CHAPTER 3

Vi Editor User's Guide

This chapter contains a user's guide to the **vi** editor, adapted from the LynxOS documentation set for LightStream® 2020 users. It is organized as follows:

- Introduction
- Editing Basics
- Terminology
- vi Regular Expressions
- Position Movement Commands
- Secondary Movement Commands
- Text-Changing Commands
- Colon Commands
- Miscellaneous Commands
- Editor Initialization
- Command Summary
- Notes on LynxOS vi

Introduction

The **vi** program is a full-screen text editor for creating and modifying files. Although it was designed for use as a software development tool, **vi** is suitable for a wide variety of editing tasks

The **vi** program determines the type of terminal on which it is being run by examining the TERM environment variable and the /etc/termcap file (or the TERMCAP environment variable). Any terminal defined in the /etc/termcap file can be used, as long as it supports at least cursor positioning and screen erase. On a LightStream 2020 node, TERM is set to vt100 in the .profile command file at login time. To change the value, type TERM=*value* at the bash prompt. For example, to set it to xterm, enter the following commands:

```
LSnode:1$ TERM=xterm
LSnode:1$ tset
LSnode:1$
```

See the *LightStream 2020 Installation Guide* for instructions for changing the default terminal type that takes effect upon logging in.

Editing Basics

If **vi** is started with no arguments, it creates a new, empty file which can be given a name and written to disk at any time. It may be started with the name of a new or existing file on the command line. (See Editor Initialization on page 10 for more details.)

The editor marks lines on the screen which are not part of the file with a single tilde (~). The editor always forces at least one line to exist.

The editor is always in *command* mode or *insert* mode (with some variants of insert mode). Command mode is the initial state of the editor, and is the mode from which nearly all editing commands are issued. Insert mode is used to add or modify text. Several commands switch the editor from command mode to insert mode, with side effects described below.

Getting Started

Because of the way **vi** is designed, one must know a large number of commands before editing becomes really convenient. However, a few simple commands are enough for rudimentary file operations, and are certainly enough for a careful typist to create new documents.

The first thing to learn is how to get to insert mode, since this is the normal way to add text to a file. The editor starts in command mode, and the entire keyboard is “hot”: any key pressed results in immediate action of some kind. The **i** key puts the editor in insert mode. Unfortunately, there is no indication of this change, visual or otherwise. In insert mode, all printable characters that are typed are inserted into the file before the current character. The return key terminates a line and moves to the next line on the screen. The backspace key (**^H**, control-**H**) can be used to back up and fix typing errors, but only within the current line. Similarly, **^W** erases over the most recently typed word, and **^U** erases up to the beginning of the line.

The **[Esc]** key, or control-**[**, returns from insert mode to command mode. In command mode, the **h** key moves one character left, **l** moves one character right, **k** moves one line up, and **j** moves one line down. (These keys form the line **hjkl** on the keyboard.) The editor sounds the bell if it is asked to move to a meaningless position, such as before the beginning of a line or after the end of the file.

The **x** key deletes individual characters. To delete a line, type **d** twice in succession. If something is deleted accidentally, recover it by typing **u**. Type **i** to re-enter insert mode at any time.

There are two ways to terminate the editing session. Type uppercase **Z** twice to save the current file and leave the editor. Type the string **:q!** plus carriage return to abort the editing session without saving changes made to the file.

Terminology

The following terms are used in the descriptions of commands:

Table 3-1 Terms Used in Command Descriptions

Term	Definition
text	Any series of ASCII characters.
buffer	A unit of memory used to hold text.
file buffer	The buffer that holds the file being edited.
white space	Tabs and spaces.

Term	Definition
word	An unbroken string of (1) letters, digits, and underscores (“_”), or (2) characters other than letters, digits, underscores, and white space, depending upon context.
big word	An unbroken string of characters other than white space.
unnamed buffer	A buffer containing the last text yanked or deleted.
window	The part of the file that is visible on the terminal.
line	A string of characters followed by a newline character. A “line” of text may take up more than one row on the screen.
current character	The character beneath the cursor.
current line	The line containing the cursor.
^*x*	Control character *x*, e.g. ^**B** means control-B (ASCII 2).
return	^**M** or the **[Return]** key.
command mode	The normal operating mode of the editor. Each key typed is taken as a command.
insert mode	The mode used for manually adding text to a file. Characters that are typed while in insert mode are inserted into the text.
replace mode	Like insert mode, except that characters typed replace characters in the buffer.
char	Any ASCII character.
named buffer	One of 26 buffers with names “a” through “z”.
mark name	A lowercase letter that names one of 26 marks that can be used to save a position in the file buffer.

vi Regular Expressions

Regular expressions are usually used for searching. Most characters match themselves in a search request, but the characters shown in the following table have special meanings:

Table 3-2 Regular Expressions Used in vi

Expression	Definition
.	Matches any single character.
[*chars***]**	Matches any *char* listed in the brackets. [*x*-*y*] is a range of characters, where the ASCII value of *x* precedes that of *y*.
[^*chars***]**	Matches any *char* not listed in the brackets.
^	Matches the beginning of a line when used as the first character in a regular expression.
$	Matches the end of a line when used as the last character in a regular expression.
*****	Matches zero or more instances of the preceding item, which may be a normal *char* or a 1-character “wildcard” such as . or [*chars*].
+	Matches one or more instances of the preceding item.
?	Matches zero or one instances of the preceding item.
\<	Matches the beginning of a word
\>	Matches the end of a word.

This feature may be turned off with the command **set nomagic**, as described on page 10. To search for one of these special characters, precede it by a backslash to prevent special interpretation. When special searching characters make several matches possible in a single line, **vi** matches the longest string possible. This becomes important for search-replace commands.

Note Parentheses and angle brackets are interpreted literally, and need not be preceded with backslashes.

Position Movement Commands

There are two kinds of movement commands. Position movement commands move the current position in the file based on file contents. These commands are shown in the following table. Note that uppercase and lowercase letters are distinct. There are commands for nearly every editing situation, but only a few are needed by beginners for simple editing.

With most position movement commands, optionally type an integer *n* before the command (with no space) to repeat the movement *n* times. The default is 1.

Table 3-3 Position Movement Commands

Command	Description
n`h` (or *n*^**H**)	Left *n* characters (1 by default). Within current line only.
n`j` (or *n*^**J**, or *n*^**N**)	Down *n* lines (1 by default), in the same column if possible.
n`k` (or *n*^**K**)	Up *n* lines (1 by default), in the same column if possible.
n`l` (or *n*^**L**, or *nspace*)	Right *n* characters (1 by default).
^**I**	Forward to the next tab stop.
n`w`	Forward to the beginning of the *n*th following *word*.
n`W`	Forward to the beginning of the *n*th following *big word*.
n`b`	Back to the beginning of the *n*th previous *word*.
n`B`	Back to the beginning of the *n*th previous *big word*.
n`e`	Forward to the end of the *n*th following *word*.
n`E`	Forward to the end of the *n*th following *big word*.
n`G`	The *n*th line of the file (default, last line).
`0`	The first character of the current line.
`_` (or `^`)	The first non-whitespace character on the current line.
`$`	The end of the current line.
`%`	The matching parenthesis, bracket, or brace.
`'`*mark* `''`	(Single quote.) The beginning of the line previously marked with *mark* (a single lowercase letter). If *mark* is **'**, move to the beginning of the line that was current just before the last absolute movement command such as **G**, **%**, or **'** itself.

Command	Description
‘*mark* ‘‘	(Back quote.) The position (line and column) previously marked with *mark* (a single lowercase letter). If *mark* is ‘ (back quote), move to the position that was current just before the last absolute movement command.
n+	The first non-whitespace character on the *n*th line below.
n-	The first non-whitespace character on the *n*th line above.
/	Search forward. Prompts on the last line of the screen for a search pattern (default = last pattern searched for).
?	Search backward. Prompts on the last line of the screen for a search pattern (default = last pattern searched for).
n	Search forward for last pattern searched for.
N	Search backward for last pattern searched for.
f*char*	Next instance of *char* on current line.
t*char*	Character preceding next instance of *char* on current line.
F*char*	Previous instance of *char* on current line.
T*char*	Character following previous instance of *char* on current line.
;	Repeat last **t** or **f** command.
,	Repeat last **T** or **F** command.
*n*H	Beginning of *n*th line from top of text (default = 1, home).
*n*L	Beginning of line *n*th row from bottom of text (default = 1).
n^M	First non-whitespace character on the *n*th line below (default = carriage return).

All the position movement commands can be used as operands to various change, delete, and substitute commands described farther on.

Secondary Movement Commands

Use secondary movement commands to move quickly through the file or to reposition the window in the file. Secondary movement commands cannot be used as operands to commands that change text.

Table 3-4 Secondary Movement Commands

Command	Description
^B	Move a full window backward.
^D	Move half a window forward.
^E	Move the window down one line. The cursor moves only if the current line scrolls off the screen.
^F	Move a full window forward.
^U	Move half a window backward.
^Y	Move the window up one line. The cursor moves only if the current line scrolls off the screen.

Command	Description
z.	Scroll the current line to the middle of the window.
z-	Scroll the current line to the bottom of the window.
z[Return]	Scroll the current line to the top of the window.

Text-Changing Commands

There are two kinds of commands that change text, those that are used with arguments and operands (movement commands), and those that are used alone.

The effect of any command that changes text can be repeated with the **.** command, or undone with the **u** or **U** command. Consequently, the sequence of entering insert mode, adding text, and returning to command mode can be repeated several times, or undone if the added text is unsatisfactory.

Several of the text-changing commands put a copy of the changed text into the unnamed buffer. These commands , are **c**, **d**, **s**, **x**, and **X**. Use **p** or **P** to retrieve the changed text.

Text-Changing Commands With Operands

The most important of the text-changing commands in **vi** operate on a variable region of text determined by an argument or operand of the command. For example, with a single command one may change the next three words, or delete lines up to the first instance of some pattern, or horizontally shift lines between the current position and some marked line.

Each of the commands that take operators or arguments is normally followed by one of the primary movement commands. The current cursor position defines one end of the region to be changed, and the movement command defines the other end. Alternatively, if a command is doubled, it affects the entire current line; for example, **dd** deletes the current line. If a repeat count *n* is included, it can be typed before the change command or before the movement command.

Table 3-5 To Change a Region Delimited by a Movement Command

Command	Description
c	Change the region of text delimited by the movement operand. If the end of the text region is within the current line, **vi** marks it with a **$** mark (dollar sign), enters replace mode until the cursor reaches the **$** mark, and then enters insert mode for additional typing. Otherwise, **vi** deletes text from the current position through the end of the region and enters insert mode.
d	Delete the region of text. Equivalent to using c and immediately pressing **[Esc]** to leave insert mode.
y	Copy text unchanged from the region to the unnamed buffer.
<	Shift each line of the region left one tab stop.
>	Shift each line of the region right one tab stop.
!	Filter the region through a LynxOS utility or other program. Prompts for a command. Standard output of the command replaces the text in the region.

Text-Changing Commands Without Operands

Other text-changing commands do not require operands, because they operate on predefined regions of text, such as individual characters or lines, and enter insert mode in various ways. A repeat count *n* may be given with many of these commands (default = 1).

Table 3-6 To Change a Predefined Text Object

Command	Description
`i`	Enter insert mode before the current character.
`I`	Move to the first non-whitespace character of the current line and enter insert mode.
`a`	Enter insert mode after the current character.
`A`	Move to the last character of the current line and enter insert mode after that character.
`o`	Add a blank line below the current line, move to the new line, and enter insert mode.
`O`	Add a blank line above the current line, move to the new line, and enter insert mode.
n`x`	Delete *n* characters to the right, starting with current character.
n`X`	Delete *n* characters to the left of current character.
`J`	Join the next line with the current line.
n`p`	Insert text from the unnamed buffer *n* times after the current position. Insert after the current line if the buffer contains lines of text.
n`P`	Insert text from the unnamed buffer *n* times before the current position. Insert before the current line if the buffer contains lines of text.
`R`	Enter replace mode, a variant of insert mode. Typing overwrites characters on the current line. As soon as the current line is completely overwritten, enter insert mode.
`r`*char*	Replace the current character with *char*.
`s`	Change the current character (same as **c1**).
`C`	Change the text in the current line (same as **c$**).
`C`	Delete the remainder of the current line (same as **d$**).
`Y`	Copy the remainder of the current line into the unnamed buffer (same as **y$**).
`S`	Change the current line (same as **cc**).
`u`	Undo the effects of the last text-changing command.
`U`	Multi-level undoing through the 9 most recent changes. (By contrast, repeating **u** has no net effect on the file.)
`&`	Repeat last search/replace command (see Colon Commands).
`~`	Repeat the effect of the last text-changing command.
`.`	Change case (uppercase or lowercase) of the current character, if it is alphabetic, and move right one character.

Miscellaneous Commands

The following commands are useful and important but defy ready categorization:

Table 3-7 Miscellaneous Commands

Command	Description
m*mark*	Mark the current line with *mark*, a single lowercase letter, for future reference by movement and change commands.
^R	Redraw the screen.
^G	Display information about the current position and file buffer.
^^	Toggle between the current file buffer and the last file buffer edited.
"*letter*	Use the buffer named *letter* for the following command. **P** or **p** inserts from *letter* instead of from the unnamed buffer, and commands that write into the unnamed buffer write into *letter* as well.
@*letter*	Perform commands found in the buffer named *letter* as if they were typed on the keyboard. To put commands into a buffer, create a new line in the main file and then delete it into the buffer. The **.** command repeats the effects of an invocation of buffered commands; the **u** and **U** commands repeat the effects of individual commands in the buffer.
^]	Interpret characters from the current position to the end of the word as a tag name, and go to that tag position. If this requires editing a new file, the current file must first be saved or the autowrite flag must be set. (See the **ta** command on page .)
ZZ	Save the current file and leave **vi**. (See also **:xit** on page .)
^C	Abort a search in progress or a colon command being entered.
^Z	Put the current editing session in the background and return to the invoking shell. To resume editing, use the appropriate shell command to bring **vi** back into the foreground.

Colon Commands

The **:** command accesses a family of commands which cannot easily be expressed by one or two keystrokes.

- When **:** is typed, **vi** prompts for a colon command.
- Colon commands may be abbreviated: one need type only enough characters to distinguish the intended command from other colon commands, then press **[Return]** or **[Esc]** to terminate the command.
- Several colon commands can be put on a single line separated by | characters, in the manner of a shell pipeline.

Colon Commands with a Range of Lines

The following colon commands operate on a range of lines within a file:

Table 3-8 Colon Commands

Command	Description
range **s**/*pat*/*repl*/g	The **s** command substitutes the replacement text *repl* in place of the pattern text *pat* on each line included in the *range*. If *range* is omitted, the default is the current line. If the optional **g** is included at the end of the command, each occurrence of *pat* is replaced, otherwise only the first instance on each line is affected. The pattern *pat* may be any regular expression (see page). If the replacement string *repl* contains the character **&**, the current instance of *pat* is substituted for it. If the replacement string *repl* contains the sequence \n, where *n* is a single digit from 1 to 9, that sequence is replaced by the *n*th portion of *pat* that is bracketed by **\(** and **\)**. Examples: Change every instance of cat to dog: :%s/cat/dog/g Remove all leading and training spaces in a file: :%s/^ *\(.*[^]\) *$.\1 Note that the pattern grouped in **\(** and **\)** explicitly specifies that the pattern end with a non-space. This is because **vi** always matches the longest patterns and sub-patterns. Parenthesize the first string of capital letters on the current line: :s/[A-Z]+/(&)/
range **w** *file* *range* **w!** *file* *range* **w>>** *file*	Write the lines in *range* to *file*. If *range* is omitted, write the entire file; in the absence of *file*, use the current file name. Use **w!** to overwrite an existing file. Use **w>>** to append to the end of an existing file.

The *range* parameter of the above commands consists of one or two of the following line specifiers:

Table 3-9 Line Specifier Strings Used to Specify Range of Colon Commands

Specifier	Definition
.	The current line.
$	The last line of the file.
n	Line number *n* of the file (a decimal integer).
'*mark*	Line previously marked by the 1-character *mark* with the **m** command.

If *range* consists of only one line specifier, it selects just that one line. If *range* consists of two line specifiers separated by a comma, it selects all lines from the first to the second, inclusive, regardless of the order in which the two line specifiers are entered. If no *range* is given, the default may be the current line or the entire file, depending on the command. The range specifier **%** is an abbreviation for **1,$**, selecting the entire file.

Colon Commands That Specify Files

The colon commands described in the following table determine the file to be edited:

Table 3-10 Colon Commands That Determine the File Being Edited

Command	Description
edit *file*	Start editing another file. If *file* is not given, the default is to reload the version of the current file found on disk (the last saved version, or the original if it has not been saved). If *file* is **#**, then **vi** switches to the alternate file, which is either the most recent file edited in this **vi** session, or the file most recently named in a colon command, even if that command failed. If the current file has not been saved, **vi** does not switch files unless the autowrite flag is set. It may be forced to switch by typing **edit!** *file*, but all changes to the current file are then lost.
next	Edit the next file in the parameter list that was given when **vi** was invoked. If the current file has not been saved, **vi** does not switch files unless the autowrite flag is set. It may be forced to switch by typing **next!**, but all changes to the current file are then lost.
quit	Exit **vi**. If the current file has not been saved, **vi** does not terminate unless the autowrite flag is set. It may be forced to exit by typing **quit!**, but all changes to the current file are then lost.
read *file*	Read the contents of *file* into the file buffer after the current line.
rewind	Change to the first parameter in the parameter list given when **vi** was invoked. The other parameters in the list can then be reexamined with the next command. If the current file has not been saved, **vi** does not switch files unless the autowrite flag is set. It may be forced to switch by typing **rewind!**, but all changes to the current file are then lost.
ta *tag*	Look up the string *tag* in a file named tags in the current working directory. Lines in this file are of the following form: *tag filename pattern* When it finds *tag*, **vi** changes to the file *filename* (if it is different from the current file) and moves to the first instance of *pattern* in that file. The tags file is usually constructed automatically, by the **ctags** program if the files contain C source code, or by some other program for other types of files. If the current file has not been saved, **vi** does not switch files unless the autowrite flag is set. It may be forced to switch by typing **ta!** *tag*, but all changes to the current file are then lost.
xit	Exit **vi** after saving the current file buffer. (See also **ZZ** on page .)

Commands That Tailor the Editing Environment

Several colon commands can be used to create keyboard shortcuts. The **:set** command can be used to manipulate the **vi** editor parameters described in the table on page 10.‘‘ These commands are as follows:

Table 3-11 Commands That Tailor the Editing Environment

Command	Description
map *key string*	Assign user-definable strings to particular characters. If *key* is typed in command mode, vi behaves as if *string* had been typed instead (but see below for map!).
map! *key string*	If *key* is typed in insert mode, vi behaves as if *string* had been typed instead. The same *key* can be used twice, once for insert mode and once for command mode.
unmap *key*	Remove any mapping established for *key* in command mode.
unmap! *key*	Remove any mapping established for *key* in insert mode.
set *flag*	Set *flag* to TRUE.
set no*flag*	Set *flag* to FALSE.
set *var=value*	Set *var* to the numeric value *value*.

The following **vi** editor parameters may be manipulated by **:set** commands.

Table 3-12 Editor Parameters That May be Set With the set Command

Parameter	Description
autoindent	If set, when text is added vi indents each new line by the same amount as the previous line.
autowrite	If set, vi writes the file to disk automatically when it switches between files or when it is stopped with ^Z.
ignorecase	If set, search commands ignore case.
list	If set, tabs in the file are made visible on the screen as ^I.
magic	If *not* set (**set nomagic**), special characters described on page need not be quoted with a backslash.
tabstop	Width in spaces of a tab stop; used also for shifting text.
wrapscan	If set, searches wrap around from one end of the file to the other.

Editor Initialization

When **vi** starts up, it looks for the variable VIINIT in the user's shell environment. If VIINIT is not set, **vi** looks for a file called .virc in the user's home directory. What it looks for is a list of colon commands, delimited by newlines, which it executes.

After executing commands found in VIINIT or $HOME/.virc, then **vi** looks for a .virc file in the current working directory. Because the commands in a local .virc take effect after those described above, the local file may be used to set parameters for editing a special set of files grouped in one directory.

If **vi** is started with the **-t** option, it uses a tag file created by **ctags** or some other method to start and load the file containing a requested tag. If a positioning command (see page 3) is given before any tag or file names, **vi** starts at the specified position rather than at the beginning of the file.

Command Summary

The following table lists all of the **vi** commands, as described above, in ASCII order. Unused characters are noted.

Table 3-13 Command-Mode Command Summary

Command	Description
^A	Not used.
^B	Move a full window backward.
^C	Abort a search in progress or a colon command being entered.
^D	Move half a window forward.
^E	Move the window down one line. The cursor moves only if the current line scrolls off the screen.
^F	Move a full window forward.
^G	Display information about the current position and file buffer.
n^H	Left *n* characters (1 by default). Within current line only.
^I	Forward to the next tab stop.
n^J	Down *n* lines (1 by default), in the same column if possible.
n^K	Up *n* lines (1 by default), in the same column if possible.
n^L	Right *n* characters (1 by default).
n^M	First non-whitespace character on the *n*th line below (default = carriage return).
n^N	Down *n* lines (1 by default), in the same column if possible.
^O	Not used.
^P	Not used.
^Q	Not used (XON character for flow control).
^R	Redraw the screen.
^S	Not used (XOFF character for flow control).
^T	Not used.
^U	Move half a window backward.

Command	Description
^V	Not used.
^W	Not used.
^X	Not used.
^Y	Move the window up one line. The cursor moves only if the current line scrolls off the screen.
^Z	Put the current editing session in the background and return to the invoking shell. To resume editing, use the appropriate shell command to bring vi back into the foreground.
^[	Not used in command mode (exits insert mode).
^\	Not used.
^]	Interpret characters from the current position to the end of the word as a tag name, and go to that tag position. If this requires editing a new file, the current file must first be saved or the autowrite flag must be set. (See the ta command below under Colon Commands.)
^^	Toggle between the current file buffer and the last file buffer edited.
^_	Not used.
n[space]	Right *n* characters (1 by default).
!	Filter the region through a LynxOS utility or other program. Prompts for a command. Standard output of the command replaces the text in the region.
"*letter*	Use the buffer named *letter* for the following command. P or p inserts from *letter* instead of from the unnamed buffer, and commands that write into the unnamed buffer write into *letter* as well.
#	Not used.
$	The end of the current line.
%	The matching parenthesis, bracket, or brace.
&	Repeat last search/replace command (see Colon Commands).
'*mark* ''	(Single quote.) The beginning of the line previously marked with *mark* (a single lowercase letter). If *mark* is ' (single quote) , move to the beginning of the line that was current just before the last absolute movement command such as **G**, **%**, or ' itself.
(	Not used.
)	Not used.
*	Not used.
n+	The first non-whitespace character on the *n*th line below.
'	(Single quote.) Repeat last T or F command.
n-	The first non-whitespace character on the *n*th line above.
.	Repeat the effect of the last text-changing command.
/	Search forward. Prompts on the last line of the screen for a search pattern (default = last pattern searched for).
0	The first character of the current line.
1...9	Not used, except to specify repeat counts to other commands.
:	Used to introduce a colon command.
;	Repeat last t or f command.
<	Shift each line of the region left one tab stop.

Command	Description
=	Not used.
>	Shift each line of the region right one tab stop.
?	Search backward. Prompts on the last line of the screen for a search pattern (default = last pattern searched for).
@*letter*	Perform commands found in the buffer named *letter* as if they were typed on the keyboard. Put commands into a buffer by creating a new line in the main file and then deleting it into the buffer. Use the . command to repeat the effects of an invocation of buffered commands; u and U commands only repeat the effects of individual commands in the buffer.
A	Move to the last character of the current line and enter insert mode after that character.
*n*B	Back to the beginning of the *n*th previous *big word*.
C	Change text in the current line (same as c$).
D	Delete remainder of the current line (same as d$).
*n*E	Forward to the end of the *n*th following *big word*.
F*char*	Previous instance of *char* on current line.
*n*G	The *n*th line of the file (default, last line).
*n*H	Beginning of *n*th line from top of text (default = 1, home).
I	Move to the first non-whitespace character of the current line and enter insert mode.
J	Join the next line with the current line.
K	Not used.
*n*L	Beginning of line *n*th row from bottom of text (default = 1).
M	Not used.
N	Search backward for last pattern searched for.
O	Add a blank line above the current line, move to the new line, and enter insert mode.
*n*P	Insert text from the unnamed buffer *n* times before the current position. Insert before the current line if the buffer contains lines of text.
Q	Not used.
R	Enter replace mode, a variant of insert mode. Typing overwrites characters on the current line. As soon as the current line is completely overwritten, enter insert mode.
S	Change the current line (same as cc).
T*char*	Character following previous instance of *char* on current line.
U	Multi-level undoing through the 9 most recent changes. (By contrast, repeating u has no net effect on the file.)
V	Not used.
*n*W	Forward to the beginning of the *n*th following *big word*.
*n*X	Delete *n* characters to the left of current character.
Y	Copy remainder of the current line into the unnamed buffer (same as y$).
ZZ	Save the current file and leave vi.
[	Not used.
\	Not used.
]	Not used.
^	The first non-whitespace character on the current line.

Command	Description
_	The first non-whitespace character on the current line.
‘*mark*	The position (line and column) previously marked with *mark* (a single lowercase letter).
‘‘	If *mark* is ‘, move to the position that was current just before the last absolute movement command.
a	Enter insert mode after the current character.
*n*b	Back to the beginning of the *n*th previous *word*.
c	Change the region of text delimited by the movement operand. If the end of the text region is within the current line, vi marks it with a $ mark (dollar sign), enters replace mode until the cursor reaches the $ mark, and then enters insert mode for additional typing. Otherwise, vi deletes text from the current position through the end of the region and enters insert mode.
d	Delete the region of text. Equivalent to using c and immediately pressing [Esc] to leave insert mode.
*n*e	Forward to the end of the *n*th following *word*.
f*char*	Next instance of *char* on current line.
g	Not used.
*n*h	Left *n* characters (1 by default). Within current line only.
i	Enter insert mode before the current character.
*n*j	Down *n* lines (1 by default), in the same column if possible.
*n*k	Up *n* lines (1 by default), in the same column if possible.
*n*l	Right *n* characters (1 by default).
m*mark*	Mark the current line with *mark*, a single lowercase letter, for future reference by movement and change commands.
n	Search forward for last pattern searched for.
o	Add a blank line below the current line, move to the new line, and enter insert mode.
*n*p	Insert text from the unnamed buffer *n* times after the current position. Insert after the current line if the buffer contains lines of text.
q	Not used.
r*char*	Replace the current character with *char*.
s	Change the current character (same as c1).
t*char*	Character preceding next instance of *char* on current line.
u	Undo the effects of the last text-changing command.
v	Not used.
*n*w	Forward to the beginning of the *n*th following *word*.
*n*x	Delete *n* characters to the right, starting with current character.
y	Copy text unchanged from the region to the unnamed buffer.
z*return*	Scroll the current line to the top of the window.
z-	Scroll the current line to the bottom of the window.
z.	Scroll the current line to the middle of the window.
{	Not used.
|	Not used.
}	Not used.

Command	Description
~	Change case (uppercase or lowercase) of the current character, if it is alphabetic, and move right one character.

Notes on LynxOS vi

The following list summarizes the salient differences between the version of **vi** running under LynxOS and other popular versions of the **vi** editor. Many of the differences are enhancements. Differences that reflect bugs found in other versions are not listed here.

- **U** undoes the last text change for up to 9 changes. In other versions, **U** undoes all changes made to the current line since it became the current line.
- After an undo command that causes a change in file position, LynxOS **vi** remembers the previous position so that `''` returns.
- The unnamed buffer is not deleted after the **edit** colon command so that text can be more easily transferred from one file to another.
- the **p** and **P** commands accept a repeat count.
- Backspace can be used to delete whitespace inserted automatically in autoindent mode. Other **vi** versions only recognize **^D** for this purpose.
- The **+** and **?** regular expression operators are supported.
- The **Y** command is defined as **y$** to be consistent with the **C** and **D** commands. Other versions of **vi** treat **Y** as if it were **yy**.
- There is no “open” mode, a terminal must support cursor movement to be used with LynxOS **vi**.
- All shift commands can be repeated with the **.** command.
- The **n** command always searches forward, and the **N** command always searches backward. In other versions of **vi**, the **n** command searches in the most recent search direction and **N** searches in the opposite of the most recent direction.
- LynxOS **vi** reads startup commands from the VIINIT environment variable instead of the EXINIT variable.