Ex/Vi Reference Manual

Keith Bostic

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, California 94720

August 16, 1994

Abstract

This document is the reference guide for the 4.4BSD implementations of **nex**/**nvi**, which are reimplementations of the historic Berkeley **ex**/**vi** editors.

Acknowledgements

Bruce Englar encouraged the early development of the historic **ex**/**vi** editor. Peter Kessler helped bring sanity to version 2's command layout. Bill Joy wrote versions 1 and 2.0 through 2.7, and created the framework that users see in the present editor. Mark Horton added macros and other features and made **ex**/**vi** work on a large number of terminals and Unix systems.

Nvi is originally derived from software contributed to the University of California, Berkeley by Steve Kirkendall, the author of the **vi** clone **elvis**.

IEEE Standard Portable Operating System Interface for Computer Environments (POSIX) 1003.2 style Regular Expression support was done by Henry Spencer.

The curses library was originally done by Ken Arnold. Scrolling and reworking for **nvi** was done by Elan Amir.

The Institute of Electrical and Electronics Engineers has given us permission to reprint portions of their documentation. Portions of this document are reprinted and reproduced from IEEE Std 1003.2-1992, IEEE Standard Portable Operating System Interface for Computer Environments (POSIX), copyright 1992 by the Institute of Electrical and Electronics Engineers, Inc.

The financial support of UUNET Communications Services is gratefully acknowledged.

1. Description

Vi is a screen oriented text editor. **Ex** is a line-oriented text editor. **Ex** and **vi** are different interfaces to the same program, and it is possible to switch back and forth during an edit session. **View** is the equivalent of using the −**R** (read-only) option of **vi**.

This reference manual is the one provided with the **nex**/**nvi** versions of the **ex**/**vi** text editors. **Nex**/**nvi** are intended as bug-for-bug compatible replacements for the original Fourth Berkeley Software Distribution (4BSD) **ex**/**vi** programs. This reference manual is accompanied by a traditional-style manual page. That manual page describes the functionality found in **ex**/**vi** in far less detail than the description here. In addition, it describes the system interface to **ex**/**vi**, e.g. command line options, session recovery, signals, environmental variables, and similar things.

This reference is intended for users already familiar with **ex**/**vi**. Anyone else should almost certainly read a good tutorial on the editor first. If you are in an unfamiliar environment, and you absolutely have to get work done immediately, see the section entitled ‘‘**Fast Startup**’’ in the manual page. It is probably enough to get you started.

There are a few features in **nex**/**nvi** that are not found in historic versions of **ex**/**vi**. Some of the more interesting of those features are briefly described in the section entitled ‘‘**Additional Features**’’ near the end of this document. For the rest of this document, **nex**/**nvi** is used only when it is necessary to distinguish it from the historic implementations of **ex**/**vi**.

Future versions of this software will be periodically made available by anonymous ftp, and can be retrieved from `ftp.cs.berkeley.edu`, in the directory `ucb/4bsd`.

2. Startup Information

Ex/**vi** interprets one of two possible environmental variables and reads up to three of five possible files during startup. The variables and files are expected to contain **ex** commands, not **vi** commands. In addition, they are interpreted *before* the file to be edited is read, and therefore many **ex** commands may not be used. Generally, any command that requires output to the screen or that needs a file upon which to operate, will cause an error if included in a startup file or environmental variable.

Because the **ex** command set supported by **nex**/**nvi** is a superset of the command set supported by most historical implementations of **ex**, **nex**/**nvi** can use the startup files created for the historical implementations, but the converse may not be true.

If the −**s** (the historic − option) is specified, or if standard input is redirected from a file, all environmental variables and startup files are ignored.

Otherwise, startup files and environmental variables are handled in the following order:

(1) The file `/etc/vi.exrc` is read, as long as it is owned by root or the effective user ID of the user.

(2) The environmental variable `NEXINIT` (or the variable `EXINIT`, if `NEXINIT` is not set) is interpreted.

(3) If neither `NEXINIT` or `EXINIT` was set, and the `HOME` environmental variable is set, the file `$HOME/.nexrc` (or the file `$HOME/.exrc`, if `$HOME/.nexrc` does not exist) is read, as long as the effective user ID of the user is root or is the same as the owner of the file.

(4) If the **exrc** option was turned on by one of the previous startup information sources, the file `.nexrc` (or the file `.exrc`, if `.nexrc` does not exist) is read, as long as the effective user ID of the user is the same as the owner of the file.

No startup file is read if it is writable by anyone other than its owner.

It is not an error for any of the startup environmental variables or files not to exist.

Once all environmental variables are interpreted, and all startup files are read, the first file to be edited is read in (or a temporary file is created). Then, any commands specified using the −**c** option are executed, in the context of that file.

3. Recovery

There is no recovery program for **nex**/**nvi**, nor does **nex**/**nvi** run setuid. Recovery files are created readable and writable by the owner only. Users may recover any file which they can read, and the superuser may recover any edit session.

Edit sessions are backed by files in the directory named by the **recdir** option (the directory `/var/tmp/vi.recover` by default), and are named ‘‘**vi.XXXXXX**’’, where ‘‘**XXXXXX**’’ is a number related to the process ID. When a file is first modified, a second recovery file containing an email message for the user is created, and is named ‘‘**recover.XXXXXX**’’, where, again, ‘‘**XXXXXX**’’ is associated with the process ID. Both files are removed at the end of a normal edit session, but will remain if the edit session is abnormally terminated or the user runs the **ex preserve** command.

The **recdir** option may be set in either the user's or system's startup information, changing the recovery directory. (Note, however, that if a memory based file system is used as the backup directory, each system reboot will delete all of the recovery files! The same caution applies to directories such as `/tmp` which are cleared of their contents by a system reboot, or `/usr/tmp` which is periodically cleared of old files on many systems.)

The recovery directory should be owned by root, or at least by a pseudo-user. In addition, if directory ‘‘sticky-bit’’ semantics are available, the directory should have the sticky-bit set so that files may only be removed by their owners. The recovery directory must be read, write, and executable by any user, i.e. mode 1777.

If the recovery directory does not exist, **ex**/**vi** will attempt to create it. This can result in the recovery directory being owned by a normal user, which means that that user will be able to remove other user's recovery and backup files. This is annoying, but is not a security issue as the user cannot otherwise access or modify the files.

The recovery file has all of the necessary information in it to enable the user to recover the edit session. In addition, it has all of the necessary email headers for *sendmail*(8). When the system is rebooted, all of the files in `/var/tmp/vi.recover` named ‘‘**recover.XXXXXX**’’ should be sent to their owners, by email, using the **−t** option of **sendmail** (or a similar mechanism in other mailers). If **ex**/**vi** receives a hangup (SIGHUP) signal, or the user executes the **ex preserve** command, **ex**/**vi** will automatically email the recovery information to the user.

If your system does not have the **sendmail** utility (or a mailer program which supports its interface) the source file `nvi/common/recover.c` will have to be modified to use your local mail delivery programs. Note, if **nex**/**nvi** is changed to use another mailer, it is important to remember that the owner of the file given to the mailer is the **nex**/**nvi** user, so nothing in the file should be trusted as it may have been modified in an effort to compromise the system.

Finally, the owner execute bit is set on backup files when they are created, and unset when they are first modified, e.g. backup files that have no associated email recovery file will have this bit set. (There is also a small window where empty files can be created and not yet have this bit set. This is due to the method in which the files are created.) Such files should be deleted when the system reboots.

A simple way to do this cleanup is to insert the following Bourne shell script into your `/etc/rc.local` (or other startup) file. The script should work with the historic Bourne shell, a POSIX 1003.2 shell or the Korn shell. (A copy of this script is included as `nvi/install/recover.script` in the **nex**/**nvi** distribution.)

```
#       @(#)recover.script      8.7 (Berkeley) 8/16/94
#
# Script to recover nvi edit sessions.
#
RECDIR=/var/tmp/vi.recover
SENDMAIL=/usr/lib/sendmail
echo 'Recovering nvi editor sessions.'

# Check editor backup files.
vibackup=`echo $RECDIR/vi.*`
if [ "$vibackup" != "$RECDIR/vi.*" ]; then
      for i in $vibackup; do
            # Only test files that are readable.
            if test ! -r $i; then
                  continue
            fi

            # Unmodified nvi editor backup files either have the
            # execute bit set or are zero length.  Delete them.
            if test -x $i -o ! -s $i; then
                  rm $i
            fi
      done
fi

# It is possible to get incomplete recovery files, if the editor crashes
# at the right time.
virecovery=`echo $RECDIR/recover.*`
if [ "$virecovery" != "$RECDIR/recover.*" ]; then
      for i in $virecovery; do
            # Only test files that are readable.
            if test ! -r $i; then
                  continue
            fi

            # Delete any recovery files that are zero length, corrupted,
            # or that have no corresponding backup file.  Else send mail
            # to the user.
            recfile=`awk '/^X-vi-recover-path:/{print $2}' < $i`
            if test -n "$recfile" -a -s "$recfile"; then
                  $SENDMAIL -t < $i
            else
                  rm $i
            fi
      done
fi
```

If you are not using the default value for the **recdir** option, be sure to substitute the value you're using for the `RECDIR` value in the recovery script.

If the path of your system's **sendmail** program (or whatever mailer you're using) is not `/usr/lib/sendmail`, be sure to substitute the correct pathname for the `SENDMAIL` value in the recovery script. Consult the manual page for details on recovering preserved or aborted editing sessions.

4. Sizing the Screen

The size of the screen can be set in a number of ways. **Ex/vi** takes the following steps until values are obtained for both the number of rows and number of columns in the screen.

(1) If the environmental variable `LINES` exists, it is used to specify the number of rows in the screen.

(2) If the environmental variable `COLUMNS` exists, it is used to specify the number of columns in the screen.

(3) The TIOCGWINSZ *ioctl*(2) is attempted on the standard error file descriptor.

(4) The termcap entry (or terminfo entry on System V machines) is checked for the ‘‘li’’ entry (rows) and the ‘‘co’’ entry (columns).

(5) The number of rows is set to 24, and the number of columns is set to 80.

If a window change size signal (SIGWINCH) is received, the new window size is retrieved using the TIOCGWINSZ *ioctl*(2) call, and all other information is ignored.

5. Character Display

In both **ex** and **vi** printable characters as defined by *isprint*(3) are displayed using the local character set.

Non-printable characters, for which *iscntrl*(3) returns true, and which are less than octal \076, are displayed as the string ‘‘`^<character>`’’, where `<character>` is the character that is the original character’s value offset from the ‘‘`@`’’ character. For example, the octal character \001 is displayed as ‘‘`^A`’’. If *iscntrl*(3) returns true for the octal character \177, it is displayed as the string ‘‘`^?`’’. All other characters are displayed as either hexadecimal values, in the form ‘‘`0x<high-halfbyte> ... 0x<low-halfbyte>`’’, or as octal values, in the form ‘‘`\<high-one-or-two-bits> ... \<low-three-bits>`’’. The display of unknown characters is based on the value of the **octal** option.

In **vi** command mode, the cursor is always positioned on the last column of characters which take up more than one column on the screen. In **vi** text input mode, the cursor is positioned on the first column of characters which take up more than one column on the screen.

6. Multiple Screens

Nvi supports multiple screens by dividing the window into regions. It also supports stacks of screens by permitting the user to change the set of screens that are currently displayed.

The command **split** divides the current screen into two regions of approximately equal size. If a list of files are specified as arguments to the **split** command, the list of files to be edited is initialized as if the **next** command had been used. If no files are specified, the new screen will begin by editing the same file as the previous screen.

When more than one screen is editing a file, changes in any screen are reflected in all other screens editing the same file. Exiting any screen without saving any changes (or explicitly discarding them) is permitted until the last screen editing the file is exited.

The **resize** command permits resizing of individual screens. Screens may be grown, shrunk or set to an absolute number of rows.

The **^W** command is used to switch between screens. Each **^W** moves to the next lower screen in the window, or to the first screen in the window if there are no lower screens.

The **bg** command ‘‘backgrounds’’ the current screen. The screen disappears from the window, and the rows it occupied are taken over by a neighboring screen. It is an error to attempt to background the only screen in the window.

The **display screens** command displays the names of the files associated with the current backgrounded screens in the window.

The **fg [file]** command ‘‘foregrounds’’ the first screen in the list of backgrounded screens that is associated with its argument. If no file argument is specified, the first screen on the list is foregrounded. Foregrounding consists of backgrounding the current screen, and replacing its space in the window with the foregrounded screen.

If the last screen in the window is exited, and there are backgrounded screens, the first screen on the list of backgrounded screens takes over the window.

7. Regular Expressions and Replacement Strings

Regular expressions are used in line addresses, as the first part of the **ex substitute**, **global**, and **vglobal** commands, and in search patterns.

The regular expressions supported by **ex**/**vi** are, by default, the Basic Regular Expressions (BRE’s) described in the IEEE POSIX Standard 1003.2. The **extended** option causes all regular expressions to be interpreted as the Extended Regular Expressions (ERE’s) described by the same standard. (See *re_format*(7) for more information.) Generally speaking, BRE’s are the Regular Expressions found in *ed*(1) and *grep*(1), and ERE’s are the Regular Expressions found in *egrep*(1).

The following is not intended to provide a description of Regular Expressions. The information here only describes strings and characters which have special meanings in the **ex**/**vi** version of RE’s, or options which change the meanings of characters that normally have special meanings in RE’s.

(1) An empty RE (e.g. ‘‘`//`’’ or ‘‘`??`’’ is equivalent to the last RE used.

(2) The construct ‘‘`\<`’’ matches the beginning of a word.

(3) The construct ‘‘`\>`’’ matches the end of a word.

(4) The character ‘‘`~`’’ matches the replacement part of the last **substitute** command.

When the **magic** option is *not* set, the only characters with special meanings are a ‘‘`^`’’ character at the beginning of an RE, a ‘‘`$`’’ character at the end of an RE, and the escaping character ‘‘`\`’’. The characters ‘‘`.`’’, ‘‘`*`’’, ‘‘`[`’’ and ‘‘`~`’’ are treated as ordinary characters unless preceded by a ‘‘`\`’’; when preceded by a ‘‘`\`’’ they regain their special meaning.

Replacement strings are the second part of a **substitute** command.

The character ‘‘`&`’’ (or ‘‘`\&`’’ if the **magic** option is *not* set) in the replacement string stands for the text matched by the RE that is being replaced. The character ‘‘`~`’’ (or ‘‘`\~`’’ if the **magic** option is *not* set) stands for the replacement part of the previous **substitute** command. It is only valid after a **substitute** command has been performed.

The string ‘‘`\#`’’, where ‘‘`#`’’ is an integer value from 1 to 9, stands for the text matched by the portion of the RE enclosed in the ‘‘`#`’’’th set of escaped parentheses, e.g. ‘‘`\(`’’ and ‘‘`\)`’’. For example, ‘‘`s/abc\(.*\)def/\1/`’’ deletes the strings ‘‘`abc`’’ and ‘‘`def`’’ from the matched pattern.

The strings ‘‘`\l`’’, ‘‘`\u`’’, ‘‘`\L`’’ and ‘‘`\U`’’ can be used to modify the case of elements in the replacement string. The string ‘‘`\l`’’ causes the next character to be converted to lowercase; the string ‘‘`\u`’’ behaves similarly, but converts to uppercase (e.g. `s/abc/\U&/` replaces the string `abc` with `ABC`). The strings ‘‘`\L`’’ causes characters up to the end of the string or the next occurrence of the strings ‘‘`\e`’’ or ‘‘`\E`’’ to be converted to lowercase; the string ‘‘`\U`’’ behaves similarly, but converts to uppercase.

If the entire replacement pattern is ‘‘`%`’’, then the last replacement pattern is used again.

In **vi**, inserting a `<control-M>` into the replacement string will cause the matched line to be split into two lines at that point. (The `<control-M>` will be discarded.)

8. General Editor Description

When **ex** or **vi** are executed, the text of a file is read (or a temporary file is created), and then all editing changes happen within the context of the copy of the file. *No changes affect the actual file until the file is written out,* either using a write command or another command which is affected by the **autowrite** option.

All files are locked (using the *flock*(2) or *fcntl*(2) interfaces) during the edit session, to avoid inadvertently making modifications to multiple copies of the file. If a lock cannot be obtained for a file because it is locked by another process, the edit session is read-only (as if the **readonly** option or the **−R** flag had been specified). If a lock cannot be obtained for other reasons, the edit session will continue, but the file status information (see the **<control-G>** command) will reflect this fact.

Both **ex** and **vi** are modeful editors, i.e. they have two modes, ‘‘command’’ mode and ‘‘text input’’ mode. The former is intended to permit you to enter commands which modifies already existing text. The latter is intended to permit you to enter new text. When **ex** first starts running, it is in command mode, and usually displays a prompt (see the **prompt** option for more information). The prompt is a single colon (‘‘`:`’’) character. There are three commands that switch **ex** into text input mode: **append**, **change** and **insert**. Once in input mode, entering a line containing only a single period (‘‘`.`’’) terminates text input mode and returns to command mode, where the prompt is redisplayed.

When **vi** first starts running, it is in command mode as well. There are eleven commands that switch **vi** into text input mode: **A**, **a**, **C**, **c**, **I**, **i**, **O**, **o**, **R**, **S** and **s**. Once in input mode, entering an `<escape>` character terminates text input mode and returns to command mode.

The following words have special meanings in both the **ex** and **vi** command descriptions:

<interrupt>

The interrupt character is used to interrupt the current operation. Normally `<control-C>`, whatever character is set for the current terminal is used.

<literal next>

The literal next character is used to escape the subsequent character from any special meaning. This character is always `<control-V>`. If the terminal is not set up to do XON/XOFF flow control, then `<control-Q>` is used to mean literal next as well.

current pathname

The pathname of the file currently being edited by vi. When the percent character (‘‘`%`’’) appears in a file name entered as part of an **ex** command argument, it is replaced by the current pathname. (The ‘‘`%`’’ character can be escaped by preceding it with a backslash.)

alternate pathname

The name of the last file name mentioned in an **ex** command, or, the previous current pathname if the last file mentioned becomes the current file. When the hash mark character (‘‘`#`’’) appears in a file name entered as part of an **ex** command argument, it is replaced by the alternate pathname. (The ‘‘`#`’’ character can be escaped by preceding it with a backslash.)

buffer

One of a number of named areas for saving copies of text. Commands that change or delete text can save the changed or deleted text into a specific buffer, for later use, if the command allows it (i.e. the **ex change** command cannot save the changed text in a named buffer). Buffers are named with a single character, preceded by a double quote, e.g. `"<character>`. Historic implementations of **ex**/**vi** limited `<character>` to the alphanumeric characters; **nex**/**nvi** permits the use of any character.

Buffers named by uppercase characters are the same as buffers named by lowercase characters, e.g. the buffer named by the English character ‘‘`A`’’ is the same as the buffer named by the character ‘‘`a`’’, with the exception that, if the buffer contents are being changed (as with a text deletion or **vi change** command), the text is *appended* to the buffer, instead of replacing the current contents.

The buffers named by the numeric characters (in English, ‘‘`1`’’ through ‘‘`9`’’), are special, in that if at least one line is changed or deleted in the file, (or a command changes or deletes a region that crosses a line boundary) a copy of the text is placed into the numeric buffer ‘‘`1`’’, regardless of the user specifying another buffer in which to save it. Before this copy is done, the previous contents of buffer ‘‘`1`’’ are moved into buffer ‘‘`2`’’, ‘‘`2`’’ into buffer ‘‘`3`’’, and so on. The contents of buffer ‘‘`9`’’ are discarded. In **vi**, text may be explicitly stored into the numeric buffers. In this case, the buffer rotation described above occurs before the replacement of the buffer’s contents. (Text cannot be explicitly stored into the numeric buffers in **ex** because of ambiguities that this would cause in the **ex** command syntax.)

When a **vi** command synopsis shows both a `[buffer]` and a `[count]`, they may be presented in any order.

Finally, all buffers are either ‘‘line’’ or ‘‘character’’ oriented. All **ex** commands which store text into buffers are line oriented. Some **vi** commands which store text into buffers are line oriented, and some are character oriented; the description for each applicable **vi** command notes whether text copied into buffers using the command is line or character oriented. In addition, the **vi** command **display buffers** displays the current orientation for each buffer. Generally, the only importance attached to this orientation is that if the buffer is subsequently inserted into the text, line oriented buffers create new lines for each of the lines they contain, and character oriented buffers create new lines for any lines *other* than the first and last lines they contain. The first and last lines are inserted into the text at the current cursor position, becoming part of the current line. If there is more than one line in the buffer, however, the current line itself will be split.

unnamed buffer
The unnamed buffer is a text storage area which is used by commands that take a buffer as an argument, when no buffer is specified by the user. There is no way to explicitly reference this buffer.

9. Vi Description

Vi takes up the entire screen to display the edited file, except for the bottom line of the screen. The bottom line of the screen is used to enter **ex** commands, and for **vi** error and informational messages. If no other information is being displayed, the default display can show the current cursor row and cursor column, an indication of whether the file has been modified, and the current mode of the editor. See the **ruler**, **showdirty** and **showmode** options for more information.

Empty lines do not have any special representation on the screen, but lines on the screen that would logically come after the end of the file are displayed as a single tilde (‘‘~’’) character. To differentiate between empty lines and lines consisting of only whitespace characters, use the **list** option. Historically, implementations of **vi** have also displayed some lines as single asterisk (‘‘`@`’’) characters. These were lines that were not correctly displayed, i.e. lines on the screen that did not correspond to lines in the file, or lines that did not fit on the current screen. **Nvi** never displays lines in this fashion.

Vi is a modeful editor, i.e. it has two modes, ‘‘command’’ mode and ‘‘text input’’ mode. When **vi** first starts, it is in command mode. There are several commands that change **vi** into text input mode. The `<escape>` character is used to resolve the text input into the file, and exit back into command mode. In **vi** command mode, the cursor is always positioned on the last column of characters which take up more than one column on the screen. In **vi** text insert mode, the cursor is positioned on the first column of characters which take up more than one column on the screen.

Generally, if the cursor line and cursor column are not on the screen, then the screen is scrolled (if the target cursor is close) or repainted (if the target cursor is far away) so that the cursor is on the screen. If the screen is scrolled, it is moved a minimal amount, and the cursor line will usually appear at the top or bottom of the screen. In the screen is repainted, the cursor line will appear in the center of the screen, unless the cursor is sufficiently close to the beginning or end of the file that this is not possible. If the **leftright** option is set, the screen may be scrolled or repainted in a horizontal direction as well as in a vertical one.

A major difference between the historical **vi** presentation and **nvi** is in the scrolling and screen oriented position commands, **<control-B>**, **<control-D>**, **<control-E>**, **<control-F>**, **<control-U>**, **<control-Y>**, **H**, **L** and **M**. In historical implementations of **vi**, these commands acted on physical (as opposed to logical, or screen) lines. For lines that were sufficiently long in relation to the size of the screen, this meant that single line scroll commands might repaint the entire screen, scrolling or screen positioning command might not change the screen or move the cursor at all, and some lines simply could not be displayed, even though **vi** would edit the file that contained them. In **nvi**, these commands act on logical, i.e. screen lines. You are unlikely to notice any difference unless you are editing files with lines significantly longer than a screen width.

Vi keeps track of the currently ‘‘most attractive’’ cursor position. Each command description (for commands that can change the current cursor position), specifies if the cursor is set to a specific location in the line, or if it is moved to the ‘‘most attractive cursor position’’. The latter means that the cursor is moved to the cursor position that is vertically as close as possible to the current cursor position. If the current line is shorter than the cursor position **vi** would select, the cursor is positioned on the last character in the line. (If the line is empty, the cursor is positioned on the first column of the line.) If a command moves the cursor to the most attractive position, it does not alter the current cursor position, and a subsequent movement will again attempt to move the cursor to that position. Therefore, although a movement to a line shorter than the currently most attractive position will cause the cursor to move to the end of that line, a subsequent movement to a longer line will cause the cursor to move back to the most attractive position.

In addition, the **$** command makes the end of each line the most attractive cursor position rather than a specific column.

Each **vi** command described below notes where the cursor ends up after it is executed. This position is described in terms of characters on the line, i.e. ‘‘the previous character’’, or, ‘‘the last character in the line’’. This is to avoid needing to continually refer to on what part of the character the cursor rests.

The following words have special meaning for **vi** commands.

previous context

The position of the cursor before the command which caused the last absolute movement was executed. Each **vi** command described in the next section that is considered an absolute movement is so noted. In addition, specifying *any* address to an **ex** command is considered an absolute movement.

motion

A second **vi** command can be used as an optional trailing argument to the **vi !**, <, >, **c**, **d**, **y**, and (depending on the **tildeop** option) **˜** commands. This command indicates the end of the region of text that’s affected by the command. The motion command may be either the command character repeated (in which case it means the current line) or a cursor movement command. In the latter case, the region affected by the command is from the starting or stopping cursor position which comes first in the file, to immediately before the starting or stopping cursor position which comes later in the file. Commands that operate on lines instead of using beginning and ending cursor positions operate on all of the lines that are wholly or partially in the region. In addition, some other commands become line oriented depending on where in the text they are used. The command descriptions below note these special cases.

The following commands may all be used as motion components for **vi** commands:

```
<control-A>    <control-H>  <control-J>  <control-M>
<control-N>    <control-P>      <space>            $
          %   ’<character>            (            )
          +              ,            -            /
          0              ;            ?            B
          E              F            G            H
          L              M            N            T
          W             [[           ]]            ^
          _   ‘<character>            b            e
          f              h            j            k
          l              n            t            w
          {              |            }
```

The optional count prefix available for some of the **vi** commands that take motion commands, or the count prefix available for the **vi** commands that are used as motion components, may be included and is *always* considered part of the motion argument. For example, the commands ‘‘`c2w`’’ and ‘‘`2cw`’’ are equivalent, and the region affected by the **c** command is two words of text. In addition, if the optional count prefix is specified for both the **vi** command and its motion component, the effect is multiplicative and is considered part of the motion argument. For example, the commands ‘‘`4cw`’’ and ‘‘`2c2w`’’ are equivalent, and the region affected by the **c** command is four words of text.

count

A positive number used as an optional argument to most commands, either to give a size or a position (for display or movement commands), or as a repeat count (for commands that modify text). The count argument is always optional and defaults to 1 unless otherwise noted in the command description.

When a **vi** command synopsis shows both a `[buffer]` and `[count]`, they may be presented in any order.

bigword

A set of non-whitespace characters preceded and followed by whitespace characters or the beginning or end of the file or line.

Groups of empty lines (or lines containing only whitespace characters) are treated as a single bigword.

word

Generally, in languages where it is applicable, **vi** recognizes two kinds of words. First, a sequence of letters, digits and underscores, delimited at both ends by: characters other than letters, digits, or underscores; the beginning or end of a line; the beginning or end of the file. Second, a sequence of characters other than letters, digits, underscores, or whitespace characters, delimited at both ends by: a letter, digit, underscore, or whitespace character; the beginning or end of a line; the beginning or end of the file.

Groups of empty lines (or lines containing only whitespace characters) are treated as a single word.

paragraph

An area of text that begins with either the beginning of a file, an empty line, or a section boundary, and continues until either an empty line, section boundary, or the end of the file.

Groups of empty lines (or lines containing only whitespace characters) are treated as a single paragraph.

Additional paragraph boundaries can be defined using the **paragraph** option.

section

An area of text that starts with the beginning of the file or a line whose first character is an open brace (‘‘`{`’’) and continues until the next section or the end of the file.

Additional section boundaries can be defined using the **sections** option.

sentence

An area of text that begins with either the beginning of the file or the first nonblank character following the previous sentence, paragraph, or section boundary and continues until the end of the file or a or a period (‘‘`.`’’) exclamation point (‘‘`!`’’) or question mark (‘‘`?`’’) character, followed by either an end-of-line or two whitespace characters. Any number of closing parentheses (‘‘`)`’’), brackets (‘‘`]`’’) or double-quote (‘‘`"`’’) characters can appear between the period, exclamation point, or question mark and the whitespace characters or end-of-line.

Groups of empty lines (or lines containing only whitespace characters) are treated as a single sentence.

10. Vi Commands

The following section describes the commands available in the command mode of the **vi** editor. In each entry below, the tag line is a usage synopsis for the command character. In addition, the final line and column the cursor rests upon, and any options which affect the command are noted.

[count] <control-A>

Search forward `count` times for the current word. The current word begins at the first non-whitespace character on or after the current cursor position, and extends up to the next non-word character or the end of the line. The search is literal, i.e. no characters in the word have any special meaning in terms of Regular Expressions. It is an error if no matching pattern is found between the starting position and the end of the file.

The **<control-A>** command is an absolute movement. The **<control-A>** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line:	Set to the line where the word is found.
Column:	Set to the first character of the word.
Options:	Affected by the **extended**, **ignorecase** and **wrapscan** options.

[count] <control-B>

Page backward `count` screens. Two lines of overlap are maintained by displaying the window starting at line `(top_line - count * window_size) + 2`, where `window_size` is the value of the **window** option. (In the case of split screens, this size is corrected to the current screen size.) This is an error if the movement is past the beginning of the file.

The **<control-B>** command is an absolute movement.

Line:	Set to the last line of text displayed on the screen.
Column:	Set to the first nonblank character of the line.
Options:	None.

[count] <control-D>

Scroll forward `count` lines. If `count` is not specified, scroll forward the number of lines specified by the last **<control-D>** or **<control-U>** command. If this is the first **<control-D>** or **<control-U>** command, scroll forward half the number of lines in the screen. (In the case of split screens, the default scrolling distance is corrected to half the current screen size.) This is an error if the movement is past the end of the file.

The **<control-D>** command is an absolute movement.

Line:	Set to the current line plus the number of lines scrolled.
Column:	Set to the first nonblank character of the line.
Options:	None.

[count] <control-E>

Scroll forward `count` lines, leaving the cursor on the current line and column, if possible. This is an error if the movement is past the end of the file.

Line:	Unchanged unless the current line scrolls off the screen, in which case it is set to the first line on the screen.
Column:	Unchanged unless the current line scrolls off the screen, in which case it is set to the most attractive cursor position.
Options:	None.

[count] <control-F>

Page forward `count` screens. Two lines of overlap are maintained by displaying the window starting at line `top_line + count * window_size - 2`, where `window_size` is the value of the **window** option. (In the case of split screens, this size is corrected to the current screen size.) This is an error if the movement is past the end of the file.

The **<control-F>** command is an absolute movement.

Line: Set to the first line on the screen.
Column: Set to the first nonblank character of the current line.
Options: None.

<control-G>

Display the file information. The information includes the current pathname, the current line, the number of total lines in the file, the current line as a percentage of the total lines in the file, if the file has been modified, was able to be locked, if the file's name has been changed, and if the edit session is read-only.

Line: Unchanged.
Column: Unchanged.
Options: None.

<control-H>
[count] h

Move the cursor back `count` characters in the current line. This is an error if the cursor is on the first character in the line.

The **<control-H>** and **h** commands may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line: Unchanged.
Column: Set to the `current - count` character, or, the first character in the line if `count` is greater than or equal to the number of characters in the line before the cursor.
Options: None.

[count] <control-J>
[count] <control-N>
[count] j

Move the cursor down `count` lines without changing the current column. This is an error if the movement is past the end of the file.

The **<control-J>**, **<control-N>** and **j** commands may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is line oriented.

Line: Set to the current line plus `count`.
Column: The most attractive cursor position.
Options: None.

<control-L>
<control-R>

Repaint the screen.

Line: Unchanged.
Column: Unchanged.
Options: None.

[count] <control-M>
[count] +

Move the cursor down `count` lines to the first nonblank character of that line. This is an error if the movement is past the end of the file.

The **<control-M>** and **+** commands may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is line oriented.

Line: Set to the current line plus `count`.
Column: Set to the first nonblank character in the line.
Options: None.

[count] <control-P>
[count] k

Move the cursor up `count` lines, without changing the current column. This is an error if the movement is past the beginning of the file.

The **<control-P>** and **k** commands may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is line oriented.

Line: Set to the current line minus count.
Column: The most attractive cursor position.
Options: None.

<control-T>

Return to the most recent tag context. The **<control-T>** command is an absolute movement.

Line: Set to the context of the previous tag command.
Column: Set to the context of the previous tag command.
Options: None.

<control-U>

Scroll backward `count` lines. If `count` is not specified, scroll backward the number of lines specified by the last **<control-D>** or **<control-U>** command. If this is the first **<control-D>** or **<control-U>** command, scroll backward half the number of lines in the screen. (In the case of split screens, the default scrolling distance is corrected to half the current screen size.) This is an error if the movement is past the beginning of the file.

The **<control-U>** command is an absolute movement.

Line: Set to the current line minus the amount scrolled.
Column: Set to the first nonblank character in the line.
Options: None.

<control-W>

Switch to the next lower screen in the window, or, to the first screen if there are no lower screens in the window.

Line: Set to the previous cursor position in the window.
Column: Set to the previous cursor position in the window.
Options: None.

<control-Y>

Scroll backward `count` lines, leaving the current line and column as is, if possible. This is an error if the movement is past the beginning of the file.

Line: Unchanged unless the current line scrolls off the screen, in which case it is set to the last line of text displayed on the screen.
Column: Unchanged unless the current line scrolls off the screen, in which case it is the most attractive cursor position.
Options: None.

<control-Z>

Suspend the current editor session. If the file has been modified since it was last completely written, and the **autowrite** option is set, the file is written before the editor session is suspended. If this write fails, the editor session is not suspended.

Line: Unchanged.
Column: Unchanged.
Options: Affected by the **autowrite** option.

<escape>

Execute **ex** commands or cancel partial commands. If an **ex** command is being entered (e.g. **/**, **?**, **:** or **!**), the command is executed. If a partial command has been entered, e.g. or the command is cancelled. Otherwise, it is an error.

Line: When an **ex** command is being executed, the current line is set as described for that command. Otherwise, unchanged.
Column: When an **ex** command is being executed, the current column is set as described for that command. Otherwise, unchanged.
Options: None.

<control-]>

Push a tag reference onto the tag stack. The tags files (see the **tags** option for more information) are searched for a tag matching the current word. The current word begins at the first non-whitespace character on or after the current cursor position, and extends up to the next non-word character or the end of the line. If a matching tag is found, the current file is discarded and the file containing the tag reference is edited.

If the current file has been modified since it was last completely written, the command will fail. The **<control-]>** command is an absolute movement.

Line: Set to the line containing the matching tag string.
Column: Set to the start of the matching tag string.
Options: Affected by the **tags** and **taglength** options.

<control-^>

Switch to the most recently edited file.

If the file has been modified since it was last completely written, and the **autowrite** option is set, the file is written out. If this write fails, the command will fail. Otherwise, if the current file has been modified since it was last completely written, the command will fail.

Line: Set to the line the cursor was on when the file was last edited.
Column: Set to the column the cursor was on when the file was last edited.
Options: Affected by the **autowrite** option.

[count] <space>
[count] l

Move the cursor forward `count` characters without changing the current line. This is an error if the cursor is on the last character in the line.

The **<space>** and **l** commands may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented. In addition, these commands may be used as the motion components of other commands when the cursor is on the last character in the line, without error.

Line: Unchanged.
Column: Set to the current character plus the next `count` characters, or to the last character on the line if `count` is greater than the number of characters in the line after the current character.

Options: None.

[count] ! motion shell-argument(s)

Replace text with results from a shell command. Pass the lines specified by the `count` and `motion` arguments as standard input to the program named by the **shell** option, and replace those lines with the output (both standard error and standard output) of that command.

After the motion is entered, **vi** prompts for arguments to the shell command.

Within those arguments, ''%'' and ''#'' characters are expanded to the current and alternate pathnames, respectively. The ''!'' character is expanded with the command text of the previous **!** or **:!** commands. (Therefore, the command **!!** repeats the previous **!** command.) The special meanings of ''%'', ''#'' and ''!'' can be overridden by escaping them with a backslash. If no **!** or **:!** command has yet been executed, it is an error to use an unescaped ''!'' character. The **!** command does *not* do shell expansion on the strings provided as arguments. If any of the above expansions change the arguments the user entered, the command is redisplayed at the bottom of the screen.

Vi then executes the program named by the **shell** option, with a **−c** flag followed by the arguments (which are bundled into a single argument).

The **!** command is permitted in an empty file.

If the file has been modified since it was last completely written, the **!** command will warn you.

Line: The first line of the replaced text.
Column: The first column of the replaced text.
Options: Affected by the **shell** option.

[count] # +|-|#

Increment or decrement the current number. The current number begins at the first non-number character on or before the current cursor position, or the beginning of the line, and extends up to the first non-number character on or after the current cursor position or the end of the line. If the trailing character is a `+`, the number is incremented by `count`. If the trailing character is a `-`, the number is decremented by `count`. If the trailing character is a `#`, the previous increment or decrement is repeated.

The format of the number (decimal, hexadecimal, and octal, and leading 0's) is retained unless the new value cannot be represented in the previous format.

Line: Unchanged.
Column: Set to the first character in the cursor word.
Options: None.

[count] $

Move the cursor to the end of a line. If `count` is specified, the cursor moves down `count - 1` lines.

It is not an error to use the **$** command when the cursor is on the last character in the line or when the line is empty.

The **$** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented, unless the cursor is at, or before the first nonblank character in the line, in which case it is line oriented. It is not an error to use the **$** command as a motion component when the cursor is on the last character in the line, although it is an error when the line is empty.

Line: Set to the current line plus `count` minus 1.
Column: Set to the last character in the line.
Options: None.

%

Move to the matching character. The cursor moves to the parenthesis or curly brace which *matches* the parenthesis or curly brace found at the current cursor position or which is the closest one to the right of the cursor on the line. It is an error to execute the **%** command on a line without a parenthesis or curly brace. Historically, any `count` specified to the **%** command was ignored.

The **%** command is an absolute movement. The **%** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented, unless the starting point of the region is at or before the first nonblank character on its line, and the ending point is at or after the last nonblank character on its line, in which case it is line oriented.

Line: Set to the line containing the matching character.
Column: Set to the matching character.
Options: None.

&

Repeat the previous substitution command on the current line.

Historically, any `count` specified to the **&** command was ignored.

Line: Unchanged.
Column: Unchanged if the cursor was on the last character in the line, otherwise, set to the first nonblank character in the line.
Options: Affected by the **edcompatible**, **extended**, **ignorecase** and **magic** options.

´<character>
‘<character>

Return to a context marked by the character `<character>`. If `<character>` is the ‘‘`'`’’ or ‘‘`` ` ``’’ character, return to the previous context. If `<character>` is any other character, return to the context marked by that character (see the **m** command for more information). If the command is the **´** command, only the line value is restored, and the cursor is placed on the first nonblank character of that line. If the command is the **‘** command, both the line and column values are restored.

It is an error if the context no longer exists because of line deletion. (Contexts follow lines that are moved, or which are deleted and then restored.)

The **´** and **‘** commands are both absolute movements. They may be used as a motion component for other **vi** commands. For the **´** command, any text copied into a buffer is line oriented. For the **‘** command, any text copied into a buffer is character oriented, unless it both starts and stops at the first character in the line, in which case it is line oriented. In addition, when using the **‘** command as a motion component, commands which move backward and started at the first character in the line, or move forward and ended at the first character in the line, are corrected to the last character of the starting and ending lines, respectively.

Line: Set to the line from the context.
Column: Set to the first nonblank character in the line, for the **´** command, and set to the context's column for the **‘** command.
Options: None.

[count] (

Back up `count` sentences.

The **(** command is an absolute movement. The **(** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented, unless the starting and stopping points of the region are the first character in the line, in which case it is line oriented. In the latter case, the stopping point of the region is adjusted to be the end of the line immediately before it, and not the original cursor position.

Line: Set to the line containing the beginning of the sentence.
Column: Set to the first nonblank character of the sentence.
Options: None.

[count])

Move forward `count` sentences.

The **)** command is an absolute movement. The **)** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented, unless the starting point of the region is the first character in the line, in which case it is line oriented. In the latter case, if the stopping point of the region is also the first character in the line, it is adjusted to be the end of the line immediately before it.

Line: Set to the line containing the beginning of the sentence.
Column: Set to the first nonblank character of the sentence.
Options: None.

[count] ,

Reverse find character `count` times. Reverse the last **F**, **f**, **T** or **t** command, searching the other way in the line, `count` times.

The **,** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line: Unchanged.
Column: Set to the searched-for character.
Options: None.

[count] –

Move to first nonblank of the previous line, `count` times.

This is an error if the movement is past the beginning of the file.

The **-** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is line oriented.

Line: Set to the current line minus `count`.
Column: Set to the first nonblank character in the line.
Options: None.

[count] .

Repeat the last **vi** command that modified text. The repeated command may be a command and motion component combination. If `count` is specified, it replaces *both* the count specified for the repeated command, and, if applicable, for the repeated motion component. If `count` is not specified, the counts originally specified to the command being repeated are used again.

As a special case, if the **.** command is executed immediately after the **u** command, the change log is rolled forward or backward, depending on the action of the **u** command.

Line: Set as described for the repeated command.
Column: Set as described for the repeated command.

Options: None.

/RE<carriage-return>
/RE/ [offset]<carriage-return>
?RE<carriage-return>
?RE? [offset]<carriage-return>
N
n

Search forward or backward for a regular expression. The commands beginning with a slash (``/'') character are forward searches, the commands beginning with a question mark (``?'') are backward searches. **Vi** prompts with the leading character on the last line of the screen for a string. It then searches forward or backward in the file for the next occurrence of the string, which is interpreted as a Basic Regular Expression.

The / and **?** commands are absolute movements. They may be used as the motion components of other **vi** commands, in which case any text copied into a buffer is character oriented, unless the search started and ended on the first column of a line, in which case it is line oriented. In addition, forward searches ending at the first character of a line, and backward searches beginning at the first character in the line, are corrected to begin or end at the last character of the previous line. (Note, forward and backward searches can occur for both / and **?** commands, if the **wrapscan** option is set.)

If an offset from the matched line is specified (i.e. a trailing ``/'' or ``?'' character is followed by a signed offset), the buffer will always be line oriented (e.g. ``/string/+0'' will always guarantee a line orientation).

The **n** command repeats the previous search.

The **N** command repeats the previous search, but in the reverse direction.

Missing RE's (e.g. ``//<carriage-return>'', ``/<carriage-return>'', ``??<carriage-return>'', or ``?<carriage-return>'' search for the last search RE, in the indicated direction.

Searches may be interrupted using the <interrupt> character.

Line: Set to the line in which the match occurred.
Column: Set to the first character of the matched string.
Options: Affected by the **edcompatible**, **extended**, **ignorecase**, **magic**, and **wrapscan** options.

0

Move to the first character in the current line. It is not an error to use the **0** command when the cursor is on the first character in the line,

The **0** command may be used as the motion component of other **vi** commands, in which case it is an error if the cursor is on the first character in the line.

Line: Unchanged.
Column: Set to the first character in the line.
Options: None.

:

Execute an ex command. **Vi** prompts for an **ex** command on the last line of the screen, using a colon (``:'') character. The command is terminated by a <carriage-return>, <newline> or <escape> character; all of these characters may be escaped by using a <literal next> character. The command is then executed.

If the **ex** command writes to the screen, **vi** will prompt the user for a `<carriage-return>` before continuing when the **ex** command finishes. Large amounts of output from the **ex** command will be paged for the user, and the user prompted for a `<carriage-return>` or `<space>` key to continue. In some cases, a quit (normally a ‘‘q’’ character) or `<interrupt>` may be entered to interrupt the **ex** command.

When the **ex** command finishes, and the user is prompted to resume visual mode, it is also possible to enter another ‘‘`:`’’ character followed by another **ex** command.

Line: The current line is set as described for the **ex** command.
Column: The current column is set as described for the **ex** command.
Options: None.

[count] ;

Repeat the last character find `count` times. The last character find is one of the **F**, **f**, **T** or **t** commands.

The **;** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line: Unchanged.
Column: Set to the searched-for character.
Options: None.

[count] < motion
[count] > motion

Shift lines left or right. Shift the number of lines in the region specified by the motion component, times `count`, left (for the < command) or right (for the > command) by the number of columns specified by the **shiftwidth** option. Only whitespace characters are deleted when shifting left; once the first character in the line contains a nonblank character, the **shift** will succeed, but the line will not be modified.

Line: Unchanged.
Column: Set to the first nonblank character in the line.
Options: Affected by the **shiftwidth** option.

@ buffer

Execute a named buffer. Execute the named buffer as **vi** commands. The buffer may include **ex** commands, too, but they must be expressed as a **:** command. If the buffer is line oriented, `<newline>` characters are logically appended to each line of the buffer. If the buffer is character oriented, `<newline>` characters are logically appended to all but the last line in the buffer.

If the buffer name is ‘‘`@`’’, or ‘‘`*`’’, then the last buffer executed shall be used. It is an error to specify ‘‘`@@`’’ or ‘‘`**`’’ if there were no buffer previous executions. The text of a macro may contain an **@** command, and it is possible to create infinite loops in this manner. (The `<interrupt>` character may be used to interrupt the loop.)

Line: The current line is set as described for the command(s).
Column: The current column is set as described for the command(s).
Options: None.

[count] A

Enter input mode, appending the text after the end of the line. If `count` is specified, the text is repeatedly input `count - 1` more times after input mode is exited.

Line: Set to the last line upon which characters were entered.

Column: Set to the last character entered.
Options: Affected by the **altwerase**, **autoindent**, **beautify**, **showmatch**, **ttywerase** and **wrapmargin** options.

[count] B

Move backward `count` bigwords. Move the cursor backward to the beginning of a bigword by repeating the following algorithm: if the current position is at the beginning of a bigword or the character at the current position cannot be part of a bigword, move to the first character of the preceding bigword. Otherwise, move to the first character of the bigword at the current position. If no preceding bigword exists on the current line, move to the first character of the last bigword on the first preceding line that contains a bigword.

The **B** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line: Set to the line containing the word selected.
Column: Set to the first character of the word selected.
Options: None.

[buffer] [count] C

Change text from the current position to the end-of-line. If `count` is specified, the input text replaces from the current position to the end-of-line, plus `count - 1` subsequent lines.

Line: Set to the last line upon which characters were entered.
Column: Set to the last character entered.
Options: Affected by the **altwerase**, **autoindent**, **beautify**, **showmatch**, **ttywerase** and **wrapmargin** options.

[buffer] D

Delete text from the current position to the end-of-line.

It is not an error to execute the **D** command on an empty line.

Line: Unchanged.
Column: Set to the character before the current character, or, column 1 if the cursor was on column 1.
Options: None.

[count] E

Move forward `count` end-of-bigwords. Move the cursor forward to the end of a bigword by repeating the following algorithm: if the current position is the end of a bigword or the character at that position cannot be part of a bigword, move to the last character of the following bigword. Otherwise, move to the last character of the bigword at the current position. If no succeeding bigword exists on the current line, move to the last character of the first bigword on the next following line that contains a bigword.

The **E** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line: Set to the line containing the word selected.
Column: Set to the last character of the word selected.
Options: None.

[count] F <character>

Search `count` times backward through the current line for `<character>`.

The **F** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line: Unchanged.
Column: Set to the searched-for character.
Options: None.

[count] G

Move to line `count`, or the last line of the file if `count` not specified.

The **G** command is an absolute movement. The **G** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is line oriented.

Line: Set to `count`, if specified, otherwise, the last line.
Column: Set to the first nonblank character in the line.
Options: None.

[count] H

Move to the screen line `count - 1` lines below the top of the screen.

The **H** command is an absolute movement. The **H** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is line oriented.

Line: Set to the line `count - 1` lines below the top of the screen.
Column: Set to the first nonblank character of the *screen* line.
Options: None.

[count] I

Enter input mode, inserting the text at the beginning of the line. If `count` is specified, the text input is repeatedly input `count - 1` more times.

Line: Set to the last line upon which characters were entered.
Column: Set to the last character entered.
Options: None.

[count] J

Join lines. If `count` is specified, `count` lines are joined; a minimum of two lines are always joined, regardless of the value of `count`.

If the current line ends with a whitespace character, all whitespace is stripped from the next line. Otherwise, if the next line starts with a open parenthesis (‘‘`(`’’) do nothing. Otherwise, if the current line ends with a question mark (‘‘`?`’’), period (‘‘`.`’’) or exclamation point (‘‘`!`’’), insert two spaces. Otherwise, insert a single space.

It is not an error to join lines past the end of the file, i.e. lines that do not exist.

Line: Unchanged.
Column: Set to the character after the last character of the next-to-last joined line.
Options: None.

[count] L

Move to the screen line `count - 1` lines above the bottom of the screen.

The **L** command is an absolute movement. The **L** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is line oriented.

Line: Set to the line `count - 1` lines above the bottom of the screen.

Column: Set to the first nonblank character of the *screen* line.
Options: None.

M

Move to the screen line in the middle of the screen.

The **M** command is an absolute movement. The **M** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is line oriented.

Historically, any `count` specified to the **M** command was ignored.

Line: Set to the line in the middle of the screen.
Column: Set to the first nonblank character of the *screen* line.
Options: None.

[count] O

Enter input mode, appending text in a new line above the current line. If `count` is specified, the text input is repeatedly input `count - 1` more times.

Historically, any `count` specified to the **O** command was ignored.

Line: Set to the last line upon which characters were entered.
Column: Set to the last character entered.
Options: Affected by the **altwerase**, **autoindent**, **beautify**, **showmatch**, **ttywerase** and **wrapmargin** options.

[buffer] P

Insert text from a buffer. Text from the buffer (the unnamed buffer by default) is inserted before the current column or, if the buffer is line oriented, before the current line.

Line: Set to the lowest numbered line insert, if the buffer is line oriented, otherwise unchanged.
Column: Set to the first nonblank character of the appended text, if the buffer is line oriented, otherwise, the last character of the appended text.
Options: None.

Q

Exit **vi** (or visual) mode and switch to **ex** mode.

Line: Unchanged.
Column: No longer relevant.
Options: None.

[count] R

Enter input mode, replacing the characters in the current line. If `count` is specified, the text input is repeatedly input `count - 1` more times.

If the end of the current line is reached, no more characters are replaced and any further characters input are appended to the line.

Line: Set to the last line upon which characters were entered.
Column: Set to the last character entered.
Options: Affected by the **altwerase**, **autoindent**, **beautify**, **showmatch**, **ttywerase** and **wrapmargin** options.

[buffer] [count] S

Substitute `count` lines.

Line: Set to the last line upon which characters were entered.
Column: Set to the last character entered.
Options: Affected by the **altwerase**, **autoindent**, **beautify**, **showmatch**, **ttywerase** and **wrapmargin** options.

[count] T <character>

Search backward, `count` times, through the current line for the character *after* the specified `<character>`.

The **T** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line: Unchanged.
Column: Set to the character *after* the searched-for character.
Options: None.

U

Restore the current line to its state before the cursor last moved to it.

Line: Unchanged.
Column: The first character in the line.
Options: None.

[count] W

Move forward `count` bigwords. Move the cursor forward to the beginning of a bigword by repeating the following algorithm: if the current position is within a bigword or the character at that position cannot be part of a bigword, move to the first character of the next bigword. If no subsequent bigword exists on the current line, move to the first character of the first bigword on the first following line that contains a bigword.

The **W** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line: The line containing the word selected.
Column: The first character of the word selected.
Options: None.

[buffer] [count] X

Delete `count` characters before the cursor. If the number of characters to be deleted is greater than or equal to the number of characters to the beginning of the line, all of the characters before the current cursor position, to the beginning of the line, are deleted.

Line: Unchanged.
Column: Set to the current character minus `count`, or the first character if count is greater than the number of characters in the line before the cursor.
Options: None.

[buffer] [count] Y

Copy (or ‘‘yank’’) `count` lines into the specified buffer.

Line: Unchanged.
Column: Unchanged.
Options: None.

ZZ

Write the file and exit **vi**. The file is only written if it has been modified since the last complete write of the file to any file.

The **ZZ** command will exit the editor after writing the file, if there are no further files to edit. Entering two ‘‘quit’’ commands (i.e. **wq**, **quit**, **xit** or **ZZ**) in a row will override this check and the editor will exit, ignoring any files that have not yet been edited.

Line: Unchanged.
Column: Unchanged.
Options: None.

[count] [[

Back up `count` section boundaries.

The **[[** command is an absolute movement. The **[[** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented, unless the starting position is column 0, in which case it is line oriented.

This is an error if the movement is past the beginning of the file.

Line: Set to the previous line that is `count` section boundaries back, or the first line of the file if no more section boundaries exist preceding the current line.
Column: Set to the first nonblank character in the line.
Options: Affected by the **sections** option.

[count]]]

Move forward `count` section boundaries.

The **]]** command is an absolute movement. The **]]** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented, unless the starting position is column 0, in which case it is line oriented.

This is an error if the movement is past the end of the file.

Line: Set to the line that is `count` section boundaries forward, or to the last line of the file if no more section boundaries exist following the current line.
Column: Set to the first nonblank character in the line.
Options: Affected by the **sections** option.

^

Move to first nonblank character on the current line.

The **^** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line: Unchanged.
Column: Set to the first nonblank character of the current line.
Options: None.

[count] _

Move down `count - 1` lines, to the first nonblank character. The **_** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is line oriented.

It is not an error to execute the **_** command when the cursor is on the first character in the line.

Line: The current line plus `count - 1`.
Column: The first nonblank character in the line.
Options: None.

[count] a

Enter input mode, appending the text after the cursor. If `count` is specified, the text input is repeatedly input `count - 1` more times.

Line: Set to the last line upon which characters were entered.
Column: Set to the last character entered.
Options: Affected by the **altwerase**, **autoindent**, **beautify**, **showmatch**, **ttywerase** and **wrapmargin** options.

[count] b

Move backward `count` words. Move the cursor backward to the beginning of a word by repeating the following algorithm: if the current position is at the beginning of a word, move to the first character of the preceding word. Otherwise, the current position moves to the first character of the word at the current position. If no preceding word exists on the current line, move to the first character of the last word on the first preceding line that contains a word.

The **b** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line: Set to the line containing the word selected.
Column: Set to the first character of the word selected.
Options: None.

[buffer] [count] c motion

Change a region of text. If only part of a single line is affected, then the last character being changed is marked with a ‘‘`$`’’. Otherwise, the region of text is deleted, and input mode is entered.

If `count` is specified, it is applied to the `motion`.

Line: Set to the last line upon which characters were entered.
Column: Set to the last character entered.
Options: Affected by the **altwerase**, **autoindent**, **beautify**, **showmatch**, **ttywerase** and **wrapmargin** options.

[buffer] [count] d motion

Delete a region of text. If `count` is specified, it is applied to the `motion`.

Line: Set to the line where the region starts.
Column: Set to the first character in the line after the last character in the region. If no such character exists, set to the last character before the region.
Options: None.

[count] e

Move forward `count` end-of-words. Move the cursor forward to the end of a word by repeating the following algorithm: if the current position is the end of a word, move to the last character of the following word. Otherwise, move to the last character of the word at the current position. If no succeeding word exists on the current line, move to the last character of the first word on the next following line that contains a word.

The **e** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line: Set to the line containing the word selected.
Column: Set to the last character of the word selected.
Options: None.

[count] f <character>

Search forward, count times, through the rest of the current line for <character>.

The **f** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line: Unchanged.
Column: Set to the searched-for character.
Options: None.

[count] i

Enter input mode, inserting the text before the cursor. If count is specified, the text input is repeatedly input count - 1 more times.

Line: Set to the last line upon which characters were entered.
Column: Set to the last character entered.
Options: Affected by the **altwerase**, **autoindent**, **beautify**, **showmatch**, **ttywerase** and **wrapmargin** options.

m <character>

Save the current context (line and column) as <character>. The exact position is referred to by ‘‘`<character>’’. The line is referred to by ‘‘'<character>’’.

Historically, <character> was restricted to lower-case letters only, **nvi** permits the use of any character.

Line: Unchanged.
Column: Unchanged.
Options: None.

[count] o

Enter input mode, appending text in a new line under the current line. If count is specified, the text input is repeatedly input count - 1 more times.

Historically, any count specified to the **o** command was ignored.

Line: Set to the last line upon which characters were entered.
Column: Set to the last character entered.
Options: Affected by the **altwerase**, **autoindent**, **beautify**, **showmatch**, **ttywerase** and **wrapmargin** options.

[buffer] p

Append text from a buffer. Text from the buffer (the unnamed buffer by default) is appended after the current column or, if the buffer is line oriented, after the current line.

Line: Set to the first line appended, if the buffer is line oriented, otherwise unchanged.
Column: Set to the first nonblank character of the appended text if the buffer is line oriented, otherwise, the last character of the appended text.
Options: None.

[count] r <character>

Replace characters. The next count characters in the line are replaced with <character>. Replacing characters with <newline> characters results in creating new, empty lines into the file.

If <character> is <escape>, the command is cancelled.

Line: Unchanged unless the replacement character is a `<newline>`, in which case it is set to the current line plus `count - 1`.
Column: Set to the last character replaced, unless the replacement character is a `<newline>`, in which case the cursor is in column 1 of the last line inserted.
Options: None.

[buffer] [count] s

Substitute `count` characters in the current line starting with the current character.

Line: Set to the last line upon which characters were entered.
Column: Set to the last character entered.
Options: Affected by the **altwerase**, **autoindent**, **beautify**, **showmatch**, **ttywerase** and **wrapmargin** options.

[count] t <character>

Search forward, `count` times, through the current line for the character immediately *before* `<character>`.

The **t** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line: Unchanged.
Column: Set to the character *before* the searched-for character.
Options: None.

u

Undo the last change made to the file. If repeated, the **u** command alternates between these two states, and is its own inverse. When used after an insert that inserted text on more than one line, the lines are saved in the numeric buffers.

The `.` command, when used immediately after the **u** command, causes the change log to be rolled forward or backward, depending on the action of the **u** command.

Line: Set to the position of the first line changed, if the reversal affects only one line or represents an addition or change; otherwise, the line preceding the deleted text.
Column: Set to the cursor position before the change was made.
Options: None.

[count] w

Move forward `count` words. Move the cursor forward to the beginning of a word by repeating the following algorithm: if the current position is at the beginning of a word, move to the first character of the next word. If no subsequent word exists on the current line, move to the first character of the first word on the first following line that contains a word.

The **w** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented.

Line: Set to the line containing the word selected.
Column: Set to the first character of the word selected.
Options: None.

[buffer] [count] x

Delete `count` characters. The deletion is at the current character position. If the number of characters to be deleted is greater than or equal to the number of characters to the end of the line, all of the characters from the current cursor position to the end of the line are deleted.

Line: Unchanged.
Column: Unchanged unless the last character in the line is deleted and the cursor is not already on the first character in the line, in which case it is set to the previous character.
Options: None.

[buffer] [count] y motion

Copy (or ‘‘yank’’) a text region specified by the `count` and motion into a buffer. If `count` is specified, it is applied to the `motion`.

Line: Unchanged, unless the region covers more than a single line, in which case it is set to the line where the region starts.
Column: Unchanged, unless the region covers more than a single line, in which case it is set to the character were the region starts.
Options: None.

[count1] z [count2] type

Redraw the screen with a window `count2` lines long, with line `count1` placed as specified by the `type` character. If `count1` is not specified, it defaults to the current line. If `count2` is not specified, it defaults to the current window size.

The following `type` characters may be used:

+ If `count1` is specified, place the line `count1` at the top of the screen. Otherwise, display the screen after the current screen, similarly to the **<control-F>** command.

<carriage-return>
Place the line `count1` at the top of the screen.

. Place the line `count1` in the center of the screen.

– Place the line `count1` at the bottom of the screen.

ˆ If `count1` is specified, place the line that is at the top of the screen when `count1` is at the bottom of the screen, at the bottom of the screen, i.e. display the screen before the screen before `count1`. Otherwise, display the screen before the current screen, similarly to the **<control-B>** command.

Line: Set to `count1` unless `count1` is not specified and the `type` character was either ‘‘ˆ’’ or ‘‘+’’, in which case it is set to the line before the first line on the previous screen or the line after the last line on the previous screen, respectively.
Column: Set to the first nonblank character in the line.
Options: None.

[count] {

Move backward `count` paragraphs.

The **{** command is an absolute movement. The **{** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented, unless the starting character is the first character on its line, in which case it is line oriented.

Line: Set to the line containing the beginning of the previous paragraph.
Column: Set to the first nonblank character in the line.
Options: Affected by the **paragraph** option.

[count] |

Move to a specific *column* position on the current line.

The **|** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented. It is an error to use the **|** command as a motion component and for the cursor not to move.

Line: Unchanged.
Column: Set to the character occupying the column position identified by `count`, if the position exists in the line. If the column length of the current line is less than `count`, the cursor is moved to the last character in the line.
Options: None.

[count] }

Move forward `count` paragraphs.

The **}** command is an absolute movement. The **}** command may be used as the motion component of other **vi** commands, in which case any text copied into a buffer is character oriented, unless the starting character is at or before any nonblank characters in its line, in which case it is line oriented.

Line: Set to the line containing the beginning of the next paragraph.
Column: Set to the first nonblank character in the line.
Options: Affected by the **paragraph** option.

[count] ˜

Reverse the case of the next `count` character(s). This is the historic semantic for the **˜** command and it is only in effect if the **tildeop** option is not set.

Lowercase alphabetic characters are changed to uppercase, and uppercase characters are changed to lowercase. No other characters are affected.

Historically, the **˜** command did not take an associated count, nor did it move past the end of the current line. As it had no associated motion it was difficult to change the case of large blocks of text. In **nvi**, if the cursor is on the last character of a line, and there are more lines in the file, the cursor moves to the next line.

It is not an error to specify a count larger than the number of characters between the cursor and the end of the file.

Line: Set to the line of the character after `count` characters, or, end of file.
Column: Set to the character after `count` characters, or, end-of-file.
Options: Affected by the **tildeop** option.

[count] ˜ motion

Reverse the case of the characters in a text region specified by the `count` and `motion`. Only in effect if the **tildeop** option is set.

Lowercase characters are changed to uppercase, and uppercase characters are changed to lowercase. No other characters are affected.

Line: Set to the line of the character after the last character in the region.
Column: Set to the character after the last character in the region.
Options: Affected by the **tildeop** option.

<interrupt>

Interrupt the current operation. Many of the potentially long-running **vi** commands may be interrupted using the terminal interrupt character. These operations include searches, file reading and writing, filter operations and map character expansion. Interrupts are also enabled when running commands outside of **vi**.

If the `<interrupt>` character is used to interrupt while entering an **ex** command, the command is aborted, the cursor returns to its previous position, and **vi** remains in command mode.

Generally, if the `<interrupt>` character is used to interrupt any operation, any changes made before the interrupt are left in place.

Line: Dependent on the operation being interrupted.
Column: Dependent on the operation being interrupted.
Options: None.

11. Vi Text Input Commands

The following section describes the commands available in the text input mode of the **vi** editor.

Historically, **vi** implementations only permitted the characters inserted on the current line to be erased. In addition, only the `<control-D>` erase character and the ‘‘`0<control-D>`’’ and ‘‘`^<control-D>`’’ erase strings could erase autoindent characters. This implementation permits erasure to continue past the beginning of the current line, and back to where text input mode was entered. In addition, autoindent characters may be erased using the standard erase characters. For the line and word erase characters, reaching the autoindent characters forms a ‘‘soft’’ boundary, denoting the end of the current word or line erase. Repeating the word or line erase key will erase the autoindent characters.

Historically, **vi** always used `<control-H>` and `<control-W>` as character and word erase characters, respectively, regardless of the current terminal settings. This implementation accepts, in addition to these two characters, the current terminal characters for those operations.

<nul>
If the first character of the input is a `<nul>`, the previous input is replayed, as if just entered.

<control-D>
If the previous character on the line was an autoindent character, erase it. Otherwise, if the user is entering the first character in the line, `<control-D>` is ignored. Otherwise, a literal `<control-D>` character is entered.

^<control-D>
If the previous character on the line was an autoindent character, erase all of the autoindent characters on the line. In addition, the autoindent level is reset to 0.

0<control-D>
If the previous character on the line was an autoindent character, erase all of the autoindent characters on the line.

<control-T>
Insert sufficient `<tab>` and `<space>` characters to move the cursor forward to a column immediately after the next column which is an even multiple of the **shiftwidth** option.

Historically, **vi** did not permit the `<control-T>` command to be used unless the cursor was at the first column of a new line or it was preceded only by autoindent characters. **Nvi** permits it to be used at any time during insert mode.

<erase>
<control-H>
Erase the last character.

<literal next>
Quote the next character. The next character will not be mapped (see the **map** command for more information) or interpreted specially. A carat (‘‘^’’) character will be displayed immediately as a placeholder, but will be replaced by the next character.

<escape>
Resolve all text input into the file, and return to command mode.

<line erase>
Erase the current line.

<control-W>
<word erase>
Erase the last word. The definition of word is dependent on the **altwerase** and **ttywerase** options.

<control-X>[0-9A-Fa-f]*
Insert a character with the specified hexadecimal value into the text.

<interrupt>
Interrupt text input mode, returning to command mode. If the `<interrupt>` character is used to interrupt inserting text into the file, it is as if the `<escape>` character was used; all text input up to the interruption is resolved into the file.

12. Ex Addressing

Addressing in **ex** (and when **ex** commands are executed from **vi**) relates to the current line. In general, the current line is the last line affected by a command. The exact effect on the current line is discussed under the description of each command. When the file contains no lines, the current line is zero.

Addresses are constructed by one or more of the following methods:

(1) The address ‘‘`.`’’ refers to the current line.

(2) The address ‘‘`$`’’ refers to the last line of the file.

(3) The address ‘‘`N`’’, where `N` is a positive number, refers to the N-th line of the file.

(4) The address ‘‘`'<character>`’’ or ‘‘`` `<character> ``’’ refers to the line marked with the name `<character>`. (See the **k** or **m** commands for more information on how to mark lines.)

(5) A regular expression (RE) enclosed by slashes (‘‘`/`’’) is an address, and it refers to the first line found by searching forward from the line *after* the current line toward the end of the file, and stopping at the first line containing a string matching the RE. (The trailing slash can be omitted at the end of the command line.)

If no RE is specified, i.e. the pattern is ‘‘`//`’’, the last RE used in any command is used in the search.

If the **extended** option is set, the RE is handled as an extended RE, not a basic RE. If the **wrapscan** option is set, the search wraps around to the beginning of the file and continues up to and including the current line, so that the entire file is searched.

The form ‘‘`\/`’’ is accepted for historic reasons, and is identical to ‘‘`//`’’.

(6) An RE enclosed in question marks (‘‘`?`’’) addresses the first line found by searching backward from the line *preceding* the current line, toward the beginning of the file and stopping at the first line containing a string matching the RE. (The trailing question mark can be omitted at the end of a command line.)

If no RE is specified, i.e. the pattern is ‘‘`??`’’, the last RE used in any command is used in the search.

If the **extended** option is set, the RE is handled as an extended RE, not a basic RE. If the **wrapscan** option is set, the search wraps around from the beginning of the file to the end of the file and continues up to and including the current line, so that the entire file is searched.

The form ‘‘`\?`’’ is accepted for historic reasons, and is identical to ‘‘`??`’’.

(7) An address followed by a plus sign (‘‘+’’) or a minus sign (‘‘-’’) followed by a number is an offset address and refers to the address plus (or minus) the indicated number of lines. If the address is omitted, the addition or subtraction is done with respect to the current line.

(8) An address of ‘‘+’’ or ‘‘–’’ followed by a number is an offset from the current line. For example, ‘‘–5’’ is the same as ‘‘.–5’’.

(9) An address ending with ‘‘+’’ or ‘‘-’’ has 1 added to or subtracted from the address, respectively. As a consequence of this rule and of the previous rule, the address ‘‘–’’ refers to the line preceding the current line. Moreover, trailing ‘‘+’’ and ‘‘–’’ characters have a cumulative effect. For example, ‘‘++–++’’ refers to the current line plus 3.

(10) A percent sign (‘‘%’’) is equivalent to the address range ‘‘1,$’’.

Ex commands require zero, one, or two addresses. It is an error to specify an address to a command which requires zero addresses.

If the user provides more than the expected number of addresses to any **ex** command, the first addresses specified are discarded. For example, ‘‘1,2,3,5’’print prints lines 3 through 5, because the **print** command only takes two addresses.

The addresses in a range are separated from each other by a comma (‘‘,’’) or a semicolon (‘‘;’’). In the latter case, the current line (‘‘.’’) is set to the first address, and only then is the second address calculated. This feature can be used to determine the starting line for forward and backward searches (see rules (5) and (6) above). The second address of any two-address sequence corresponds to a line that follows, in the file, the line corresponding to the first address. The first address must be less than or equal to the second address. The first address must be greater than or equal to the first line of the file, and the last address must be less than or equal to the last line of the file.

13. Ex Description

The following words have special meanings for **ex** commands.

<eof>

The end-of-file character is used to scroll the screen in the **ex** editor. This character is normally `<control-D>`, however, whatever character is set for the current terminal is used.

line

A single-line address, given in any of the forms described in the section entitled ‘‘**Ex Addressing**’’. The default for `line` is the current line.

range

A line, or a pair of line addresses, separated by a comma or semicolon. (See the section entitled ‘‘**Ex Addressing**’’ for more information.) The default for range is the current line *only*, i.e. ‘‘.,.’’. A percent sign (‘‘%’’) stands for the range ‘‘1,$’’. The starting address must be less than, or equal to, the ending address.

count

A positive integer, specifying the number of lines to be affected by the command; the default is 1. Generally, a count past the end-of-file may be specified, e.g. the command ‘‘p 3000’’ in a 10 line file is acceptable, and will print from the current line through the last line in the file.

flags

One or more of the characters ‘‘#’’, ‘‘p’’, and ‘‘l’’. When a command that accepts these flags completes, the addressed line(s) are written out as if by the corresponding **#**, **l** or **p** commands. In addition, any number of ‘‘+’’ or ‘‘–’’ characters can be specified before, after, or during the flags, in which case the line written is not necessarily the one affected by the command, but rather the line addressed by the offset address specified. The default for `flags` is none.

file

A pattern used to derive a pathname; the default is the current file. File names are subjected to normal *sh*(1) word expansions.

Anywhere a file name is specified, it is also possible to use the special string ‘‘`/tmp`’’. This will be replaced with a temporary file name which can be used for temporary work, e.g. ‘‘`:e /tmp`’’ creates and edits a new file.

If both a count and a range are specified for commands that use either, the starting line for the command is the *last* line addressed by the range, and `count`- subsequent lines are affected by the command, e.g. the command ‘‘`2,3p4`’’ prints out lines 3, 4, 5 and 6.

When only a line or range is specified, with no command, the implied command is either a **list**, **number** or **print** command. The command used is the most recent of the three commands to have been used (including any use as a flag). If none of these commands have been used before, the **print** command is the implied command. When no range or count is specified and the command line is a blank line, the current line is incremented by 1 and then the current line is displayed.

Zero or more whitespace characters may precede or follow the addresses, count, flags, or command name. Any object following a command name (such as buffer, file, etc.), that begins with an alphabetic character, should be separated from the command name by at least one whitespace character.

Any character, including `<carriage-return>`, ‘‘`%`’’ and ‘‘`#`’’ retain their literal value when preceded by a backslash.

14. Ex Commands

The following section describes the commands available in the **ex** editor. In each entry below, the tag line is a usage synopsis for the command.

Each command can be entered as the abbreviation (those characters in the synopsis command word preceding the ‘‘[’’ character), the full command (all characters shown for the command word, omitting the ‘‘[’’ and ‘‘]’’ characters), or any leading subset of the full command down to the abbreviation. For example, the args command (shown as ‘‘`ar[gs]`’’ in the synopsis) can be entered as ‘‘`ar`’’, ‘‘`arg`’’ or ‘‘`args`’’.

Each **ex** command described below notes the new current line after it is executed, as well as any options that affect the command.

A comment. Command lines beginning with the double-quote character (‘‘`"`’’) are ignored. This permits comments in editor scripts and startup files.

<end-of-file>

Scroll the screen. Write the next N lines, where N is the value of the **scroll** option. The command is the end-of-file terminal character, which may be different on different terminals. Traditionally, it is the `<control-D>` key.

Historically, the **eof** command ignored any preceding count, and the `<end-of-file>` character was ignored unless it was entered as the first character of the command. This implementation treats it as a command *only* if entered as the first character of the command line, and otherwise treats it as any other character.

Line: Set to the last line written.
Options: None.

! argument(s)
[range]! argument(s)

Execute a shell command, or filter lines through a shell command. In the first synopsis, the remainder of the line after the ‘‘`!`’’ character is passed to the program named by the **shell** option, as a single argument.

Within the rest of the line, ‘‘%’’ and ‘‘#’’ are expanded into the current and alternate pathnames, respectively. The character ‘‘!’’ is expanded with the command text of the previous **!** command. (Therefore, the command **!!** repeats the previous **!** command.) The special meanings of ‘‘%’’, ‘‘#’’, and ‘‘!’’ can be overridden by escaping them with a backslash. If no **!** or **:!** command has yet been executed, it is an error to use an unescaped ‘‘!’’ character. The **!** command does *not* do shell expansion on the strings provided as arguments. If any of the above expansions change the command the user entered, the command is redisplayed at the bottom of the screen.

Ex then executes the program named by the **shell** option, with a **−c** flag followed by the arguments (which are bundled into a single argument).

The **!** command is permitted in an empty file.

If the file has been modified since it was last completely written, the command will warn you.

A single ‘‘!’’ character is displayed when the command completes.

In the second form of the **!** command, the remainder of the line after the ‘‘!’’ is passed to the program named by the **shell** option, as described above. The specified lines are passed to the program as standard input, and the standard and standard error output of the program replace the original lines.

Line: Unchanged if no range was specified, otherwise set to the first line of the range.
Options: Affected by the **autowrite** and **writeany** options.

[range] nu[mber] [count] [flags]
[range] # [count] [flags]

Display the selected lines, each preceded with its line number.

The line number format is ‘‘%6d’’, followed by two spaces.

Line: Set to the last line displayed.
Options: None.

@ buffer
*** buffer**

Execute a buffer. Each line in the named buffer is executed as an **ex** command. If no buffer is specified, or if the specified buffer is ‘‘@’’ or ‘‘*’’, the last buffer executed is used.

[range] <[< ...] [count] [flags]

Shift lines left or right. The specified lines are shifted to the left (for the < command) or right (for the > command), by the number of columns specified by the **shiftwidth** option. Only leading whitespace characters are deleted when shifting left; once the first column of the line contains a nonblank character, the **shift** command will succeed, but the line will not be modified.

If the command character < or > is repeated more than once, the command is repeated once for each additional command character.

Line: If the current line is set to one of the lines that are affected by the command, it is unchanged. Otherwise, it is set to the first nonblank character of the lowest numbered line shifted.
Options: Affected by the **shiftwidth** option.

[line] = [flags]

Display the line number. Display the line number of `line` (which defaults to the last line in the file).

Line: Unchanged.
Options: None.

[range] >[> ...] [count] [flags]

Shift right. The specified lines are shifted to the right by the number of columns specified by the **shiftwidth** option, by inserting tab and space characters. Empty lines are not changed.

If the command character ‘‘>’’ is repeated more than once, the command is repeated once for each additional command character.

Line: Set to the last line modified by the command.
Options: None.

ab[brev] lhs rhs

Add an abbreviation to the current abbreviation list. In **vi**, if `lhs` is entered such that it is preceded and followed by characters that cannot be part of a word, it is replaced by the string `rhs`.

Line: Unchanged.
Options: None.

[line] a[ppend][!]

The input text is appended to the specified line. If line 0 is specified, the text is inserted at the beginning of the file. Set to the last line input. If no lines are input, then set to `line`, or to the first line of the file if a `line` of 0 was specified. Following the command name with a ‘‘!’’ character causes the **autoindent** option to be toggled for the duration of the command.

Line: Unchanged.
Options: Affected by the **altwerase**, **autoindent**, **beautify**, **showmatch**, **ttywerase** and **wrapmargin** options.

ar[gs]

Display the argument list. The current argument is displayed inside of ‘‘[’’ and ‘‘]’’ characters. The argument list is the list of operands specified on startup, which can be replaced using the **next** command.

Line: Unchanged.
Options: None.

bg

Vi mode only. Background the current screen.

Line: Set to the current line when the screen was last edited.
Options: None.

[range] c[hange][!] [count]

Replace the lines with input text. Following the command name with a ‘‘!’’ character causes the **autoindent** option to be toggled for the duration of the command.

Line: Set to the last line input, or, if no lines were input, set to the line before the target line, or to the first line of the file if there are no lines preceding the target line.
Options: Affected by the **altwerase**, **autoindent**, **beautify**, **showmatch**, **ttywerase** and **wrapmargin** options.

chd[ir][!] [directory]
cd[!] [directory]

Change the current working directory. The `directory` argument is subjected to *sh*(1) word expansions. When invoked with no directory argument and the `HOME` environment variable is set, the directory named by the `HOME` environment variable becomes the new current directory. Otherwise, the new current directory becomes the directory returned by the *getpwent*(3) routine.

The **chdir** command will fail if the file has been modified since the last complete write of the file. You can override this check by appending a ‘‘`!`’’ character to the command.

Line:	Unchanged.
Options:	Affected by the **cdpath** option.

[range] co[py] line [flags]
[range] t line [flags]

Copy the specified lines (range) after the destination line. Line 0 may be specified to insert the lines at the beginning of the file.

Line:	Unchanged.
Options:	None.

[range] d[elete] [buffer] [count] [flags]

Delete the lines from the file. The deleted text is saved in the specified buffer, or, if no buffer is specified, in the unnamed buffer. If the command name is followed by a letter that could be interpreted as either a buffer name or a flag value (because neither a `count` or `flags` values were given), **ex** treats the letter as a `flags` value if the letter immediately follows the command name, without any whitespace separation. If the letter is preceded by whitespace characters, it treats it as a buffer name.

Line:	Set to the line following the deleted lines, or to the last line if the deleted lines were at the end.
Options:	None.

di[splay] b[uffers] | s[creens] | t[ags]

Display buffers, screens or tags. The **display** command takes one of three additional arguments, which are as follows:

b[uffers]	Display all buffers (including named, unnamed, and numeric) that contain text.
s[creens]	Display the file names of all background screens.
t[ags]	Display the tags stack.

Line:	Unchanged.
Options:	None.

e[dit][!] [+cmd] [file]
ex[!] [+cmd] [file]

Edit a different file. If the current buffer has been modified since the last complete write, the command will fail. You can override this by appending a ‘‘`!`’’ character to the command name.

If the ‘‘`+cmd`’’ option is specified, that **ex** command will be executed in the new file. Any **ex** command may be used, although the most common use of this feature is to specify a line number or search pattern to set the initial location in the new file.

Line:	If you have previously edited the file, the current line will be set to your last position in the file. If that position does not exist, or you have not previously edited the

file, the current line will be set to the first line of the file if you are in **vi** mode, and the last line of the file if you are in **ex**.
Options: Affected by the **autowrite** and **writeany** options.

exu[sage] [command]

Display usage for an **ex** command. If `command` is specified, a usage statement for that command is displayed. Otherwise, usage statements for all **ex** commands are displayed.

Line: Unchanged.
Options: None.

f[ile] [file]

Display and optionally change the file name. If a file name is specified, the current pathname is changed to the specified name. The current pathname, the number of lines, and the current position in the file are displayed.

Line: Unchanged.
Options: None.

fg [name]

Vi mode only. Foreground the specified screen. Swap the current screen with the specified backgrounded screen. If no screen is specified, the first background screen is foregrounded.

Line: Set to the current line when the screen was last edited.
Options: None.

[range] g[lobal] /pattern/ [commands]
[range] v /pattern/ [commands]

Apply commands to lines matching (or not matching) a pattern. The lines within the given range that match (‘‘`g[lobal]`’’), or do not match (‘‘`v`’’) the given pattern are selected. Then, the specified **ex** command(s) are executed with the current line (‘‘`.`’’) set to each selected line. If no range is specified, the entire file is searched for matching, or not matching, lines.

Multiple commands can be specified, one per line, by escaping each `<newline>` character with a backslash, or by separating commands with a ‘‘`|`’’ character. If no commands are specified, the command defaults to the **print** command.

For the **append**, **change** and **insert** commands, the input text must be part of the global command line. In this case, the terminating period can be omitted if it ends the commands.

The **visual** command may also be specified as one of the **ex** commands. In this mode, input is taken from the terminal. Entering a **Q** command in **vi** mode causes the next line matching the pattern to be selected and **vi** to be reentered, until the list is exhausted.

The **global**, **v** and **undo** commands cannot be used as part of these commands.

The editor options **autoprint**, **autoindent**, and **report** are turned off for the duration of the **global** and **v** commands.

Line: The last line modified.
Options: None.

he[lp]

Display a help message.

Line: Unchanged.
Options: None.

[line] i[nsert][!]

The input text is inserted before the specified line. Following the command name with a ‘‘!’’ character causes the **autoindent** option setting to be toggled for the duration of this command.

Line: Set to the last line input; if no lines were input, set to the line before the target line, or to the first line of the file if there are no lines preceding the target line.
Options: Affected by the **altwerase**, **autoindent**, **beautify**, **showmatch**, **ttywerase** and **wrapmargin** options.

[range] j[oin][!] [count] [flags]

Join lines of text together.

A `count` specified to the command specifies that the last line of the `range` plus `count` subsequent lines will be joined. (Note, this differs by one from the general rule where only `count`- subsequent lines are affected.)

If the current line ends with a whitespace character, all whitespace is stripped from the next line. Otherwise, if the next line starts with a open parenthesis (‘‘`(`’’), do nothing. Otherwise, if the current line ends with a question mark (‘‘`?`’’), period (‘‘`.`’’) or exclamation point (‘‘`!`’’), insert two spaces. Otherwise, insert a single space.

Appending a ‘‘`!`’’ character to the command name causes a simpler join with no white-space processing.

Line: Unchanged.
Options: None.

[range] l[ist] [count] [flags]

Display the lines unambiguously. Tabs are displayed as ‘‘`^I`’’, and the end of the line is marked with a ‘‘`$`’’ character.

Line: Set to the last line displayed.
Options: None.

map[!] [lhs rhs]

Define or display maps (for **vi** only).

If ‘‘`lhs`’’ and ‘‘`rhs`’’ are not specified, the current set of command mode maps are displayed. If a ‘‘`!`’’ character is appended to to the command, the text input mode maps are displayed.

Otherwise, when the ‘‘`lhs`’’ character sequence is entered in **vi**, the action is as if the corresponding ‘‘`rhs`’’ had been entered. If a ‘‘`!`’’ character is appended to the command name, the mapping is effective during text input mode, otherwise, it is effective during command mode. This allows ‘‘`lhs`’’ to have two different macro definitions at the same time: one for command mode and one for input mode.

Whitespace characters require escaping with a `<literal`next> character to be entered in the `lhs` string in visual mode.

Normally, keys in the `rhs` string are remapped (see the **remap** option), and it is possible to create infinite loops. However, keys which map to themselves are not further remapped, regardless of the setting of the **remap** option. For example, the command ‘‘`:map n nz.`’’ maps the ‘‘`n`’’ key to the **n** and **z** commands.

To exit an infinitely looping map, use the terminal <interrupt> character.

Line: Unchanged.
Options: None.

[line] ma[rk] <character>
[line] k <character>

Mark the line with the mark `<character>`. The expressions ‘‘`'<character>`’’ and ‘‘`` `<character> ``’’ can then be used as an address in any command that uses one.

Line: Unchanged.
Options: None.

[range] m[ove] line

Move the specified lines after the target line. A target line of 0 places the lines at the beginning of the file.

Line: Set to the first of the moved lines.
Options: None.

mk[exrc][!] file

Write the abbreviations, editor options and maps to the specified file. Information is written in a form which can later be read back in using the **ex source** command. If `file` already exists, the **mkexrc** command will fail. This check can be overridden by appending a ‘‘!’’ character to the command.

Line: Unchanged.
Options: None.

n[ext][!] [file ...]

Edit the next file from the argument list. The **next** command will fail if the file has been modified since the last complete write. This check can be overridden by appending the ‘‘!’’ character to the command name. The argument list can optionally be replaced by specifying a new one as arguments to this command. In this case, editing starts with the first file on the new list.

Line: Set as described for the **edit** command.
Options: Affected by the options **autowrite** and **writeany**.

[line] o[pen] /pattern/ [flags]

Enter open mode. Open mode is the same as being in **vi**, but with a one-line window. All the standard **vi** commands are available. If a match is found for the optional RE argument, the cursor is set to the start of the matching pattern.

This command is not yet implemented.

Line: Unchanged, unless the optional RE is specified, in which case it is set to the line where the matching pattern is found.
Options: Affected by the **open** option.

pre[serve]

Save the file in a form that can later be recovered using the **ex −r** option. When the file is preserved, an email message is sent to the user.

Line: Unchanged.
Options: None.

prev[ious][!]

Edit the previous file from the argument list. The **previous** command will fail if the file has been modified since the last complete write. This check can be overridden by appending the ``!'' character to the command name.

Line: Set as described for the **edit** command.
Options: Affected by the options **autowrite** and **writeany**. None.

[range] p[rint] [count] [flags]

Display the specified lines.

Line: Set to the last line displayed.
Options: None.

[line] pu[t] [buffer]

Append buffer contents to the current line. If a buffer is specified, its contents are appended to the line, otherwise, the contents of the unnamed buffer are used.

Line: Set to the line after the current line.
Options: None.

q[uit][!]

End the editing session. If the file has been modified since the last complete write, the **quit** command will fail. This check may be overridden by appending a ``!'' character to the command.

If there are more files to edit, the **quit** command will fail. Appending a ``!'' character to the command name or entering two **quit** commands (i.e. **wq**, **quit**, **xit** or **ZZ**) in a row) will override this check and the editor will exit.

Line: Unchanged.
Options: None.

[line] r[ead][!] [file]

Read a file. A copy of the specified file is appended to the line. If `line` is 0, the copy is inserted at the beginning of the file. If no file is specified, the current file is read; if there is no current file, then `file` becomes the current file. If there is no current file and no `file` is specified, then the **read** command will fail.

If `file` is preceded by a ``!'' character, `file` is treated as if it were a shell command, and passed to the program named by the `SHELL` environment variable. The standard and standard error outputs of that command are read into the file after the specified line. The special meaning of the ``!'' character can be overridden by escaping it with a backslash (``\'') character.

Line: When executed from **ex**, the current line is set to the last line read. When executed from **vi**, the current line is set to the first line read.
Options: None.

rec[over] file

Recover `file` if it was previously saved. If no saved file by that name exists, the **recover** command behaves similarly to the **edit** command.

Line: Set as described for the **edit** command.
Options: None.

res[ize] [+|-]size

Vi mode only. Grow or shrink the current screen. If `size` is a positive, signed number, the current screen is grown by that many lines. If `size` is a negative, signed number, the current screen is shrunk by that many lines. If `size` is not signed, the current screen is set to the specified `size`. Applicable only to split screens.

Line: Unchanged.
Options: None.

rew[ind][!]

Rewind the argument list. If the current file has been modified since the last complete write, the **rewind** command will fail. This check may be overridden by appending the ‘‘`!`’’ character to the command.

Otherwise, the current file is set to the first file in the argument list.

Line: Set as described for the **edit** command.
Options: Affected by the **autowrite** and **writeany** options.

se[t] [option[=[value]] ...] [nooption ...] [option? ...] [all]

Display or set editor options. When no arguments are specified, the editor option **term**, and any editor options whose values have been changed from the default settings are displayed. If the argument `all` is specified, the values of all of editor options are displayed.

Specifying an option name followed by the character ‘‘`?`’’ causes the current value of that option to be displayed. The ‘‘`?`’’ can be separated from the option name by whitespace characters. The ‘‘`?`’’ is necessary only for Boolean valued options. Boolean options can be given values by the form ‘‘`set option`’’ to turn them on, or ‘‘`set nooption`’’ to turn them off. String and numeric options can be assigned by the form ‘‘`set option=value`’’. Any whitespace characters in strings can be included literally by preceding each with a backslash. More than one option can be set or listed by a single set command, by specifying multiple arguments, each separated from the next by whitespace characters.

Line: Unchanged.
Options: None.

sh[ell]

Run a shell program. The program named by the **shell** option is run with a **−i** (for interactive) flag. Editing is resumed when that program exits.

Line: Unchanged.
Options: None.

so[urce] file

Read and execute **ex** commands from a file. **Source** commands may be nested.

Line: Unchanged.
Options: None.

sp[lit] [file ...]

Vi mode only. Split the screen. The current screen is split into two screens, of approximately equal size. If the cursor is in the lower half of the screen, the screen will split up, i.e. the new screen will be above the old one. If the cursor is in the upper half of the screen, the new screen will be below the old one.

If `file` is specified, the new screen is editing that file, otherwise, both screens are editing the same file, and changes in each will be be reflected in the other. The argument list for the new screen consists of the list of files specified as arguments to this command, or, the current pathname if no files are specified.

Line: If `file` is specified, set as for the **edit** command, otherwise unchanged.
Options: None.

[range] s[ubstitute] [/pattern/replace/] [options] [count] [flags]
[range] & [options] [count] [flags]
[range] ˜ [options] [count] [flags]

Make substitutions. Replace the first instance of `pattern` with the string `replace` on the specified line(s). If the ‘‘`/pattern/repl/`’’ argument is not specified, the ‘‘`/pattern/repl/`’’ from the previous **substitute** command is used.

If `options` includes the letter ‘‘`c`’’ (confirm), you will be prompted for confirmation before each replacement is done. An affirmative response (in English, a ‘‘`y`’’ character) causes the replacement to be made. A quit response (in English, a ‘‘`q`’’ character) causes the **substitute** command to be terminated. Any other response causes the replacement not to be made, and the **substitute** command continues. If `options` includes the letter ‘‘`g`’’ (global), all nonoverlapping instances of `pattern` in the line are replaced.

The **&** version of the command is the same as not specifying a pattern or replacement string to the **substitute** command, and the ‘‘&’’ is replaced by the pattern and replacement information from the previous substitute command.

The **˜** version of the command is the same as **&** and **s**, except that the search pattern used is the last RE used in *any* command, not necessarily the one used in the last **substitute** command.

For example, in the sequence

```
s/red/blue/
/green
~
```

the ‘‘˜’’ is equivalent to ‘‘`s/green/blue/`’’.

The **substitute** command may be interrupted, using the terminal interrupt character. All substitutions completed before the interrupt are retained.

Line: Set to the last line upon which a substitution was made.
Options: None.

su[spend][!]
st[op][!]
<control-Z>

Suspend the edit session. Appending a ‘‘`!`’’ character to these commands turns off the **autowrite** option for the command.

Line: Unchanged.
Options: Affected by the **autowrite** option.

ta[g][!] tagstring

Edit the file containing the specified tag. Search for the tagstring, which can be in a different file. If the tag is in a different file, then the new file is edited. If the current file has been modified since the last complete write, the **tag** command will fail. This check can be overridden by appending the ‘‘`!`’’ character to the command name.

The **tag** command searches for `tagstring` in the tags file(s) specified by the option. (See *ctags*(1) for more information on tags files.)

Line: Set to the line indicated by the tag.

Options: Affected by the **autowrite**, **taglength**, **tags** and **writeany** options.

tagp[op][!] [file | number]

Pop to the specified tag in the tags stack. If neither `file` or `number` is specified, the **tagpop** command pops to the most recent entry on the tags stack. If `file` or `number` is specified, the **tagpop** command pops to the most recent entry in the tags stack for that file, or numbered entry in the tags stack, respectively. (See the **display** command for information on displaying the tags stack.)

If the file has been modified since the last complete write, the **tagpop** command will fail. This check may be overridden by appending a ‘‘`!`’’ character to the command name.

Line: Set to the line indicated by the tag.
Options: Affected by the **autowrite**, and **writeany** options.

tagt[op][!]

Pop to the least recent tag on the tags stack, clearing the tags stack.

If the file has been modified since the last complete write, the **tagpop** command will fail. This check may be overridden by appending a ‘‘`!`’’ character to the command name.

Line: Set to the line indicated by the tag.
Options: Affected by the **autowrite**, and **writeany** options.

una[bbrev] lhs

Delete an abbreviation. Delete `lhs` from the current list of abbreviations.

Line: Unchanged.
Options: None.

u[ndo]

Undo the last change made to the file. Changes made by **global**, **v**, **visual** and map sequences are considered a single command. If repeated, the **u** command alternates between these two states, and is its own inverse.

Line: Set to the last line modified by the command.
Options: None.

unm[ap][!] lhs

Unmap a mapped string. Delete the command mode map definition for `lhs`. If a ‘‘`!`’’ character is appended to the command name, delete the text input mode map definition instead.

Line: Unchanged.
Options: None.

ve[rsion]

Display the version of the **ex/vi** editor.

[line] vi[sual] [type] [count] [flags]

Ex mode only. Enter **vi**. The `type` is optional, and can be ‘‘−’’, ‘‘+’’ or ‘‘ˆ’’, as in the **ex z** command, to specify the the position of the specified line in the screen window. (The default is to place the line at the top of the screen window.) A `count` specifies the number of lines that will initially be displayed. (The default is the value of the **window** editor option.)

Line: Unchanged unless `line` is specified, in which case it is set to that line.
Options: None.

vi[sual][!] [+cmd] [file]

Vi mode only. Edit a new file. Identical to the ‘‘`edit[!] [+cmd] [file]`’’ command.

viu[sage] [command]

Display usage for a **vi** command. If `command` is specified, a usage statement for that command is displayed. Otherwise, usage statements for all **vi** commands are displayed.

Line: Unchanged.
Options: None.

[range] w[rite][!] [>>] [file]
[range] w[rite] [!] [file]
[range] wn[!] [>>] [file]
[range] wq[!] [>>] [file]

Write the file. The specified lines (the entire file, if no range is given) is written to `file`. If `file` is not specified, the current pathname is used. If `file` is specified, and it exists, or if the current pathname was set using the **file** command, and the file already exists, these commands will fail. Appending a ‘‘`!`’’ character to the command name will override this check and the write will be attempted, regardless.

Specifying the optional ‘‘`>>`’’ string will cause the write to be appended to the file, in which case no tests are made for the file already existing.

If the file is preceded by a ‘‘`!`’’ character, the program named in the SHELL environment variable is invoked with file as its second argument, and the specified lines are passed as standard input to that command. The ‘‘`!`’’ in this usage must be separated from command name by at least one whitespace character. The special meaning of the ‘‘`!`’’ may be overridden by escaping it with a backslash (‘‘`\`’’) character.

The **wq** version of the write command will exit the editor after writing the file, if there are no further files to edit. Appending a ‘‘`!`’’ character to the command name or entering two ‘‘quit’’ commands (i.e. **wq**, **quit**, **xit** or **ZZ**) in a row) will override this check and the editor will exit, ignoring any files that have not yet been edited.

The **wn** version of the write command will move to the next file after writing the file, unless the write fails.

Line: Unchanged.
Options: Affected by the **readonly** and **writeany** options.

[range] x[it][!] [file]

Write the file if it has been modified. The specified lines are written to `file`, if the file has been modified since the last complete write to any file. If no `range` is specified, the entire file is written.

The **xit** command will exit the editor after writing the file, if there are no further files to edit. Appending a ‘‘`!`’’ character to the command name or entering two ‘‘quit’’ commands (i.e. **wq**, **quit**, **xit** or **ZZ**) in a row) will override this check and the editor will exit, ignoring any files that have not yet been edited.

Line: Unchanged.
Options: Affected by the **readonly** and **writeany** options.

[range] ya[nk] [buffer] [count]

Copy the specified lines to a buffer. If no buffer is specified, the unnamed buffer is used.

Line: Unchanged.
Options: None.

[line] z [type] [count] [flags]

Adjust the window. If no `type` is specified, then `count` lines following the specified line are displayed. The default `count` is the value of the **window** option. The `type` argument changes the position at which `line` is displayed on the screen by changing the number of lines displayed before and after `line`. The following `type` characters may be used:

–	Place the line at the bottom of the screen.
+	Place the line at the top of the screen.
.	Place the line in the middle of the screen.
ˆ	Write out count lines starting `count * 2` lines before `line`; the net effect of this is that a ‘‘`zˆ`’’ command following a **z** command writes the previous page.
=	Center `line` on the screen with a line of hyphens displayed immediately before and after it. The number of preceding and following lines of text displayed are reduced to account for those lines.

Line: Set to the last line displayed, with the exception of the `type`, where the current line is set to the line specified by the command.
Options: Affected by the option.

15. Set Options

There are a large number of options that may be set (or unset) to change the editor's behavior. This section describes the options, their abbreviations and their default values.

In each entry below, the first part of the tag line is the full name of the option, followed by any equivalent abbreviations. (Regardless of the abbreviations, it is only necessary to use the minimum number of characters necessary to distinguish an abbreviation from all other commands for it to be accepted, in **nex**/**nvi**. Historically, only the full name and the official abbreviations were accepted by **ex**/**vi**. Using full names in your startup files and environmental variables will probably make them more portable.) The part in square brackets is the default value of the option. Most of the options are boolean, i.e. they are either on or off, and do not have an associated value.

Options apply to both **ex** and **vi** modes, unless otherwise specified.

For information on modifying the options or to display the options and their current values, see the ‘‘set’’ command in the section entitled ‘‘**Ex Commands**’’.

altwerase [off]

Vi only. Change how **vi** does word erase during text input. When this option is set, text is broken up into three classes: alphabetic, numeric and underscore characters, other nonblank characters, and blank characters. Changing from one class to another marks the end of a word. In addition, the class of the first character erased is ignored (which is exactly what you want when erasing pathname components).

autoindent, ai [off]

If this option is set, whenever you create a new line (using the **vi A**, **a**, **C**, **c**, **I**, **i**, **O**, **o**, **R**, **r**, **S**, and **s** commands, or the **ex append**, **change**, and **insert** commands) the new line is automatically indented to align the cursor with the first nonblank character of the line from which you created it. Lines are indented using tab characters to the extent possible (based on the value of the **tabstop** option) and then using space characters as necessary. For commands inserting text into the middle of a line, any blank characters to the right of the cursor are discarded, and the first nonblank character to the right of the cursor is aligned as described above.

The indent characters are themselves somewhat special. If you do not enter more characters on the new line before moving to another line, or entering `<escape>`, the indent character will be deleted and the line will be empty. For example, if you enter `<carriage-return>`

twice in succession, the line created by the first `<carriage-return>` will not have any characters in it, regardless of the indentation of the previous or subsequent line.

Indent characters also require that you enter additional erase characters to delete them. For example, if you have an indented line, containing only blanks, the first `<word-erase>` character you enter will erase up to end of the indent characters, and the second will erase back to the beginning of the line. (Historically, only the **<control-D>** key would erase the indent characters. Both the **<control-D>** key and the usual erase keys work in **nvi**.) In addition, if the cursor is positioned at the end of the indent characters, the keys ‘‘`0<control-D>`’’ will erase all of the indent characters for the current line, resetting the indentation level to 0. Similarly, the keys ‘‘`^<control-D>`’’ will erase all of the indent characters for the current line, leaving the indentation level for future created lines unaffected.

Finally, if the **autoindent** option is set, the **S** and **cc** commands change from the first nonblank of the line to the end of the line, instead of from the beginning of the line to the end of the line.

autoprint, ap [off]

Ex only. Cause the current line to be automatically displayed after the **ex** commands <, >, **copy**, **delete**, **join**, **move**, **put**, **t**, **Undo**, and **undo**. This automatic display is suppressed during **global** and **vglobal** commands, and for any command where optional flags are used to explicitly display the line.

autowrite, aw [off]

If this option is set, the **vi !**, **^^**, **^]** and **<control-Z>** commands, and the **ex edit**, **next**, **rewind**, **stop**, **suspend**, **tag**, **tagpop**, and **tagtop** commands automatically write the current file back to the current file name if it has been modified since it was last written. If the write fails, the command fails and goes no further.

Appending the optional force flag character ‘‘`!`’’ to the **ex** commands **next**, **rewind**, **stop**, **suspend**, **tag**, **tagpop**, and **tagtop** stops the automatic write from being attempted.

(Historically, the **next** command ignored the optional force flag.) Note, the **ex** commands **edit**, **quit**, **shell**, and **xit** are *not* affected by the **autowrite** option.

beautify, bf [off]

If this option is set, all control characters that are not currently being specially interpreted, other than `<tab>`, `<newline>`, and `<form-feed>`, are discarded from commands read in by **ex** from command files, and from input text entered to **vi** (either into the file or to the colon command line). Text files read by **ex**/**vi** are *not* affected by the **beautify** option.

cdpath [environment variable CDPATH, or current directory]

This option is used to specify a colon separated list of directories which are used as path prefixes for any relative path names used as arguments for the **cd** command. The value of this option defaults to the value of the environmental variable `CDPATH` if it is set, otherwise to the current directory. For compatibility with the POSIX 1003.2 shell, the **cd** command does *not* check the current directory as a path prefix for relative path names unless it is explicitly specified. It may be so specified by entering an empty string or a ‘‘`.`’’ character into the `CDPATH` variable or the option value.

columns, co [80]

The number of columns in the screen. Setting this option causes **ex**/**vi** to set (or reset) the environmental variable `COLUMNS`. See the section entitled ‘‘**Sizing the Screen**’’ more information.

comment [off]

Vi only. If the first non-empty line of the file begins with the string ‘‘`/*`’’, this option causes **vi** to skip to the end of that C-language comment (probably a terribly boring legal notice) before displaying the file.

directory, dir [environment variable TMPDIR, or /tmp]
The directory where temporary files are created. The environmental variable `TMPDIR` is used as the default value if it exists, otherwise `/tmp` is used.

edcompatible, ed [off]
Remember the values of the ‘‘c’’ and ‘‘g’’ suffices to the **substitute** commands, instead of initializing them as unset for each new command. Specifying pattern and replacement strings to the **substitute** command unsets the ‘‘c’’ and ‘‘g’’ suffices as well.

errorbells, eb [off]
Ex only. **Ex** error messages are normally presented in inverse video. If that is not possible for the terminal, setting this option causes error messages to be announced by ringing the terminal bell.

exrc, ex [off]
If this option is turned off in the system or $HOME startup files, the local startup files are never read (unless they are the same as the system or $HOME startup files). Turning it on has no effect, i.e. the normal checks for local startup files are performed, regardless. See the section entitled ‘‘**Startup Information**’’ for more information.

extended [off]
This option causes all regular expressions to be treated as POSIX 1003.2 Extended Regular Expressions (which are similar to historic *egrep*(1) style expressions).

flash [on]
This option causes the screen to flash instead of beeping the keyboard, on error, if the terminal has the capability.

hardtabs, ht [8]
This option defines the spacing between hardware tab settings, i.e. the tab expansion done by the operating system and/or the terminal itself. As **nex**/**nvi** never writes `<tab>` characters to the terminal, unlike historic versions of **ex**/**vi**, this option does not currently have any affect.

ignorecase, ic [off]
This option causes regular expressions, both in **ex** commands and in searches, to be evaluated in a case-insensitive manner.

keytime [6]
The 10th’s of a second **ex**/**vi** waits for a subsequent key to complete a key mapping.

leftright [off]
Vi only. This option causes the screen to be scrolled left-right to view lines longer than the screen, instead of the traditional **vi** screen interface which folds long lines at the right-hand margin of the terminal.

lines, li [24]
Vi only. The number of lines in the screen. Setting this option causes **ex**/**vi** to set (or reset) the environmental variable `LINES`. See the section entitled ‘‘**Sizing the Screen**’’ for more information.

lisp [off]
Vi only. This option changes the behavior of the **vi (**, **)**, **{**, **}**, **[[** and **]]** commands to match the Lisp language. Also, the **autoindent** option’s behavior is changed to be appropriate for Lisp.

This option is not yet implemented.

list [off]

This option causes lines to be displayed in an unambiguous fashion. Specifically, tabs are displayed as control characters, i.e. ‘‘`^I`’’, and the ends of lines are marked with a ‘‘`$`’’ character.

magic [on]

This option is on by default. Turning the **magic** option off causes all regular expression characters except for ‘‘`^`’’ and ‘‘`$`’’, to be treated as ordinary characters. To re-enable characters individually, when the **magic** option is off, precede them with a backslash ‘‘`\`’’ character. See the section entitled ‘‘**Regular Expressions and Replacement Strings**’’ for more information.

matchtime [7]

Vi only. The 10th’s of a second **ex**/**vi** pauses on the matching character when the **showmatch** option is set.

mesg [on]

This option allows other users to contact you using the *talk*(1) and *write*(1) utilities, while you are editing. **Ex**/**vi** does not turn message on, i.e. if messages were turned off when the editor was invoked, they will stay turned off. This option only permits you to disallow messages for the edit session. See the *mesg*(1) utility for more information.

modelines, modeline [off]

If the **modelines** option is set, **ex**/**vi** has historically scanned the first and last five lines of each file as it is read for editing, looking for any **ex** commands that have been placed in those lines. After the startup information has been processed, and before the user starts editing the file, any commands embedded in the file are executed.

Commands were recognized by the letters ‘‘e’’ or ‘‘v’’ followed by ‘‘x’’ or ‘‘i’’, at the beginning of a line or following a tab or space character, and followed by a ‘‘:’’, an **ex** command, and another ‘‘:’’.

This option is a security problem of immense proportions, and should not be used under any circumstances.

This option will never be implemented.

number, nu [off]

Precede each line displayed with its current line number.

octal [off]

Display unknown characters as octal numbers, instead of the default hexadecimal.

open [on]

Ex only. If this option is not set, the **open** and **visual** commands are disallowed.

optimize, opt [on]

Vi only. Throughput of text is expedited by setting the terminal not to do automatic carriage returns when printing more than one (logical) line of output, greatly speeding output on terminals without addressable cursors when text with leading white space is printed.

This option is not yet implemented.

paragraphs, para [IPLPPPQPP LIpplpipbp]

Vi only. Define additional paragraph boundaries for the **{** and **}** commands. The value of this option must be a character string consisting of zero or more character pairs.

In the text to be edited, the character string `<newline>.<char-pair>`, (where `<char-`

pair> is one of the character pairs in the option's value) defines a paragraph boundary. For example, if the option were set to LaA<space>##, then all of the following additional paragraph boundaries would be recognized:

<newline>.La
<newline>.A<space>
<newline>.##

prompt [on]

Ex only. This option causes **ex** to prompt for command input with a ``:'' character; when it is not set, no prompt is displayed.

readonly, ro [off]

This option causes a force flag to be required to attempt to write the file back to the original file name. Setting this option is equivalent to using the **−R** command line option, or editing a file which lacks write permission.

recdir [/var/tmp/vi.recover]

The directory where recovery files are stored.

If you change the value of **recdir**, be careful to choose a directory whose contents are not regularly deleted. Bad choices include directories in memory based filesystems, or /tmp, on most systems, as their contents are removed when the machine is rebooted.

Public directories like /usr/tmp and /var/tmp are usually safe, although some sites periodically prune old files from them. There is no requirement that you use a public directory, e.g. a sub-directory of your home directory will work fine.

Finally, if you change the value of **recdir**, you must modify the recovery script to operate in your chosen recovery area.

See the section entitled ``**Recovery**'' for further information.

redraw, re [off]

Vi only. The editor simulates (using great amounts of output), an intelligent terminal on a dumb terminal (e.g. during insertions in **vi** the characters to the right of the cursor are refreshed as each input character is typed).

This option is not yet implemented.

remap [on]

If this option is set, it is possible to define macros in terms of other macros. Otherwise, each key is only remapped up to one time. For example, if ``A'' is mapped to ``B'', and ``B'' is mapped to ``C'', The keystroke ``A'' will be mapped to ``C'' if the **remap** option is set, and to ``B'' if it is not set.

report [5]

Set the threshold of the number of lines that need to be changed or yanked before a message will be displayed to the user. For everything but the yank command, the value is the largest value about which the editor is silent, i.e. by default, 6 lines must be deleted before the user is notified. However, if the number of lines yanked is greater than *or equal to* the set value, it is reported to the user.

ruler [off]

Vi only. Display a row/column ruler on the colon command line.

scroll, scr [window / 2]

Set the number of lines scrolled by the **vi <control-D>** and **<control-U>** commands.

Historically, the **ex z** command, when specified without a count, used two times the size of the scroll value; the POSIX 1003.2 standard specified the window size, which is a better choice.

sections, sect [NHSHH HUnhsh]

Vi only. Define additional section boundaries for the **[[** and **]]** commands. The **sections** option should be set to a character string consisting of zero or more character pairs. In the text to be edited, the character string `<newline>.<char-pair>`, (where `<char-pair>` is one of the character pairs in the option's value), defines a section boundary in the same manner that **paragraph** option boundaries are defined.

shell, sh [environment variable SHELL, or /bin/sh]

Select the shell used by the editor. The specified path is the pathname of the shell invoked by the **vi !** shell escape command and by the **ex shell** command. This program is also used to resolve any shell meta-characters in **ex** commands.

shiftwidth, sw [8]

Set the autoindent and shift command indentation width. This width is used by the **autoindent** option and by the **<**, **>**, and **shift** commands.

showdirty [off]

Vi only. Display an asterisk on the colon command line if the file has been modified.

showmatch, sm [off]

Vi only. This option causes **vi**, when a ‘‘`}`’’ or ‘‘`)`’’ is entered, to briefly move the cursor the matching ‘‘`{`’’ or ‘‘`(`’’. See the **matchtime** option for more information.

showmode [off]

Vi only. This option causes **vi** to display a string identifying the current editor mode on the colon command line.

sidescroll [16]

Vi only. Sets the number of columns that are shifted to the left or right, when **vi** is doing left-right scrolling and the left or right margin is crossed. See the **leftright** option for more information.

slowopen, slow [off]

This option affects the display algorithm used by **vi**, holding off display updating during input of new text to improve throughput when the terminal in use is slow and unintelligent.

This option is not yet implemented.

sourceany [off]

If this option is turned on, **vi** historically read startup files that were owned by someone other than the editor user. See the section entitled ‘‘**Startup Information**’’ for more information. This option is a security problem of immense proportions, and should not be used under any circumstances.

This option will never be implemented.

tabstop, ts [8]

This option sets tab widths for the editor display.

taglength, tl [0]

This option sets the maximum number of characters that are considered significant in a tag name. Setting the value to 0 makes all of the characters in the tag name significant.

tags, tag [tags /var/db/libc.tags /sys/kern/tags]
Sets the list of tags files, in search order, which are used when the editor searches for a tag.

term, ttytype, tty [environment variable TERM]
Set the terminal type. Setting this option causes **ex**/**vi** to set (or reset) the environmental variable `TERM`.

terse [off]
This option has historically made editor messages less verbose. It has no effect in this implementation. See the **verbose** option for more information.

tildeop
Modify the ~ command to take an associated motion.

timeout, to [on]
If this option is set, **ex**/**vi** waits for a specific period for a subsequent key to complete a key mapping (see the **keytime** option). If the option is not set, the editor waits until enough keys are entered to resolve the ambiguity, regardless of how long it takes.

ttywerase [off]
Vi only. This option changes how **vi** does word erase during text input. If this option is set, text is broken up into two classes, blank characters and nonblank characters. Changing from one class to another marks the end of a word.

verbose [off]
Vi only. **Vi** historically bells the terminal for many obvious mistakes, e.g. trying to move past the left-hand margin, or past the end of the file. If this option is set, an error message is displayed for all errors.

w300 [no default]
Vi only. Set the window size if the baud rate is less than 1200 baud. See the **window** option for more information.

w1200 [no default]
Vi only. Set the window size if the baud rate is equal to 1200 baud. See the **window** option for more information.

w9600 [no default]
Vi only. Set the window size if the baud rate is greater than 1200 baud. See the **window** option for more information.

warn [on]
Ex only. This option causes a warning message to the terminal if the file has been modified, since it was last written, before a **!** command.

window, w, wi [environment variable LINES]
This option determines the default number of lines in a screenful, as written by the **z** command. It also determines the number of lines scrolled by the **vi** commands **<control-F>** and **<control-B>**. The value of window can be unrelated to the real screen size, although it starts out as the number of lines on the screen (see the section entitled ‘‘**Sizing the Screen**’’ for more information). Setting the value of the **window** option is the same as using the **–w** command line option.

If the value of the **window** option (as set by the **window**, **w300**, **w1200** or **w9600** options) is smaller than the actual size of the screen, large screen movements will result in displaying only that smaller number of lines on the screen. (Further movements in that same area will result in the screen being filled.) This can provide a performance improvement when viewing different places in one or more files over a slow link.

wrapmargin, wm [0]

Vi only. If the value of the **wrapmargin** option is non-zero, **vi** will split lines so that they end at least that number of characters before the right-hand margin of the screen. (Note, the value of **wrapmargin** is *not* a text length. In a screen that is 80 columns wide, the command ‘‘`:set wrapmargin=8`’’ attempts to keep the lines less than or equal to 72 columns wide.)

Lines are split at the previous whitespace character closest to the number. Any trailing whitespace characters before that character are deleted. If the line is split because of an inserted `<space>` or `<tab>` character, and you then enter another `<space>` character, it is discarded.

If wrapmargin is set to 0, or if there is no blank character upon which to split the line, the line is not broken.

wrapscan, ws [on]

This option causes searches to wrap around the end or the beginning of the file, and back to the starting point. Otherwise, the end or beginning of the file terminates the search.

writeany, wa [off]

If this option is set, file-overwriting checks that would usually be made before the **write** and **xit** commands, or before an automatic write (see the **autowrite** option), are not made. This allows a write to any file, provided the file permissions allow it.

16. Additional Features in Nex/Nvi

There are a few features in **nex**/**nvi** that are not found in historic versions of **ex**/**vi**. Some of the more interesting of those features are as follows:

8-bit clean data, large lines, files

Nex/**nvi** will edit any format file. Line lengths are limited by available memory, and file sizes are limited by available disk space. The **vi** text input mode command **<control-X>** can insert any possible character value into the text.

Split screens

The **split** command divides the screen into multiple editing regions. The **<control-W>** command rotates between the foreground screens. The **resize** command can be used to grow or shrink a particular screen.

Background and foreground screens

The **bg** command backgrounds the current screen, and the **fg** command foregrounds backgrounded screens. The **display** command can be used to list the background screens.

Tag stacks

Tags are now maintained in a stack. The **<control-T>** command returns to the previous tag location. The **tagpop** command returns to the most recent tag location by default, or, optionally to a specific tag number in the tag stack, or the most recent tag from a specified file. The **display** command can be used to list the tags stack. The **tagtop** command returns to the top of the tag stack.

New displays

The **display** command can be used to display the current buffers, the backgrounded screens, and the tags stack.

Infinite undo

Changes made during an edit session may be rolled backward and forward. A **.** command immediately after a **u** command continues either forward or backward depending on whether the **u** command was an undo or a redo.

Usage information

The **exusage** and **viusage** commands provide usage information for all of the **ex** and **vi** commands by default, or, optionally, for a specific command or key.

Extended Regular Expressions

The **extended** option causes Regular Expressions to be interpreted as as Extended Regular Expressions, (i.e. *egrep*(1) style Regular Expressions).

Word search

The **<control-A>** command searches for the word referenced by the cursor.

Number increment

The **#** command increments or decrements the number referenced by the cursor.

Previous file

The **previous** command edits the previous file from the argument list.

Left-right scrolling

The **leftright** option causes **nvi** to do left-right screen scrolling, instead of the traditional **vi** line wrapping.

17. Index

Table of Contents