CONVEX

vi Quick Reference

Order No. DSW-019

First Edition, Rev. 1
February 1991

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX
vi Quick Reference

Order No. DSW-019
First Edition, Rev. 1
February 1991

Printed in the United States of America.

Acknowledgments

CONVEX Computer Corporation expresses sincere appreciation to the Purdue University Computing Center for their contribution to this reference.

Contents

Preface

How to use this reference

In this reference you'll find `vi` and `ex` commands divided into sections according to their functions. You'll also find a section on using regular expressions and on setting editing options.

To find the task you want to perform, look over its corresponding section for the command syntax and description.

How to use command descriptions

If a command description instructs to *enter* a command, you need to press the **RETURN** key after typing the command. If a description instucts you to *type* a command, the command will execute without pressing the **RETURN** key.

Different fonts indicate how to use different parts of the commands.

`courier bold`	Type exactly as shown.
italic	Substitute with the appropriate item or value.
KEYCAP	Press the indicated key.

The **CTRL** key (control) should be held down while pressing another key associated with it in the command syntax description. For example, in the command syntax description:

CTRL-d

you need to press the **CTRL** key and type `d` simultaneously.

Introduction

What is `vi`?

`vi` is the visual mode of the UNIX `ex` text editor. These are referred to as `vi` and `ex`, respectively.

When using `vi`, you can move the cursor around the screen to edit text. You can also execute `ex` commands while in `vi`.

Definitions: command mode and insert mode

`vi` has two modes:

command mode	Key strokes execute commands instead of typing the usual keyboard letters and symbols.
insert mode	Your keyboard acts as a typewriter so that you can enter text.

Repeating `vi` commands

Repeat most motion, deletion, and substitution commands by entering an integer before the command. A few examples are:

`5x`	Delete character at cursor and four characters right of cursor, if they exist.
`3b`	Move left three alphanumeric words.
`5(`	Go backwards through the file five sentences.
`10r`*x*	Replace ten characters, with a single character, *x*, starting at the cursor and moving right.

Combining `vi` commands

Most `vi` commands can be combined with others to affect a desired change over a desired range. Do this by using commands with targets, as the following sections describe.

What are command targets?

target Unit of cursor motion, such as a character, a word, a line, and end of file.

All sections in this reference that contain targets are:

- ❑ "Moving cursor by character or line" on page 6
- ❑ "Moving cursor by word" on page 7
- ❑ "Moving through a file" on page 8
- ❑ "Searching for characters" on page 11
- ❑ "Positioning the cursor on the screen" on page 12

Commands that work with targets

The five `vi` commands that can be combined with targets are:

- ❑ `c` (change)
- ❑ `d` (delete)
- ❑ `y` (yank)
- ❑ `>` (right shift)
- ❑ `<` (left shift)

All of these commands can be found in sections "Replacing and changing text" on page 9, "Deleting text" on page 9, "'Yanking and putting' text" on page 10, and "Shifting text" on page 10, respectively.

The five commands listed in this section on page 2 operate on a specified target. For example, here are some combinations of commands and targets, and what they do:

`dG`	Delete current line through the end of file.
`dB`	Delete one blank-delimited word preceding the cursor.
`cw` *string*	Change one alphanumeric word into string, starting at the cursor.
`y)`	Yank the current sentence from the version of the file on which you are now working and put into an unnamed buffer. (You will not notice anything happening except, possibly, for a message at the bottom of your screen telling you how many lines have been yanked.) Put the yanked text anywhere you want using the `P` or `p` command. (Refer to the section "'Yanking and putting' text" on page 10.)

The possibilities of tailoring commands to suit your needs are quite varied. The best way to master command combinations is to try out any you can imagine. The `u` command will undo most unwanted or unexpected changes. (See "Undoing and repeating changes," page 9.)

Using `vi` commands

Entering a `vi` session

To use `vi` to create or modify a file, enter

`vi` *option filename*

where *filename* is the name of the file you want to edit, and *option* is none, or one or more of the following:

`-t` *tag*	Edit the file containing *tag* and position the editor at its definition, where *tag* is a C or Fortran object name. (Refer to the ctags(1) or ftags(1) man pages.)
`-r`	Recover file modifications after a system crash.
`-R`	Set file permission to read only. You can view the file, but cannot modify it.
+*command*	Editor begins execution by processing the specified editing *command*.
`-l`	Set `showmatch` option.
`-w`*n*	Set default window size to *n*, where *n* is the number of lines of text you want to view when editing.
`-x`	Unencrypt encrypted files. Note: not available in the international distribution of ConvexOS.

Exiting a `vi` session

`ZZ`	Exit `vi` after saving changes.
`:x`	Exit `vi` after saving changes.
`:wq`	Write your file (with or without changes) and quit `vi`.
`:q!`	Quit `vi` without saving changes.

Inserting text

a *text* ESC	Append *text* after cursor.
A *text* ESC	Append *text* at end of current line.
i *text* ESC	Insert *text* before cursor.
I *text* ESC	Insert *text* before first printable character on current line.
o *text* ESC	Open line below current line and insert *text*.
O *text* ESC	Open line above current line and insert *text*.
ESC	End insert and return to command mode.

Special insert-mode commands†

CTRL-h	Erase character left of cursor.
CTRL-w	Erase word left of cursor. Refer to "How vi defines a 'word'" on page 7.
CTRL-u	Erase current line.
CTRL-v	Enter next character as text and not as a command (for typing in ESC and other control characters).

†These may vary depending on your tty settings.

Moving cursor by character or line

`l`	Move right one character.
`h`	Move left one character.
`k`	Move up one line, same column.
`j`	Move down one line, same column.
→	Move right one character. †
←	Move left one character.†
↑	Move up one line, same column.†
↓	Move down one line, same column.†
SPACE	Move right one character.
BACKSPACE	Move left one character.
0 (zero)	Go to first column of current line.
$	Go to last character of current line.
n |	Go to column *n* of current line.
^	Go to first printable character on current line.
-	Up one line to first printable character.
+	Down one line, to first printable character.
RETURN	Down one line, to first printable character.

†Indicates command that cannot be used as a target. Refer to "Combining `vi` commands," page 2.

Moving cursor by word

How `vi` defines a "word"

word A string of alphanumeric characters delimited by a nonalphanumeric character or blank.

Or, a string of non-alphanumeric characters delimited by an alphanumeric character or blank.

For example, given the string:

```
THISis1word,, but thisis!@3
```

the words as in the example above are:

```
THISis1word

,,

but

thisis

!@

3
```

However, if you divide the above string into blank-delimited words, you have the following list:

```
THISis1word,,

but

thisis!@3
```

Using motion commands by word

Below are some motion commands using this definition of a word and blank-delimited word.

w Move right one word.

W Move right one blank-delimited word.

b Move left one word.

B Move left one blank-delimited word.

e Move to the end of current or next word.

E Move to the end of current or next blank-delimited word.

Moving through a file

*n*G	Go to line *n*. If *n* is omitted, go to end of file.
)	Move forward up to first character of next sentence.
(	Move backward to first character of current or previous sentence.
}	Move forward up to first character of next paragraph.
{	Move backward to first character of current or previous paragraph.
]]	Move forward up to first character of next `nroff` section or C function.
[[	Move backward to first character of current or previous `nroff` section or C function.

Altering text

Replacing and changing text

`r`*x*	Replace character at cursor with character *x*.
`R` *text* ESC	Replace original characters by typing over with *text*.
`c` *target text* ESC	Change *target* to *text*. (Refer to "Combining vi commands" on page 2.)
`cc` *text* ESC	Change current line to *text*.
`C` *text* ESC	Change from cursor to end of line to *text*.
`s` *text* ESC	Substitute character at cursor with *text*.
`S` *text* ESC	Substitute entire current line with *text*.

Deleting text

`x`	Delete character at cursor.
`X`	Delete character before cursor.
`d` *target*	Delete *target*. (Refer to "Combining vi commands" on page 2.)
`dd`	Delete current line.
`D`	Delete from cursor to end of line.

Undoing and repeating changes

`.`	Repeat previous change.
`u`	Undo previous change.
`U`	Restore current line to original condition, even after repeated changes. (As long as you haven't moved from it.)

Copying and moving text

"Yanking and putting" text

`y` *target*	Yank *target* to unnamed buffer. (Refer to the section "Combining `vi` commands," page 2.)
`yy`	Yank current line to unnamed buffer.
`Y`	Yank current line to unnamed buffer.
`p` (lower case)	Put back yanked or deleted lines below current line.
`P` (upper case)	Put back yanked or deleted lines above current line.
`J`	Join line below current line to the end of current line.

Shifting text

< *target*	Shift current line left one *target* value.
<<	Shift current line left one "shift width."
> *target*	Shift current line right one *target* value.
>>	Shift current line right one "shift width."

Searching

Searching for characters

`f`*x*	Find single character *x* on current line, moving right.
`F`*x*	Find single character *x* on current line, moving left.
`t`*x*	Move right up to single character *x* on current line.
`T`*x*	Move left up to single character *x* on current line.
`;`	Repeat previous `f`, `F`, `t`, or `T` command.
`,`	Reverse previous `f`, `F`, `t`, or `T` command.
`%`	Find matching (), { }, or [ ].

Searching with regular expressions[†]

/*reg_exp* RETURN	Find regular expression *reg_exp*, forward. See section "Using regular expressions," page 19.
? *reg_exp* RETURN	Find regular expression *reg_exp*, backward. See section "Using regular expressions," page 19.
`n`	Repeat previous / or ? command.
`N`	Repeat previous / or ? command in reverse direction through the file.

[†]For more information on regular expressions, refer to "Using regular expressions," page 19.

Changing your display

Positioning the cursor on the screen

H	Go to first printable character, top of the screen.
M	Go to first printable character, middle of the screen.
L	Go to first printable character, bottom of the screen.

Adjusting screen "window"

CTRL-f	Page forward one screen.
CTRL-b	Page backward one screen.
CTRL-d	Scroll down half a page.
CTRL-u	Scroll up half a page.†
CTRL-e	Scroll down one line.
CTRL-y	Scroll up one line.
z RETURN	Scroll current line to top of screen.
z–	Scroll current line to bottom of screen.
z.	Scroll current line to center of screen.
CTRL-l	(Lower case L) Redraw the screen.
CTRL-r	Refresh screen so that current changes can be seen.

Displaying file information

CTRL-g	Show current cursor position, file name, file size, and file status.

†Cannot take counts. Refer to "Repeating vi commands," page 1.

Using marks and tags

Tags are a way of marking places in files so that you can quickly and easily go there in order to edit or view that section. Using tags is a two-step process:

1. Creating the marker or tags file.†
2. Moving to the marker or tag.

Marking sections

`m` *x*	Mark current cursor position with *x*, where *x* is any letter from a to z.
`` ` ``*x*	Return to character position marked with letter *x*.
`'`*x*	Return to line marked with letter *x*.

Tagging sections in other files

`:ta` *filename*	Search the tag file for the file name *filename* and go to the line associated with it.
CTRL-]	Use the file name currently under the cursor as the *filename* argument for `:ta` and execute it.
CTRL-t	Pop the most recent tag off the tag stack and go to that file and line.

†For more information on tags files, refer to the ctags(1) or ftags(1) man pages.

Mapping and abbreviating

Mapping strings to sequences

:map	Show all mappings.
:map *string seq*	Map a string of characters to a sequence of one or more commands, where *string* is the string you wish to define, and *seq* is the command sequence you wish to assign to it. For example: **:map v xp** defines v to swap characters while in command mode. To map in insert mode, refer to `:map!` commands.
:unmap *string*	Unmap a string of characters, where *string* is the string you wish to undefine.
:map!	Show all mappings for insert mode.
:map! *string seq*	Same as `:map`, except mapping is done in insert mode.
:unmap! *string*	Same as `:unmap`, except unmaps insert mode mapping.

Abbreviating strings

:ab	Show all abbreviations.
:ab *abbrev string*	Define a character string to be another character string, where *abbrev* is the abbreviation, and *string* is the string you wish to abbreviate.
:unab *abbrev*	Unabbreviate *abbrev*.

Using named registers

`"` *x* `yy`	Yank the current line to register *x*, where *x* is any letter from a to z.
`@`*x*	Execute command stored in register *x*, where *x* is any letter from a to z.
`@@`	Execute the previous register command.

Using ex commands

Entering ex mode from vi

From vi, ex commands can be powerful tools. To execute a single ex command from vi, enter a colon followed by the ex command you wish to execute, like this:

: *ex-command*

The cursor moves to a command line at the bottom left-hand corner of your screen. Following is a list of ex commands.

Writing and quitting

:x	Quit editing, saving changes.
:q!	Quit editing without saving changes. Changes cannot be recovered.
:e *filename*	Quit editing current file and begin editing file *filename*.
:n	Edit next file in file list if multiple files were given as arguments to vi.
:*x*,*y* w *filename*	Write lines *x* through *y* to file *filename*. (Without the *x*,*y* addresses, w will write the current version of the entire file. Without *filename*, w will write the current version of the file to the original file name specified when ex or vi was invoked.)

Searching and substituting text†

Substituting text

`:`*x*`,` *y* `s/`*re*`/`*new*`/`*opts*	Substitute the first occurrence of the regular expression, *re*, with the string, *new*, from line numbers *x* through *y*. Where *opts* can be:

- `g` Global substitution. Substitute every occurrence throughout the specified lines.
- `c` Check substitution. You are asked to verify each substitution. Type `y` after each change you want to make.

Searching text

`:g/`*reg_exp*`/`*command*	Execute ex command, *command*, on all lines containing the regular expression *reg_exp*.
`:v/`*reg_exp*`/`*command*	Execute ex command, *command*, on all lines not containing the regular expression *reg_exp*.

†For more information on using regular expressions, refer to the section "Using regular expressions," page 19.

Other ex commands and features

Deleting lines

`:`*x*`,` *y* `d`	Delete lines *x* through *y*.

Transferring and moving lines

`:`*x*`,` *y* `t` *z*	Transfer lines *x* through *y* after line *z*, where *x*, *y*, and *z* are line numbers. (The original lines are not deleted.)
`:`*x*`,` *y* `m`*z*	Move lines *x* through *y* after line *z*. (The original lines are deleted.)

Reading lines from other files

`:`*n* `r` *filename*	Copy file *filename* into current file after line *n*. (If you don't specify line number *n*, `r` copies the file *filename* after the current line. Substitute *n* with 0 (zero) to indicate top of file.)

Executing shell commands

`:`*x*`,`*y*`!`*shell-cmd*	Execute the specified shell-command, *shell-cmd*, replacing lines *x* through *y* with the *shell-cmd* output.

Definitions of address variables

The following characters can be substituted for the line number arguments *x*, *y*, and *z* in the previously mentioned commands:

`$`	Represents last line.
`.`	Represents current line.
`%`	Represents first through last line and is an abbreviation for `1,$`.

Using regular expressions

Regular expressions are a way of specifying general types and groupings of characters, instead of literal ones. Below is a table of regular expressions.

Descriptions of regular expressions

c	Match character *c*.
c	Match character *c*. For use with regular expression characters that have special meanings. For example, `\.` matches a period.
^	Match beginning of line.
$	Match end of line.
.	Match any single character.
[$c_1c_2c_3$ ...]	Match any character contained in the brackets.
[^$c_1c_2c_3$ ...]	Match any character not contained in the brackets.
*x**	Match zero or more occurrences of *x*, where *x* is any regular expression.
\\(*subexpression*\\)	Delimits a subexpression within a regular expression, where *subexpression* is itself a regular expression.
n	Subsection reference, where *n* is an integer that corresponds to the position of the substring. The counting of substrings starts with one, not zero. For example: `\1` Refers to the first substring. `\2` Refers to the second substing.

Examples of regular expressions

`CONVEX`	Matches the word CONVEX anywhere on any line.
`^CONVEX`	Matches the word CONVEX when it begins a line.
`CONVEX$`	Matches the word CONVEX when it ends a line.
`CONVEX.$`	Matches a line ending in any single character that is preceded by the word CONVEX.
`^CONVEX$`	Matches a line that contains only one occurrence of the word CONVEX.
`^$`	Matches an empty line. The line will not be matched if it has any characters in it, including blanks or non-printed characters.
`[Cc]onvex`	Matches the words convex or Convex anywhere on any line.
`CONVEX\.$`	Matches any line ending in CONVEX followed by a period.

For more information

For more information see the ed(1) man page.

Setting editor options

You can `set` many options in `vi` such as automatic indentation, line numbering, ignoring case in searching, by issuing the `set` command. Here is how to use `set`:

`:set`	Show changed options.
`:set all`	Show current settings on all options.
`:set` *option*	Start *option*.
`:set no`*option*	Disable *option*.
`:set` *option*?	Show value of *option*.

Some useful editing options and how they are used with the `set` command are listed below.

`:set ai`	Automatically indent current line as far as the previous line.
`:set ic`	Ignore case differences while searching.
`:set list`	On the screen, represent tabs as `^I` and end-of-line markers as `$`.
`:set nu`	Show line numbers.
`:set ro`	Set read only. You will not be able to write to the file unless you use the `:w!` command.
`:set sm`	Show matching (or { when typing } or) in insert mode. Gives an audio signal if no match.
`:set sw=`*n*	Set **CTRL-t**, **CTRL-d**, <<, and >> to *n* spaces.

`:set wm=`*n*	Set right margin to size of *n* spaces and start a new line when it is reached. (Acts as an automatic carriage return.)
`:set ws`	Set wrap scan. When a search command reaches the end of file, it continues searching at the beginning of the file and continues to search until it reaches the line from which it was invoked.

CONVEX vi Quick Reference
Part no. 710-011330-001