$`		move cursor to last char on current line
`%`		find matching `({[ ]})`
`h`	`nh`	move cursor left one or *n* chars
`j`	`nj`	move cursor down one or *n* lines
`k`	`nk`	move cursor up one or *n* lines
`l`	`nl`	move cursor right one or *n* chars
`<bar>`	`n<bar>`	same as `l`
`<ret>`	`n<ret>`	move cursor to beg of next or *n* lines
`+`	`n+`	same as `<ret>`
`-`	`n-`	move to first non-blank char in prev or *n* prev lines
`G`		move cursor to first non-blank char in last line of file
`1G`		move cursor to first char in first line of file
`nG`		move cursor to first non-blank char in line *n* of file
`<ctrl>G`	`:f<ret>`	display file informations at bottom screen, for example: `"vi_ex.doc" [Modified] line 29 of 486 --5%--`
`:#<ret>`		display at screen bottom cursor line with line number and prompt for return to continue
`H`	`nH`	move cursor to BOL at top of screen or *n* lines
`L`	`nL`	move cursor to BOL at bottom of screen or *n* lines from bottom
`M`		move cursor to BOL at middle of screen

In the prev list *n* is any positive integer but it cannot exceed the number of lines between the current line and the end or beg of the file or screen if movement is vertical. If *n* exceed the `vi` prediction we hear the beep and the command is ignored.

4 Text Scrolling Commands

`<ctrl>B`	`n<ctrl>B`	scroll backward one or *n* prev screen
`<ctrl>U`		scroll backward one half screen
`n<ctrl>U`		set half screen scroll to *n* lines, then scroll backward one half screen
`<ctrl>D`	`n<ctrl>D`	same as `<ctrl>U` and `n<ctrl>U` but forward
`<ctrl>F`	`n<ctrl>F`	scroll forward to next or nth screen
`<ctrl>Y`	`n<ctrl>Y`	scroll backward one or *n* lines
`<ctrl>E`	`n<ctrl>E`	scroll forward one or *n* lines

The default half screen is 12 lines. Specifying *n* in one of the above commands resets the default value.

5 Positioning Cursor Line in Display Window

To move the line on which the cursor is into a different position (scrolling surrounding text accordingly):

`z<ret>`	moves current line at top of screen
`z.`	moves current line at middle of screen
`z-`	moves current line at bottom of screen

Do not confuse **z** with **H M L** commands, which move the cursor and do not change the location of text lines on screen.

6 Searching for Text

`/regular_expression<ret>`	forward search
`?regular_expression<ret>`	backward search
`n`	repeat search in same direction specified (/ or ?)
`N`	repeat search in opposite direction

If EOF is encountered before the pattern is matched, the search wraps to BOF and continues until the pattern is found or the cursor location is reached (in this case we have a message: Pattern not found).

7 Searching one char in line

Four movement commands are provided for searching forward and backward in the current line for the next or nth occurrence of a given char. They do not search beyond BOL or EOL.

`fc`	`nfc`	forward to next or nth char *c*
`Fc`	`nFc`	backward to next or nth char *c*
`tc`	`ntc`	forward to char before next or nth *c*
`Tc`	`nTc`	backward to char after next or nth *c*
`;`	`n;`	to next char *c* or nth in same direction as prev search
`,`	`n,`	to next char *c* or nth in opposite direction as prev search

8 Word Commands

`w`	`nw`	forward to next or nth BOW or first non-alpha char
`W`	`nW`	forward to next or nth BOW; whitespace only as separator
`e`	`ne`	forward to next or nth EOW or first non-alpha char

E	nE	forward to next or nth EOW; whitespace only as separator
b	nb	backward to next or nth BOW or first non-alpha char

Word commands are not restricted to current line, cursor is wrapped to preceding or following lines in order to meet specified word count.

9 Sentence - Paragraph - Section

Three movement commands enable to skip backward or forward over sentences, paragraphs and sections. A sentence must end with . ? ! followed by two or more spaces. A paragraph is defined as default by a line beginning with:
.IP .LP .PP .QP .P .LI .bp
A section is defined by a line beginning with: .NH .SH .H .HU
Any string at BOL beginning with . can be defined as section or paragraph delimiter by the commands:
:set paragraphs=STRING
:set sections=STRING
where STRING is the user definition without .

)	n)	move cursor forward to next or nth adjacent BOSentence
(	n(	move cursor backward to next or nth adjacent BOSentence
}	n}	move cursor forward to next or nth adjacent BOParagraph
{	n{	move cursor backward to next or nth adjacent BOParagraph
]]	n]]	move cursor forward to next or nth adjacent BOSection
[[	n[[	move cursor backward to next or nth adjacent BOSection

10 Recovering Mistakes or Deleted Text

u	undo last text change not regarding cursor position if the last change was an undo, undo the preceding undo
U	undo all changes made in current line (can be used only once) not allowed if cursor is moved from current line

When change, delete or yank command are executed, the object is copied into a buffer for a possible recover:

p	put buffer contents in text after cursor pos
P	put buffer contents in text before cursor pos

We can use p command to swap chars or lines:

xp	swaps current char with following char
ddp	swaps current line with following line

11 Adding New Text

`i`	insert text before cursor pos
`I`	insert text before first visible char in current line
`a`	append text after cursor pos
`A`	append text to EOL
`o`	open new line after current line
`O`	open new line before current line
`<esc>`	terminate insert or append mode

12 Insert ASCII Control Char in Text

It is possible to insert ASCII Control Chars during insert, append, replace or substitute. Some Control Chars are inserted directly by typing `<ctrl>`*x*:

`<ctrl>G`	bell
`<ctrl>L`	form feed

Be careful: `<ctrl>`*x* can operate directly as editor command (i.e. `<ctrl>[` operates insert break). If we want to insert a visible `<esc>`, we must use: `<ctrl>V` followed by `<esc>`.

13 Deleting Chars, Lines and Words

`x`		`nx`	delete one or *n* chars starting at current cursor pos
`X`		`nX`	delete one or *n* chars starting with the char immediately preceding the current cursor pos
`D`	or	`d$`	delete all chars from current cursor pos to EOL
`d0`	or	`d|`	delete all chars from first left col to char preceding current cursor pos
`dd`		`ndd`	delete current or *n* lines beginning at current line
`dG`			delete all lines starting with current line to EOF
`d1G`			delete all lines starting with current line to BOF
`dnG`			delete all lines starting with current line to line *n* (forward or backward depending on pos of line *n* relative to current line)
`d-`		`nd-`	delete current and preceding line or *n* lines
`d+`		`nd+`	delete current and following line or *n* lines
`dw`		`ndw`	delete from cursor pos to end of current word or *n* words
`db`		`ndb`	delete from nearest preceding beg of word or *n* words to char before current cursor pos

If one or more words beyond the end of current line are deleted, the following line is appended to current line during deletion.

14 Deleting Sentence, Paragraph and Section

`d)`	`nd)`	delete from cursor pos to first or *n* following EOSentence
`d}`	`nd}`	delete from cursor pos to first or *n* following EOParagraph
`d]`	`nd]`	delete from cursor pos to first or *n* following EOSection
`d(`	`nd(`	delete from closest prev or *n* BOSentence to char before cursor
`d{`	`nd{`	delete from closest prev or *n* BOParagraph to char before cursor
`d[`	`nd[`	delete from closest prev or *n* BOSection to char before cursor

15 Deleting to a Text Location in Line or File

`dfc`	`dnfc`	delete text from current pos to first or nth occurrence of *c* on current line toward EOL (including *c*) forward
`dFc`	`dnFc`	delete text from first or nth occurrence of *c* on current line toward BOL to char preceding cursor (including *c*) backward
`dtc`	`dntc`	delete text from current pos to first or nth occurrence of *c* on current line toward EOL (not including *c*) forward
`dTc`	`dnTc`	delete text from char following first or nth occurrence of *c* on current line toward BOL to char preceding cursor backward (not including *c*)
`d/pattern<ret>`		delete text from current pos to first occurrence of text matching `pattern` *forward* to EOF (not including `pattern`). If search wraps to BOF before `pattern` is matched, deletion begins with `pattern` and all text is removed up to, but not including, current cursor pos.
`d?pattern<ret>`		delete text from current pos to first occurrence of text matching `pattern` *backward* to BOF (including `pattern` but excluding cursor pos). If search wraps to EOF before `pattern` is matched, deletion begins with current pos and continue up to, but not including, the matching `pattern` .

16 Replace and Change Text

`~`		swap lowercase < − > uppercase at cursor pos
`:. ! tr '[a-z]' '[A-Z]'`		swap current line to uppercase
`r`		replace one char at cursor pos
`nr`		repeat *n* times the char to be replaced at cursor pos (overstrike)
`R`		replace text until `<esc>` at cursor pos (overstrike)
`nR`		insert *n* times text replaced until `<esc>` at cursor pos
`s`	`ns`	replace one or *n* chars with text until `<esc>` at cursor pos

`S`		delete current line and replace text until `<esc>`
`cc`	`ncc`	change current or *n* lines beginning at current line
`cG`	`c1G`	change all lines starting at current line to EOF or BOF
`cnG`		change all lines starting at current line to line *n* (forward or backward depending on pos of line *n* relative to current line)
`c-`	`nc-`	change current and preceding or *n* lines
`c+`	`nc+`	change current and following or *n* lines
`C`	`c$`	change all chars from current cursor pos to EOL
`c0`	`c|`	change all chars from column 1 to char preceding cursor pos
`cw`	`ncw`	change from cursor pos to end of current or *n* words
`cb`	`ncb`	change from nearest or *n* preceding BOW to char before current cursor pos
`c)`		change from cursor pos to next EOSentence
`c(`		change from preceding SOSentence to char before cursor
`c}`		change from cursor pos to next EOParagrapf
`c{`		change from preceding SOParagrapf to char before cursor
`c]]`		change from cursor pos to next EOSentence
`c[[`		change from preceding SOSection to char before cursor

Changing text such as words, sentences or paragraphs are not restricted to current line. If the number of specified objects exceeds current line contents, the object is extended until the text specification is completely satisfied.

`cfc`	`cnfc`	change from cursor pos to first or nth occurrence of *c* on current line forward to EOL until `<esc>`
`cFc`	`cnFc`	change from cursor pos to first or nth occurrence of *c* on current line backward to BOL until `<esc>`
`ctc`	`cntc`	change from before cursor pos to first or nth occurrence of *c* on current line forward to EOL until `<esc>`
`cTc`	`cnTc`	change from before cursor pos to first or nth occurrence of *c* on current line backward to BOL until `<esc>`

`c/pattern<ret>`	change from cursor pos to first occurrence of `pattern` forward to EOF (not including `pattern`). If search wraps to BOF before `pattern` is matched, change begins with `pattern` and all text is removed up to, but not including, current cursor pos.
`c?pattern<ret>`	change from cursor pos to first occurrence of `pattern` backward to BOF. If `pattern` is matched before BOF is reached, change starts with `pattern` up to, but not including, current cursor pos. If search wraps to EOF before `pattern` is matched, change begins with cursor pos up to, but not including, `pattern` .

Rewiew of change, delete or copy command:

Object	Change	Delete	Copy
1 word	cw	dw	yw
more words, ignoring punctuation	2cW	2dW	2yW
more words back	3cb	3db	3yb
1 line	cc	dd	yy
...to EOL	C | c$	D | d$	y$
...to BOL	c0	d0	y0
single char	r	x	yl

From Cursor to...	Change	Delete	Copy
bottom of screen	cL	dL	yL
next line	c+	d+	y+
next sentence	c)	d)	y)
next paragraph	c}	d}	y}
pattern	c/text	d/text	y/text
EOF	cG	dG	yG
line number...	c13G	d13G	y13G

17 Repeating a Text Change Operation

.	repeat the last operation of text change

It can be used after delete, replace, change, yank/put or any other command that changes or deletes text.

18 Shifting Lines Horizontally Left or Right

`>>`	`n>>`	moves 8 columns right one or *n* lines
`<<`	`n<<`	moves 8 columns left one or *n* lines
`:init,end>`		shift 8 columns right from *init* to *end*
`:12<`		shift 8 columns left 12 lines from current pos

The default value for 'shiftwidth' is 8, but it can be altered using:
`:set shiftwidth=n`
where *n* is the number of columns to shift.

It is also possible to shift a text block using markers (named by one lowercase char from *a* to **z**), first move cursor in desired pos and type:

ma	mark cursor pos with name *a*

second move to a new line pos and type:

`>'a`	right shift of block text named *a*
`<'a`	left shift of block text named *a*

19 Automatic Indenting

`vi` provides an **`autoindent`** option by setting:

`:set ai<ret>`	set autoindent on
`:set noai<ret>`	set autoindent off

`vi` default is noautoindent, a user configuration file .exrc in user's home directory may contain the 'set' command (explained later) to make automatic indenting. The current indent can be changed during insert mode:

`spaces`	`tabs`	increase indent to right on new line
`<ctrl>D`		decrease indent to left on new line by `shiftwidth` chars or to column 1 whichever occurs first
`^ followed by <ctrl>D`		to type a single line at column 1 without changing current indent value

If, for some need, it is necessary to transorm tabs in true whitespaces, we must use the **`expand`** program to do the job (refer to expand(1) manual entry):
```
expand old_file > new_file
```
N.B. expand **IS NOT** a `vi` command.

20 Using Buffers

`vi` can use 36 buffers for copying or moving text:

`unnamed buffer`	as default buffer
`35 named buffers`	form *a* to `z` (lowercase) and from `1` to `9`

When a delete or yank is performed, text is copied into the default buffer (unless a named buffer is specified). The buffer can be placed elsewhere with p or P command. The named buffer is unchanged until a new entry modifies the buffer contents, the default buffer is destroyed even if the change is not a delete or yank but also insert or append. Named buffer maintains its contents during multiple editing, default buffer contents is lost at the end of a file edit. Buffer names are lowercase but must be specified in uppercase for append action.

`"a<yank|delete>`	yank or delete line(s) in buffer *a*
`"A<yank|delete>`	yank or delete line(s) appended in buffer *a*
	only name from *a* to `Z` (uppercase) can be used.
`"ap`	place *a* buffer contents after current char or line
`"aP`	place *a* buffer contents before current char or line

21 Execute a Buffer

When we yank or delete a line containing a `vi` or `ex` command, we can execute the command as:

`@a`		execute buffer *a* as `vi` or `ex` command (with or whithout : in col 1)
`:@a`		execute buffer *a* as `ex` command only (with : in col 1)
`@@`	`:@@`	execute again buffer *a* content

22 Using Markers

`vi` can use 26 markers, from *a* to `z` (lowercase), to locate any position in file:

`ma`	mark cursor pos as marker *a*
`‘a`	move to location marked *a* (N.B. `‘` = `backquote`)
`’a`	move to BOL (first non-blank char) containing marker *a*
`‘‘`	move to last operated marker or toggles with last cursor pos, if no marker is set, cursor moves to BOF
`’’`	move to BOL operated marker or toggles with BOL of last cursor pos, if no marker is set, cursor moves to BOF

If a line or char associated with a marker is deleted, also the marker is canceled.

23 Global/Limited Search/Replace

To perform search or replace we must use `ex` commands by typing:
`:<command><ret>`
The command appears at bottom of screen.
To perform more `ex` commands, it is usefull to switch from `vi` to `ex` mode, to obtain the : prompt:

`Q`	from `vi` to `ex` mode
`vi`	from `ex` to `vi` mode

When in ex mode with Q command, it is possible to perform commands on more lines:

`:%s/[space tab][space tab]*/\<ret>/g`	split whole file in one word lines
`:%s/$/\<ret>/g`	double spacing text

`:g/pattern/p`	search and print `pattern` in whole file
`:g/pattern/p!`	print all lines NOT including `pattern` in whole file
`:init,end g/pattern/p`	search and print from *init* to *end* line in file
`:/pattern1/,/pattern2/p`	search and print from *pattern1* to *pattern2*

g means global, p means print. All found lines are printed on screen and a message appears at the bottom line:
[Hit return to continue]
Cursor moves to last pattern founded.

`:s/old/new`	change only first occurrence of `old` to `new` on current line
`:s/old/~`	repeat previous change on another line
`:&`	repeat previous substitute command
`:s/old/\unew`	change only first occurrence of `old` to `New` on current line
`:s/old/\lNEW`	change only first occurrence of `old` to `nEW` on current line
`:s/old/\Unew`	change only first occurrence of `old` to `NEW` on current line
`:s/old/\LNEW`	change only first occurrence of `old` to `new` on current line
`:s/old/new/g`	change every occurrence of `old` to `new` on current line
`:50,100 s/old/new/g`	change every occurrence of `old` to `new` from line 50 to line 100
`:% s/old/new/g`	change every occurrence of `old` to `new` in whole file
`:% s/old/new/gc`	as previous command but with confirm. It displays entire line where string has been located, string will be marked by a series of `^^^^`. answer `y` to make replacement, `<ret>` for no replacement
`:s`	repeats last substitution
`:g/pattern/s/old/new/g`	search `pattern` in whole file and change `old` to `new` globally on that line
`:%s/[space tab]*$//`	remove blanks and/or tabs at EOL in whole file

Recognized colon command for *init* end/or *end* :

`:$`	last line in file
`:.`	current line
`:%`	abbreviation for 1,$ (whole file)

`:g`	whole file
`:n`	nth line in file
`:.-n`	nth line before current line
`:.+n`	nth line after current line
`:n,m`	from line *n* to line m
`:.-n,.+m`	nth preceding line to mth following line

To visualize non-printing control chars hidden in a file:

`:l`	$ indicates EOL, ^I indicates tab.

To visualize tabs and EOL for whole file:

`:set list`	
`:set nolist`	toggle back to normal mode

24 Editing Multiple Files

`vi` can manipulate more than one file at once:

`\vi one two three`	editing three files in succession

We can know which file is editing by typing:

`:ar`	obtaining at bottom screen (if two is in use): `one [two] three`

ar means args.

We can close editing by using:

`:w`	with a message on bottom screen: `"vi_ex.doc" 670 lines, 27747 chars`
ZZ	with a message on bottom screen: `2 more files to edit`

To proceed to next file:

`:n`	next file
`<shift><ctrl>^`	toggle between files, eventually with `rewind`
`:e#`	reopen previous file

It is possible to preserve files already edited, first type:
`:set autowrite`
next, rewind file pointer to first file typing:

`:rew`	close current file and reopen first file for editing (using :n)
`:rew!`	immediatly reopen first file (without :n)

Named buffers are preserved between files, thus we can copy (using p or P) named buffers contents into a later file in the series.

25 Merging files

We can merge another file or command result after the line including cursor:

`:r filename`	insert *filename* after cursor
`:10r filename`	insert *filename* after line 10
`:g/pattern/r filename`	search `pattern` and insert *filename* after cursor
`:r !UNIX-command`	insert the result of `UNIX-command` after cursor

Cursor moves at beg of inserted file or command.

26 Write Command

`:w`	save current file during editing
`:w \new`	save current file with name `new` , only if it does not exist
`:w! \verb!file`	save current file overwriting `file`, N.B. no space before !
`:10,20 w \new`	save from line 10 to 20 with name `new` , only if not existing
`:10,20 w!`	save from line 10 to 20 overwriting current file
`:10,20 w! \new`	save from line 10 to 20 overwriting `new`
`:'a,'b w \new`	save from marker *a* to b with name `new` ,only if not existing
`:.,/pattern/w`	save from current pos to `pattern` overwriting current file
`:10,20 w >> \new`	save from line 10 to 20 appending to file `new`
`:w %nnn`	save current file with name `filenamennn`
`:w /a/b/c/\new`	save current file in path `/a/b/c` with name `new`

27 Escaping to UNIX Shell

`:!command`	execute command and prompts: `Hit return to continue`
`:w !command`	as previous, N.B. a space before !
`:!`	repeath most recent shell escape
`!!`	spawn a new Bourne shell from `vi` , to retrive editor: `<ctrl>D`
`:!csh`	spawn a new C shell, to retrive editor: `<ctrl>D`
	Prompts: Hit return to continue
`:!ksh`	spawn a new Korn shell, to retrive editor: `<ctrl>D`
	Prompts: Hit return to continue
`:init,end!sort`	provide sorting from *init* to *end* lines
`:init,end!fmt`	provide a simple formatting, from *init* to *end* line format to 72 chars per line

N.B. the following chars are significant for `vi` and csh:
`! & | % + - * ? / ^ < > ( ) && || << >> # ; $`
Which of these chars is interpreted as special char depends on context in which it is used. In any case, preceding the char with \ cancels its interpretation as special char.

28 Tag Files for Multiple Programs

`vi` includes tag file capability that, when used with the `ctags` (see `man ctags` for more), simplifies random editing of code segments in large programs. Functionality is provided only for Fortran, Pascal and C code.

`ctags prog*`	creates the file `tags` for all `prog*` source code

Example: we have 4 files: main.f one.f two.f three.f
in file three.f we have also a subroutine: four.f
running: `ctags *.f`
we obtain the file: `tags`
containing the following lines:

```
Mmain   main.f  /^   program main$/
four    three.f /^   subroutine four$/
one     one.f   /^   subroutine one$/
three   three.f /^   subroutine three$/
two     two.f   /^   subroutine two$/
```

If we want to edit subroutine four, there are some options:

`\vi -t four`	`vi` edits directly file three.f positioning cursor on line containing `subroutine four`
`:ta Mmain\`	from `vi` to recall `main`
`:ta two`	from `vi` to recall subroutine `two`
`<ctrl>]`	from `vi` , positioning cursor on init of subroutine name

It is possible to edit files in mixed programming language, creating a `tags` file as follows:

`ctags *.[cfp]`	create a single `tags` file fron code written in C, Fortran and Pascal

N.B. it is safer to use always `autowrite`

29 Abbreviations as Typing Aids

It is usefull to use abbreviations instead of long or difficult text:

`:ab word text`	adds `word` to current list of abbreviations, `word` is the abbreviated form for `text`. When `vi` is in append/insert mode, if `word` is typed (as a complete word with blanks before and after), editor expands the abbreviation. Defined abbreviations are discarded at session end. Permanent abbreviations can be entered in file `.exrc` (see later)
`:una word`	delete `word` from list of abbreviations

`ab` means abbreviate, `una` means unabbreviate.
For example: `:ab crt cathode ray tube`
when in insert we type `crt`, editor expand to `cathode ray tube`

30 Append or Change Line of Text

Append/change operates only in `ex` mode, from `vi` type `Q` to change mode.

`:a! \>`	adds text after current line until . in col 1 is typed
`:a!`	as previous but toggles autoindent (upon append termination autoindent reverts its normal state)
`:.+12a`	adds text after 12th line following current pos
`:init,end c`	first and last line to change until . in col 1 is typed
`:init c n`	chang *n* lines from *init*
`:c n`	change from current to nth line
`:.+2c13`	13 lines are replaced starting at second line after current
`:c!`	change current line but toggles autoindent

a means append, *c* means change.

31 Insert text

Insert operates only in `ex` mode, from `vi` type `Q` to change mode.

`:i`	insert text after current line until . in col 1 is typed
`:i!`	as previous but toggles autoindent (upon append termination autoindent reverts its normal state)
`:10 i`	insert text after line 10

`i` means insert.

32 Join Lines on Single Line

`:J`	current and next line are combined

`:J4`	combine 4 lines from current
`:init,end J`	combine from *init* to *end* line as single line
`:10J3`	combine 3 lines from line 10
`:J!`	current and next line are combined with no change in whitespace

`J` means join.

33 Yank Text for Copy Operation

Yank command copies specified lines into buffer for farther use:

`yy`	copy current line into unnamed buffer
`nyy`	copy *n* lines into unnamed buffer
`:init,end y`	copy, into unnamed buffer, lines fron *init* to *end*
`:init y n`	copy *n* lines, into unnamed buffer, from *init*
`:y n`	copy *n* lines from current line, into unnamed buffer
`:y a n`	copy *n* lines from current line into buffer *a*

34 Map a Macro to a Key

We can define macros and associate them with a keybord key:

`:map key macro`	defines `macro` and associates with `key` in command mode only
`:unm key`	undefines `key`
`:map! key`	macro defines `macro` and associates with `key` also in insert/append
`:unm! key`	undefines `key`
`:map`	shows defined keys in command mode only
`:map!`	shows defined keys also in insert/append

For example:

`:map ^A dw`	defines `<ctrl>A` as delete word
`:map ^A /pattern/^Mdw`	defines `<ctrl>A` as search `pattern` and delete word (note `<ctrl>M` obtained with `<ctrl>V` and `<ret>` to complete command search)

N.B. be carefull: map key *a* redefines append command.

35 Move or Copy Lines to a New Location

`:init,end m dest`	delete lines from *init* to *end* and copy after `dest`
`:n m dest`	delete nth line and copy after `dest'`
`:init,end co dest`	copy lines from *init* to *end* after `dest`

`:n co dest`	copy nth line after **dest**
`:%co$`	copy entire file after last line

Copy command accepts the following flags, only using **ex**:

`#`	print current line with line number after copy
`p`	print current line without line number after copy (default)

m means move, co means copy.

For example:

`:.,+5m10`	delete 5 lines from current pos and put them after line 10
`:.+2c10`	copy second line after current after line 10

36 Delete Lines

`:init,end d`	delete lines from *init* to *end*
`:init d n`	delete lines from *init* for *n* lines
`:d n`	delete *n* lines from current pos

For example:

`:d u 10`	delete 10 lines from current and put them in buffer u
`:/text/+2,dR3`	delete three lines starting at second line after line containing **text** and append to buffer **r**
`:'a+5,$-4d`	delete from 5th line after the line that contains marker *a* through the 4th line before EOF

d means delete.

37 Edit Different File

`:e filename`	terminates current editing and start new session for **filename** If **autowrite** is not set and current file has been modified but not written, command aborts with a message; if **autowrite** is set and current file has been modified, current file is written before new file is loaded.
`:e! filename`	terminates current editing, modified or not, and **filename** is loaded.
`:e!`	editor reload current file
`:e+n filename`	same as first command, but editor begins at line *n*
`:e+/pattern`	same as previous, but starting from line containing **pattern** N.B. **pattern** must contains no spaces or tabs

`e` means edit.

38 Print Lines Numbers

`:init,end nu`	print lines with numeration from *init* to *end* line
`:init nu n`	print lines with numeration from *init* to nth line
`:#n`	print lines with numeration from current pos to nth line
`:.=`	print only line number of current pos
`:=`	print only last line number in file

`nu` means number.

39 Restore Yanked/Deleted Line Back in File

`:pu`	restores last deleted/yanked lines after current pos
`:pu a`	restores buffer *a* after current pos
`:50 pu a`	restores buffer *a* after line 50

`pu` means put.

40 Set/List Editor Option

Set command sets or lists current editor configuration parameters:

`:se param`	sets `param` to a specified value
`:se param?`	lists current setting of `param`
`:se all`	lists all editor options
`:se`	lists only options changed from default

41 Input `ex` Command from File

To execute source command from `vi` but in `ex` mode, type `Q` from `vi`:

`:so filename`	editor reads and executes `ex` commands from `filename` commands in `filename` can be nested

`so` means source

42 Undo Changes

Undo command restores all changes made by most recent editing command to their original form:

`:u`	restores all changes

43 Editor Version Number

`:ve`	prints editor current version, for example: `Version 3.7, 18-Oct-85`

44 Configuring vi/ex Editor

We have three ways to configure editor automatically:

- define non-default values using the environment variable EXINIT
- create configuration in file `.exrc`
- embed `ex` commands in first and/or last five lines of current file edited

When `vi/ex` starts, the editor searches for environment variable $EXINIT and uses its contents as configuration command if it exists,if not the editor searchs for file `.exrc` in home and/or in current directory, if neither EXINIT nor `.exrc` exist, default values are used. After completing the above tasks, the editor opens the file to edit, and then, if `modelines` option is set, it scans the first and last five lines in file to determine whether any `ex` commands have been placed there,if so, the commands are executed before editing control is passed to user. Warning: the edit commands must be deleted to use file outside editor.

`:set all`	show all default and changed options
`:set option`	enable option
`:set nooption`	disable option
`:set option=value`	assign a value to option

Typical default options are:

```
noautoindent    nonumber                          noslowopen
autoprint       open                              nosourceany
noautowrite     nooptimize                        tabstop=8
nobeautify      paragraphs=IPLPPPQPP LIpplpipbp   taglength=0
directory=/tmp  prompt                            tags=tags /usr/lib/tags
noedcompatible  noreadonly                        term=vt300
```

```
noerrorbells      redraw                      noterse
hardtabs=8        remap                       timeout
noignorecase      report=5                    ttytype=vt300
nolisp            scroll=11                   warn
nolist            sections=NHSHH HUnhsh       window=23
magic             shell=/bin/sh               wrapscan
mesg              shiftwidth=8                wrapmargin=0
nomodeline        noshowmatch                 nowriteany
```

Example of EXINIT variable in `.login` file:
`setenv EXINIT 'set redraw wm=8'`

45 Option Descriptions

Each option is recognized by `vi` or `ex` (or both) as indicated and some can be abbreviated.

```
1) autoindent \verb!(vi/ex)!
   abbr: ai     default: noai
```

To change autoindent on new line, space over to desired column to increase indent. To decrease indent to previous `shiftwidth` column, use `<ctrl>D` as first char in the line. To input a single line with no indent and return to previous indent, use ^ followed by `<ctrl>D` at beg of unindented line. If a new line starts with one or more tabs or spaces, next following line is started at the new indent.

```
2) autoprint \verb!(ex)!
   abbr: ap     default: ap
```

Option prints current line after the commands:
`copy, delete, join, l, move, shift, substitute, undo`
essentially it is the same as adding p at the end of each of above commands.

```
3) autowrite \verb!(vi/ex)!
   abbr: aw     default: noaw
```

Buffer contents is written to current file if `vi` or `ex` encounters:
`rewind, tag, !` (shell escape)
In `vi` ^ (switch files) or ^| (tag goto) trigger autowrite. If we want to bypass autowrite, we can use ! (forced command). To prevent autowrite:

`quit!`	instead of `quit`
`edit`	instead of `next`
`rewind!`	instead of `rewind`
`stop!`	followed by the `tag!` command instead of `tag`
`shell`	instead of !

from `vi`:
`:e#` (switching between two files)
`:ta!` (using tag files to find text segments)

```
4) beautify \verb!(vi/ex)!
   abbr: bf     default: nobf
```

Allows to eliminate all control chars except tab, newline and formfeed when we enter text in insert or append mode.

```
5) directory \verb!(vi/ex)!
   abbr: dir     default: dir=/tmp
```

Specifies which directory is to be used by the editor when we are creating the buffer file following an edit file command from within the editor. Option does not affect the buffer location if the option is set during the session. This command is expecially needed when reaching disk limit (file system full).

```
6) edcompatible \verb!(vi/ex)!
   abbr: ed      default: noed
```

With option enabled, if g (global) or *c* (check) flags are present in a substitute command, the flags are toggled and the command is processed accordingly.

```
7) errorbells \verb!(vi/ex)!
   abbr: eb      default: noeb
```

Used only on terminals that do not support inverse video to print messages.

```
8) hardtabs \verb!(vi/ex)!
   abbr: ht      default: ht=8
```

Defines spacing between hardware tab setting and number of spaces used by system when expanding tab chars.

```
9) ignorecase \verb!(vi/ex)!
   abbr: ic      default: noic
```

Matching regular expressions, command maps all uppercase chars in text to lowercase.

```
10) lisp \verb!(vi/ex)!
   abbr: none    default: nolisp
```

Special autoindent for Lisp source.

```
11) list \verb!(vi/ex)!
   abbr: none    default: nolist
```

Shows tabs and newline.

```
12) magic \verb!(vi/ex)!
   abbr: none    default: magic
```

Setting nomagic reduces the number of regular expression metachars to only ^ and $. To reenable metachars while in nomagic, precede them with \.

```
13) mesg \verb!(vi/ex)!
   abbr: none    default: mesg
```

Enables other users to send messages to terminal.

```
14) modelines \verb!(vi/ex)!
   abbr: modeline       default: nomodeline
```

With modeline editor scans first and last five lines in file looking for `ex` commands. Modelines must appear in a single line as:
`ex: set <option>:`
To separate multiple command on a single line, use `|`.

```
15) number \verb!(vi/ex)!
   abbr: nu     default: nonu
```

Displays lines with line number.

```
16) open \verb!(ex)!
   abbr: op     default: open
```

Allows entry to `vi` mode from `ex`

```
17) optimize \verb!(vi/ex)!
    abbr: opt   default: noopt
```

Suppresses automatic CR by the terminal when direct cursor addressing is not supported.

```
18) paragraphs \verb!(vi/ex)!
    abbr: para  default: paragraphs=IPLPPPQPP LIpplpipbp
```

Specifies the one/two-char macro name to be used by `nroff`.

```
  19) prompt \verb!(ex)!
      abbr: none  default: prompt
```

Editor prompts for new command when in command mode by printing `:`.

```
  20) readonly \verb!(vi/ex)!
      abbr: ro    default: noreadonly
```

Set readonly attribute to editing file.

```
  21) redraw \verb!(vi/ex)!
      abbr: none  default: redraw
```

Option simulates intelligent on dumb terminal. Editor prints new chars on current line to the right of cursor and reprints lines as needed when inserting, deleting or changing visible chars on display.

```
  22) remap \verb!(vi/ex)!
      abbr: none  default: remap
```

Links a macro directly to last macro found in series. For examples if *a* is mapped to `b` and `b` to *c* , remap will map *a* to *c* .

```
  23) report \verb!(vi/ex)!
      abbr: none  default: report=5
```

Sets a threshold of change (number of lines). Editor will notify when this threshold is exceeded.

```
  24) scroll \verb!(vi/ex)!
      abbr: none  default: scroll=11
```

Sets number of lines scrolled when editor receives a `<ctrl>D`.

```
  25) sections \verb!(vi/ex)!
      abbr: sect  default: sect=NHSHH HUnhsh
```

Specifies the one/two-char macro name to be used by `nroff`.

```
  26) shell \verb!(vi/ex)!
      abbr: sh    default: sh=/bin/sh
```

Defines path and filename of user Shell environment variable.

```
27) shiftwidth \verb!(vi/ex)!
    abbr: sw    default: sw=8
```

Sets spacing between tab stops. Use shiftwidth to reverse tabbing with `<ctrl>D`, when using autoindent while appending text, and when using << and >> commands.

```
28) showmatch \verb!(vi/ex)!
    abbr: sm    default: nosm
```

In editor open mode, cursor moves to matching (or { for one second when closing) or } is typed and then returns to closing char.

```
29) slowopen \verb!(vi/ex)!
    abbr: slow  default: noslow
```

Only for slow terminal.

```
30) tabstop \verb!(vi/ex)!
    abbr: ts    default: tabstop=8
```

Defines tab spacing used when editor expands tabs.

```
31) taglength \verb!(vi/ex)!
    abbr: tl    default: tl=0
```

Defines max number of chars considered significant in a tag. Setting to 0 makes all chars significant.

```
32) tags \verb!(vi/ex)!
    abbr: none  default: tags=tags /usr/lib/tags
```

Defines path and filename to be used as tag files for tag command or `-t` option when editor starts.

```
33) term \verb!(vi/ex)!
    abbr: none  default: term=vt300
```

Defines terminal type.

```
34) terse \verb!(vi/ex)!
    abbr: none  default: noterse
```

Types shorter error diagnostics.

```
35) timeout \verb!(vi/ex)!
    abbr: none  default: timeout
```

If set, the timeout function is enabled, meaning that if an escape char is not followed within the time limit by another char, the escape is treated as a separate char rather than as part of a two-char sequence.

```
36) ttytype \verb!(vi/ex)!
    abbr: none  default: ttytype=vt300
```

Defines ttytype for terminal in use with editor.

```
37) warn \verb!(vi/ex)!
    abbr: none  default: warn
```

Editor send a message if no `'no write since last change'` message appears before a ! or shell command.

```
38) window \verb!(vi/ex)!
    abbr: none  default: window=23
```

Specifies number of lines displayed in a text window.

```
39) wrapscan \verb!(vi)!
    abbr: ws    default: wrapscan
```

Pattern searches resulting from a `/?nN` command automatically wrap around to opposite EOF and continue whenever BOF or EOF is reached.

```
40) wrapmargin \verb!(vi/ex)!
    abbr: wm    default: wrapmargin=0
```

Num of chars for automatic wrapping.

```
41) writeany
    abbr: wa    default: nowriteany
```

Option inhibits checks before write command, so we can write to any file that system's protection will allow.

```
42) nosourceany (not documented)
```

46 Regular Expressions

Regular expressions are a simple pattern matching language used for locating text in a file. All regular expressions are constructed from series of one or more single char expressions. Single char expressions can take several forms:

`Typing chars`	`A-Z a-z 0-9 ! @ # % < > ( ) { } , ~ | : ; ? + = - _ <tab> <blk> <ctrl chars>`	Any alphanumeric or symbol char that can be typed except chars used in substitution. These chars match only identical chars in text. We must precede ? with \ if ? is used as first char in a backward search.
`Substitution or search control chars`	`. ^ $ / [ ] \ * -`	These chars represent another char or beg or end of line or serve as delimiters, range identifiers, or ascape chars in regular expressions. However, under certain conditions, - and] are interpreted directly as explained below.
`Sets or ranges of chars`	`[set_of_chars] or [range_of_chars] or [combination_of_both]`	A group of single chars or range of chars enclosed within a pair of `[]` (such as `[actz58&]` or `[3-7]`) where a match is accepted if any of the chars between the `[]` or in the specified range appears in the position defined by the position of the single char expression in a larger expression. The second form example accepts a match if 3, 4, 5, 6 or 7 appears in the position indicated.

The - is interpreted as a range specifier when defining sets of chars, as in the single char expression `[a-z]`, unless it is the first char in a set of chars, as in the expression `[-abdfgh12]`, which match any one of the chars -, *a* , `b`, `d`, `f`, `h`, `1`, or 2. Likewise, the `]` terminates the expression unless it is the first char in the set, as in the group `[]=+rt12]`, which matches any one of the chars `]`, `=`, `+`, `r`, `t`, `1` or 2.

47 BOL and EOL in Regular Expression

`^expression`	searches for `expression` at BOL
`expression$`	searches for `expression` at EOL

48 Arbitrary Chars

When we search words that differ for one char only (i.e. `these` and `those`), we can use `.` instead of char that differs:
`/th.se`
unfortunally we find also: `th se` or `thxse`
another way is:
`/th[oe]se`
To find a word regardless of its position in a line:
`/\<word\>`
this matches `^word`, `word$` and `word` elsewhere in the line.

`/^.*[0-9]`	represents an arbitrary number (zero or more) of arbitrary chars lying between BOL and last occurrence in line of any numbers lying in the range `0` to `9`.

Brackets define ranges of chars and char sets to match a given char pos.
Examples:

`/\[0-9\]`	find `[0-9]`
`/\[[a-z][a-z][a-z]\]`	find a word made of only 3 chars, alphabetic and lowercase, enclosed between `[]`
`/th[aeo][tyue]`	find words as: that, they, thou, thee but not them

We can exclude from search some chars:

`/[^aslm]`	match accepts any char axcept `aslm`

49 Metachars Summary

`.`	matches any single char except NL (spaces are also chars)
`*`	matches any number (including 0) of the single char that immediately precedes it (`.*` means match any number of any char)
`[ ]`	matches any one of the chars enclosed between brackets
`[^ ]`	matches any one char not in list
`\{n,m\}`	matches a range of occurrences of the single char that immediately precedes it, *n* and `m` are integers from 0 to 256 that specify how many occurrences to match: `\{n\}` matchs exactly *n* occurrences `\{n,\}` matchs at least *n* occurrences `\{n,m\}` matchs any number of occurrences from *n* to `m` for examples: `A\{2,3\}` matchs either `AA` or `AAA` but not `A`
`^`	requires that the following regular expression be found at BOL
`$`	requires that the preceding regular expression be found at EOL
`\`	the following special char is an ordinary char, for examples: `\.` stands for a dot, `\*` for an asterisk

`\( \)`	saves pattern enclosed between `\(` and `\)` into special holding space, up to 9 patterns can be saved on single line. They can be replayed in substitutions by the sequences `\1` to `\9`, for examples: `:%s/old\([ ,.;:!?]\)/new\1/g` replaces in whole file `old` followed by either ` ,.;:!?` additionally, the char that is matched is saved using `\(` and `\)` and restored on the right side with `\1` the same task is performed by: `:%s/\<old\>/new/g` `\<old\>` will find all instances of the word `old` , whether followed by punctuation or space
`\< \>`	matches chars at beg (`\<`) or at end (`\>`) of a word, for examples: `\<ac` matches only words which begin with `ac`, such as `action` but not `react`
`~`	matches whatever regular expression was used in last search, for examples: if we search for `The`, we could search for `Then` with `/~n`. We can use this pattern only in a regular search (with `/`)

A APPENDIX: Quick reference.

Movement commands

Char

`h j k l`	← ↑ ↓ →

Text

`w W b B`	forward, backward by word
`e E`	EOW

Lines

`0 $`	first, last pos in current line
`^`	first char in current line (ignore space)
`+ -`	first char of next, prev line
`n|`	col *n* of current line
`H L`	top, last line of screen

Screen

`<ctrl>F <ctrl>B`	scroll forward, backward one screen
`<ctrl>D <ctrl>U`	scroll down, up half screen
`<ctrl>L`	refresh screen

Search

`/text`	search forward for text
`/^text`	saerch text at BOL
`/text$`	saerch text at EOL
`?text`	search backward for text

`n N`	repeat last search same, opposite direction
Line number	
`<ctrl>G`	display current line number and file name
`nG`	move to line number *n*
`1G G`	BOF, EOF
`''`	return to position before G command
`:n`	move to line *n*
Insert	
`i a`	insert text before, after cursor
`I A`	insert text at BOL, EOL
`o O`	open new line below, above cursor pos
Change	
`r R`	overstrike one char, line
`cw`	change word
`cc C`	change line, to EOL
`s S`	substitute char, line
Delete, Move	
`x X nx`	delete next, prev char or next *n*
`dw`	delete word
`db`	delete word backward
`dd D`	delete line, to EOL
`p P`	put delete or yanked text after, before cursor
Yank	
`yy nyy`	copy current or *n* lines to internal buffer
`yw ynw`	copy current or *n* words to internal buffer
`YP`	duplicate line
Exit commands	
`ZZ`	exit and save
`:w :wq`	write, write and quit
`:q!`	forced quit
`:n,mw file`	write n:m lines to file
Input Commands	
`:r file`	read file
`:r !cmd`	insert shell command output
Marker	
`ma`	mark line as *a*
`'a`	go to line marked *a*
Cut $ Paste	
`"add "andd`	cut line or *n* lines to buffer *a*
`"ap "aP`	paste after, before
`J nJ`	join current line with next one or *n* next lines
`xp`	transpose current and next char
Miscellaneous	
`.`	repeat last edit action
`%`	show matching `([{ }])`

`<esc>`	stop insert
`u`	undo previous change
`~`	change case of current char
`:sh`	escape to the shell (return to `vi` with `<ctrl>D`)
`:!`	execute a shell command
Environment	
`:set all`	show environment
`:se ic`	ignore case
`:se nu`	print line number
`:se nonu`	disable line number

B APPENDIX: Abbreviations used in the report

`BOL`	beginning of line
`EOL`	end of line
`EOW`	end of word
`BOW`	beginning of word
`BOF`	beginning of file
`EOF`	end of file
`beg`	beginning
`prev`	previous
`pos`	position
`EOSentence`	end of sentence
`EOParagraph`	end of paragraph
`EOSection`	end of section
`SOSentence`	start of sentence
`SOParagraph`	start of paragraph
`SOSection`	start of section

Contents