# Vi Command & Function Reference

*Alan P.W. Hewett*

*Revised for version 2.12 by Mark Horton*

## 1. Author's Disclaimer

This document does not claim to be 100% complete. There are a few commands listed in the original document that I was unable to test either because I do not speak **lisp**, because they required programs we don't have, or because I wasn't able to make them work. In these cases I left the command out. The commands listed in this document have been tried and are known to work. It is expected that prospective users of this document will read it once to get the flavor of everything that **vi** can do and then use it as a reference document. Experimentation is recommended. If you don't understand a command, try it and see what happens.

[Note: In revising this document, I have attempted to make it completely reflect version 2.12 of **vi**. It does not attempt to document the VAX version (version 3), but with one or two exceptions (wrapmargin, arrow keys) everything said about 2.12 should apply to 3.1. *Mark Horton*]

## 2. Notation

**[option]** is used to denote optional parts of a command. Many **vi** commands have an optional count. **[cnt]** means that an optional number may precede the command to multiply or iterate the command. **{variable item}** is used to denote parts of the command which must appear, but can take a number of different values. **<character [-character]>** means that the character or one of the characters in the range described between the two angle brackets is to be typed. For example **<esc>** means the **escape** key is to be typed. **<a-z>** means that a lower case letter is to be typed. **^<character>** means that the character is to be typed as a **control** character, that is, with the **<cntl>** key held down while simultaneously typing the specified character. In this document control characters will be denoted using the *upper case* character, but ^<uppercase chr> and ^<lowercase chr> are equivalent. That is, for example, **<^D>** is equal to **<^d>**. The most common character abbreviations used in this list are as follows:

| | |
|---|---|
| <esc> | escape, octal 033 |
| <cr> | carriage return, ^M, octal 015 |
| <lf> | linefeed ^J, octal 012 |
| <nl> | newline, ^J, octal 012 (same as linefeed) |
| <bs> | backspace, ^H, octal 010 |
| <tab> | tab, ^I, octal 011 |
| <bell> | bell, ^G, octal 07 |
| <ff> | formfeed, ^L, octal 014 |
| <sp> | space, octal 040 |
| <del> | delete, octal 0177 |

## 3. Basics

To run **vi** the shell variable **TERM** must be defined and exported to your environment. How you do this depends on which shell you are using. You can tell which shell you have by the character it prompts you for commands with. The Bourne shell prompts with '$', and the C shell prompts with '%'. For these examples, we will suppose that you are using an HP 2621 terminal, whose termcap name is "2621".

### 3.1. Bourne Shell

To manually set your terminal type to 2621 you would type:

```
TERM=2621
export TERM
```

There are various ways of having this automatically or semi-automatically done when you log in. Suppose you usually dial in on a 2621. You want to tell this to the machine, but still have it work when you use a hardwired terminal. The recommended way, if you have the **tset** program, is to use the sequence

```
tset −s −d 2621 > tset$$
. tset$$
rm tset$$
```

in your .login (for csh) or the same thing using '.' instead of 'source' in your .profile (for sh). The above line says that if you are dialing in you are on a 2621, but if you are on a hardwired terminal it figures out your terminal type from an on-line list.

### 3.2. The C Shell

To manually set your terminal type to 2621 you would type:

```
setenv TERM 2621
```

There are various ways of having this automatically or semi-automatically done when you log in. Suppose you usually dial in on a 2621. You want to tell this to the machine, but still have it work when you use a hardwired terminal. The recommended way, if you have the **tset** program, is to use the sequence

```
tset −s −d 2621 > tset$$
source tset$$
rm tset$$
```

in your .login.* The above line says that if you are dialing in you are on a 2621, but if you are on a hardwired terminal it figures out your terminal type from an on-line list.

## 4. Normal Commands

**Vi** is a visual editor with a window on the file. What you see on the screen is **vi**'s current notion of what your file will contain, (at this point in the file), when it is written out. Most commands do not cause any change in the screen until the complete command is typed. Should you get confused while typing a command, you can abort the command by typing an <del> character. You will know you are back to command level when you hear a <bell>. Usually typing an <esc> will produce the same result. When **vi** gets an improperly formatted command it rings the <bell>. Following are the **vi** commands broken down by function.

### 4.1. Entry and Exit

To enter **vi** on a particular *file*, type

**vi** *file*

The file will be read in and the cursor will be placed at the beginning of the first line. The first screenfull of the file will be displayed on the terminal.

To get out of the editor, type

ZZ

If you are in some special mode, such as input mode or the middle of a multi-keystroke command, it may be necessary to type <esc> first.

---

* On a version 6 system without environments, the invocation of tset is simpler, just add the line "tset −d 2621" to your .login or .profile.

### 4.2. Cursor and Page Motion

**NOTE:** The arrow keys (see the next four commands) on certain kinds of terminals will not work with the PDP-11 version of vi. The control versions or the hjkl versions will work on any terminal. Experienced users prefer the hjkl keys because they are always right under their fingers. Beginners often prefer the arrow keys, since they do not require memorization of which hjkl key is which. The mnemonic value of hjkl is clear from looking at the keyboard of an adm3a.

[cnt]<bs> or [cnt]h or [cnt]←
: Move the cursor to the left one character. Cursor stops at the left margin of the page. If cnt is given, these commands move that many spaces.

[cnt]^N or [cnt]j or [cnt]↓ or [cnt]<lf>
: Move down one line. Moving off the screen scrolls the window to force a new line onto the screen. Mnemonic: **N**ext

[cnt]^P or [cnt]k or [cnt]↑
: Move up one line. Moving off the top of the screen forces new text onto the screen. Mnemonic: **P**revious

[cnt]<sp> or [cnt]l or [cnt]→
: Move to the right one character. Cursor will not go beyond the end of the line.

[cnt]-
: Move the cursor up the screen to the beginning of the next line. Scroll if necessary.

[cnt]+ or [cnt]<cr>
: Move the cursor down the screen to the beginning of the next line. Scroll up if necessary.

[cnt]$
: Move the cursor to the end of the line. If there is a count, move to the end of the line "cnt" lines forward in the file.

^
: Move the cursor to the beginning of the first word on the line.

0
: Move the cursor to the left margin of the current line.

[cnt]|
: Move the cursor to the column specified by the count. The default is column zero.

[cnt]w
: Move the cursor to the beginning of the next word. If there is a count, then move forward that many words and position the cursor at the beginning of the word. Mnemonic: next-**w**ord

[cnt]W
: Move the cursor to the beginning of the next word which follows a "white space" (<sp>,<tab>, or <nl>). Ignore other punctuation.

[cnt]b
: Move the cursor to the preceding word. Mnemonic: **b**ackup-word

[cnt]B
: Move the cursor to the preceding word that is separated from the current word by a "white space" (<sp>,<tab>, or <nl>).

[cnt]e
: Move the cursor to the end of the current word or the end of the "cnt"'th word hence. Mnemonic: **e**nd-of-word

[cnt]E
: Move the cursor to the end of the current word which is delimited by "white space" (<sp>,<tab>, or <nl>).

[line number]G
: Move the cursor to the line specified. Of particular use are the sequences "1G" and "G", which move the cursor to the beginning and the end of the file respectively. Mnemonic: **G**o-to

**NOTE:** The next four commands (^D, ^U, ^F, ^B) are not true motion commands, in that they cannot be used as the object of commands such as delete or change.

[cnt]^D
: Move the cursor down in the file by "cnt" lines (or the last "cnt" if a new count isn't given. The initial default is half a page.) The screen is simultaneously scrolled up. Mnemonic: **D**own

[cnt]^U
: Move the cursor up in the file by "cnt" lines. The screen is simultaneously scrolled down. Mnemonic: **U**p

[cnt]^F    Move the cursor to the next page. A count moves that many pages. Two lines of the previous page are kept on the screen for continuity if possible. Mnemonic: **F**orward-a-page

[cnt]^B    Move the cursor to the previous page. Two lines of the current page are kept if possible. Mnemonic: **B**ackup-a-page

[cnt](    Move the cursor to the beginning of the next sentence. A sentence is defined as ending with a ".", "!", or "?" followed by two spaces or a <nl>.

[cnt])    Move the cursor backwards to the beginning of a sentence.

[cnt]}    Move the cursor to the beginning of the next paragraph. This command works best inside **nroff** documents. It understands two sets of **nroff** macros, **–ms** and **–mm**, for which the commands ".IP", ".LP", ".PP", ".QP", "P", as well as the nroff command ".bp" are considered to be paragraph delimiters. A blank line also delimits a paragraph. The **nroff** macros that it accepts as paragraph delimiters is adjustable. See **paragraphs** under the **Set Commands** section.

[cnt]{    Move the cursor backwards to the beginning of a paragraph.

]]    Move the cursor to the next "section", where a section is defined by two sets of **nroff** macros, **–ms** and **–mm**, in which ".NH", ".SH", and ".H" delimit a section. A line beginning with a <ff><nl> sequence, or a line beginning with a "{" are also considered to be section delimiters. The last option makes it useful for finding the beginnings of C functions. The **nroff** macros that are used for section delimiters can be adjusted. See **sections** under the **Set Commands** section.

[[    Move the cursor backwards to the beginning of a section.

%    Move the cursor to the matching parenthesis or brace. This is very useful in C or lisp code. If the cursor is sitting on a **( ) {** or **}** the cursor is moved to the matching character at the other end of the section. If the cursor is not sitting on a brace or a parenthesis, **vi** searches forward until it finds one and then jumps to the match mate.

[cnt]H    If there is no count move the cursor to the top left position on the screen. If there is a count, then move the cursor to the beginning of the line "cnt" lines from the top of the screen. Mnemonic: **H**ome

[cnt]L    If there is no count move the cursor to the beginning of the last line on the screen. If there is a count, then move the cursor to the beginning of the line "cnt" lines from the bottom of the screen. Mnemonic: **L**ast

M    Move the cursor to the beginning of the middle line on the screen. Mnemonic: **M**iddle

m<a-z>    This command does not move the cursor, but it **marks** the place in the file and the character "<a-z>" becomes the label for referring to this location in the file. See the next two commands. Mnemonic: **m**ark **NOTE:** The mark command is not a motion, and cannot be used as the target of commands such as delete.

´<a-z>    Move the cursor to the beginning of the line that is marked with the label "<a-z>".

`<a-z>    Move the cursor to the exact position on the line that was marked with with the label "<a-z>".

´´    Move the cursor back to the beginning of the line where it was before the last "non-relative" move. A "non-relative" move is something such as a search or a jump to a specific line in the file, rather than moving the cursor or scrolling the screen.

``    Move the cursor back to the exact spot on the line where it was located before the last "non-relative" move.

### 4.3. Searches

The following commands allow you to search for items in a file.

[cnt]f{chr}

Search forward on the line for the next or "cnt"'th occurrence of the character "chr". The cursor is placed **at** the character of interest. Mnemonic: **f**ind character

[cnt]F{chr}

Search backwards on the line for the next or "cnt"'th occurrence of the character "chr". The cursor is placed **at** the character of interest.

[cnt]t{chr}

Search forward on the line for the next or "cnt"'th occurrence of the character "chr". The cursor is placed **just preceding** the character of interest. Mnemonic: move cursor up **t**o character

[cnt]T{chr}

Search backwards on the line for the next or "cnt"'th occurrence of the character "chr". The cursor is placed **just preceding** the character of interest.

[cnt]; Repeat the last "f", "F", "t" or "T" command.

[cnt], Repeat the last "f", "F", "t" or "T" command, but in the opposite search direction. This is useful if you overshoot.

[cnt]/[string]/<nl>

Search forward for the next occurrence of "string". Wrap around at the end of the file does occur. The final **</>** is not required.

[cnt]?[string]?<nl>

Search backwards for the next occurrence of "string". If a count is specified, the count becomes the new window size. Wrap around at the beginning of the file does occur. The final **<?>** is not required.

n Repeat the last /[string]/ or ?[string]? search. Mnemonic: **n**ext occurrence.

N Repeat the last /[string]/ or ?[string]? search, but in the reverse direction.

:g/[string]/[editor command]<nl>

Using the **:** syntax it is possible to do global searches ala the standard UNIX "ed" editor.

### 4.4. Text Insertion

The following commands allow for the insertion of text. All multicharacter text insertions are terminated with an <esc> character. The last change can always be **undone** by typing a **u**. The text insert in insertion mode can contain newlines.

a{text}<esc> Insert text immediately following the cursor position. Mnemonic: **a**ppend

A{text}<esc> Insert text at the end of the current line. Mnemonic: **A**ppend

i{text}<esc> Insert text immediately preceding the cursor position. Mnemonic: **i**nsert

I{text}<esc> Insert text at the beginning of the current line.

o{text}<esc> Insert a new line after the line on which the cursor appears and insert text there. Mnemonic: **o**pen new line

O{text}<esc> Insert a new line preceding the line on which the cursor appears and insert text there.

### 4.5. Text Deletion

The following commands allow the user to delete text in various ways. All changes can always be **undone** by typing the **u** command.

[cnt]x Delete the character or characters starting at the cursor position.

[cnt]X Delete the character or characters starting at the character preceding the cursor position.

D Deletes the remainder of the line starting at the cursor. Mnemonic: **D**elete the rest of line

[cnt]d{motion}

Deletes one or more occurrences of the specified motion. Any motion from sections 4.1 and 4.2

can be used here. The d can be stuttered (e.g. [cnt]dd) to delete cnt lines.

### 4.6. Text Replacement

The following commands allow the user to simultaneously delete and insert new text. All such actions can be **undone** by typing **u** following the command.

r<chr> — Replaces the character at the current cursor position with <chr>. This is a one character replacement. No <esc> is required for termination. Mnemonic: **r**eplace character

R{text}<esc> — Starts overlaying the characters on the screen with whatever you type. It does not stop until an <esc> is typed.

[cnt]s{text}<esc> — Substitute for "cnt" characters beginning at the current cursor position. A "$" will appear at the position in the text where the "cnt"'th character appears so you will know how much you are erasing. Mnemonic: **s**ubstitute

[cnt]S{text}<esc> — Substitute for the entire current line (or lines). If no count is given, a "$" appears at the end of the current line. If a count of more than 1 is given, all the lines to be replaced are deleted before the insertion begins.

[cnt]c{motion}{text}<esc>
Change the specified "motion" by replacing it with the insertion text. A "$" will appear at the end of the last item that is being deleted unless the deletion involves whole lines. Motion's can be any motion from sections 4.1 or 4.2. Stuttering the c (e.g. [cnt]cc) changes cnt lines.

### 4.7. Moving Text

**Vi** provides a number of ways of moving chunks of text around. There are nine buffers into which each piece of text which is deleted or "yanked" is put in addition to the "undo" buffer. The most recent deletion or yank is in the "undo" buffer and also usually in buffer 1, the next most recent in buffer 2, and so forth. Each new deletion pushes down all the older deletions. Deletions older than 9 disappear. There is also a set of named registers, a-z, into which text can optionally be placed. If any delete or replacement type command is preceded by **"<a-z>**, that named buffer will contain the text deleted after the command is executed. For example, **"a3dd** will delete three lines starting at the current line and put them in buffer **"a**.* There are two more basic commands and some variations useful in getting and putting text into a file.

["<a-z>][cnt]y{motion}

Yank the specified item or "cnt" items and put in the "undo" buffer or the specified buffer. The variety of "items" that can be yanked is the same as those that can be deleted with the "d" command or changed with the "c" command. In the same way that "dd" means delete the current line and "cc" means replace the current line, "yy" means yank the current line.

["<a-z>][cnt]Y — Yank the current line or the "cnt" lines starting from the current line. If no buffer is specified, they will go into the "undo" buffer, like any delete would. It is equivalent to "yy". Mnemonic: **Y**ank

["<a-z>]p — Put "undo" buffer or the specified buffer down **after** the cursor. If whole lines were yanked or deleted into the buffer, then they will be put down on the line following the line the cursor is on. If something else was deleted, like a word or sentence, then it will be inserted immediately following the cursor. Mnemonic: **p**ut buffer

It should be noted that text in the named buffers remains there when you start editing a new file with the **:e file<esc>** command. Since this is so, it is possible to copy or delete text from one file and carry it over to another file in the buffers. However, the undo buffer and the ability to undo are lost when changing files.

["<a-z>]P — Put "undo" buffer or the specified buffer down **before** the cursor. If whole lines where yanked or deleted into the buffer, then they will be put down on the line preceding the line the cursor is on.

* Referring to an upper case letter as a buffer name (A-Z) is the same as referring to the lower case letter, except that text placed in such a buffer is appended to it instead of replacing it.

If something else was deleted, like a word or sentence, then it will be inserted immediately preceding the cursor.

[cnt]>{motion}
The shift operator will right shift all the text from the line on which the cursor is located to the line where the **motion** is located. The text is shifted by one **shiftwidth**. (See section 6.) **>>** means right shift the current line or lines.

[cnt]<{motion}
The shift operator will left shift all the text from the line on which the cursor is located to the line where the **item** is located. The text is shifted by one **shiftwidth**. (See section 6.) **<<** means left shift the current line or lines. Once the line has reached the left margin it is not further affected.

[cnt]={motion}
Prettyprints the indicated area according to **lisp** conventions. The area should be a lisp s-expression.

### 4.8. Miscellaneous Commands

**Vi** has a number of miscellaneous commands that are very useful. They are:

ZZ
This is the normal way to exit from vi. If any changes have been made, the file is written out. Then you are returned to the shell.

^L
Redraw the current screen. This is useful if someone "write"s you while you are in "vi" or if for any reason garbage gets onto the screen.

^R
On dumb terminals, those not having the "delete line" function (the vt100 is such a terminal), **vi** saves redrawing the screen when you delete a line by just marking the line with an "@" at the beginning and blanking the line. If you want to actually get rid of the lines marked with "@" and see what the page looks like, typing a ^R will do this.

**.**
"Dot" is a particularly useful command. It repeats the last text modifying command. Therefore you can type a command once and then to another place and repeat it by just typing ".".

u
Perhaps the most important command in the editor, u undoes the last command that changed the buffer. Mnemonic: **u**ndo

U
Undo all the text modifying commands performed on the current line since the last time you moved onto it.

[cnt]J
Join the current line and the following line. The <nl> is deleted and the two lines joined, usually with a space between the end of the first line and the beginning of what was the second line. If the first line ended with a "period", then two spaces are inserted. A count joins the next cnt lines. Mnemonic: **J**oin lines

Q
Switch to **ex** editing mode. In this mode **vi** will behave very much like **ed**. The editor in this mode will operate on single lines normally and will not attempt to keep the "window" up to date. Once in this mode it is also possible to switch to the **open** mode of editing. By entering the command **[line number]open<nl>** you enter this mode. It is similar to the normal visual mode except the window is only **one** line long. Mnemonic: **Q**uit visual mode

^]
An abbreviation for a tag command. The cursor should be positioned at the beginning of a word. That word is taken as a tag name, and the tag with that name is found as if it had been typed in a :tag command.

[cnt]!{motion}{UNIX cmd}<nl>
Any UNIX filter (e.g. command that reads the standard input and outputs something to the standard output) can be sent a section of the current file and have the output of the command replace the original text. Useful examples are programs like **cb**, **sort**, and **nroff**. For instance, using **sort** it would be possible to sort a section of the current file into a new list. Using **!!** means take a line or lines starting at the line the cursor is currently on and pass them to the UNIX command. **NOTE:** To just escape to the shell for one command, use :!{cmd}<nl>, see section 5.

z{cnt}<nl>
This resets the current window size to "cnt" lines and redraws the screen.

### 4.9. Special Insert Characters

There are some characters that have special meanings during insert modes. They are:

^V — During inserts, typing a ^V allows you to quote control characters into the file. Any character typed after the ^V will be inserted into the file.

[^]^D or [0]^D — <^D> without any argument backs up one **shiftwidth**. This is necessary to remove indentation that was inserted by the **autoindent** feature. ^<^D> temporarily removes all the autoindentation, thus placing the cursor at the left margin. On the next line, the previous indent level will be restored. This is useful for putting "labels" at the left margin. 0<^D> says remove all autoindents and stay that way. Thus the cursor moves to the left margin and stays there on successive lines until <tab>'s are typed. As with the <tab>, the <^D> is only effective before any other "non-autoindent" controlling characters are typed. Mnemonic: **D**elete a shiftwidth

^W — If the cursor is sitting on a word, <^W> moves the cursor back to the beginning of the word, thus erasing the word from the insert. Mnemonic: erase **W**ord

<bs> — The backspace always serves as an erase during insert modes in addition to your normal "erase" character. To insert a <bs> into your file, use the <^V> to quote it.

### 5. **:** Commands

Typing a ":" during command mode causes **vi** to put the cursor at the bottom on the screen in preparation for a command. In the ":" mode, **vi** can be given most **ed** commands. It is also from this mode that you exit from **vi** or switch to different files. All commands of this variety are terminated by a <nl>, <cr>, or <esc>.

:w[!] [file] — Causes **vi** to write out the current text to the disk. It is written to the file you are editing unless "file" is supplied. If "file" is supplied, the write is directed to that file instead. If that file already exists, **vi** will not perform the write unless the "!" is supplied indicating you *really* want to destroy the older copy of the file.

:q[!] — Causes **vi** to exit. If you have modified the file you are looking at currently and haven't written it out, **vi** will refuse to exit unless the "!" is supplied.

:e[!] [+[cmd]] [file]

Start editing a new file called "file" or start editing the current file over again. The command ":e!" says "ignore the changes I've made to this file and start over from the beginning". It is useful if you really mess up the file. The optional "+" says instead of starting at the beginning, start at the "end", or, if "cmd" is supplied, execute "cmd" first. Useful cases of this are where cmd is "n" (any integer) which starts at line number n, and "/text", which searches for "text" and starts at the line where it is found.

^^ — Switch back to the place you were before your last tag command. If your last tag command stayed within the file, ^^ returns to that tag. If you have no recent tag command, it will return to the same place in the previous file that it was showing when you switched to the current file.

:n[!] — Start editing the next file in the argument list. Since **vi** can be called with multiple file names, the ":n" command tells it to stop work on the current file and switch to the next file. If the current file was modifies, it has to be written out before the ":n" will work or else the "!" must be supplied, which says discard the changes I made to the current file.

:n[!] file [file file ...]

Replace the current argument list with a new list of files and start editing the first file in this new list.

:r file — Read in a copy of "file" on the line after the cursor.

:r !cmd — Execute the "cmd" and take its output and put it into the file after the current line.

:!cmd — Execute any UNIX shell command.

:ta[!] tag — **Vi** looks in the file named **tags** in the current directory. **Tags** is a file of lines in the format:

tag filename **vi**-search-command

If **vi** finds the tag you specified in the **:ta** command, it stops editing the current file if necessary and if the current file is up to date on the disk and switches to the file specified and uses the search pattern specified to find the "tagged" item of interest. This is particularly useful when editing multi-file C programs such as the operating system. There is a program called **ctags** which will generate an appropriate **tags** file for C and f77 programs so that by saying **:ta function<nl>** you will be switched to that function. It could also be useful when editing multi-file documents, though the **tags** file would have to be generated manually.

## 6. Special Arrangements for Startup

**Vi** takes the value of **$TERM** and looks up the characteristics of that terminal in the file **/etc/termcap**. If you don't know **vi**'s name for the terminal you are working on, look in **/etc/termcap**.

When **vi** starts, it attempts to read the variable EXINIT from your environment.* If that exists, it takes the values in it as the default values for certain of its internal constants. See the section on "Set Values" for further details. If EXINIT doesn't exist you will get all the normal defaults.

Should you inadvertently hang up the phone while inside **vi**, or should the computer crash, all may not be lost. Upon returning to the system, type:

vi −r file

This will normally recover the file. If there is more than one temporary file for a specific file name, **vi** recovers the newest one. You can get an older version by recovering the file more than once. The command "vi -r" without a file name gives you the list of files that were saved in the last system crash (but *not* the file just saved when the phone was hung up).

## 7. Set Commands

**Vi** has a number of internal variables and switches which can be set to achieve special affects. These options come in three forms, those that are switches, which toggle from off to on and back, those that require a numeric value, and those that require an alphanumeric string value. The toggle options are set by a command of the form:

:set option<nl>

and turned off with the command:

:set nooption<nl>

Commands requiring a value are set with a command of the form:

:set option=value<nl>

To display the value of a specific option type:

:set option?<nl>

To display only those that you have changed type:

:set<nl>

and to display the long table of all the settable parameters and their current values type:

:set all<nl>

Most of the options have a long form and an abbreviation. Both are listed in the following table as well as the normal default value.

To arrange to have values other than the default used every time you enter **vi**, place the appropriate **set** command in EXINIT in your environment, e.g.

* On version 6 systems Instead of EXINIT, put the startup commands in the file .exrc in your home directory.

EXINIT='set ai aw terse sh=/bin/csh'
export EXINIT

or

setenv EXINIT 'set ai aw terse sh=/bin/csh'

for **sh** and **csh**, respectively. These are usually placed in your .profile or .login. If you are running a system without environments (such as version 6) you can place the set command in the file .exrc in your home directory.

autoindent ai
Default: noai Type: toggle
When in autoindent mode, vi helps you indent code by starting each line in the same column as the preceding line. Tabbing to the right with <tab> or <^T> will move this boundary to the right, and it can be moved to the left with <^D>.

autoprint ap
Default: ap Type: toggle
Causes the current line to be printed after each ex text modifying command. This is not of much interest in the normal **vi** visual mode.

autowrite aw
Default: noaw type: toggle
Autowrite causes an automatic write to be done if there are unsaved changes before certain commands which change files or otherwise interact with the outside world. These commands are :!, :tag, :next, :rewind, ^^, and ^].

beautify bf
Default: nobf Type: toggle
Causes all control characters except <tab>, <nl>, and <ff> to be discarded.

directory dir
Default: dir=/tmp Type: string
This is the directory in which **vi** puts its temporary file.

errorbells eb
Default: noeb Type: toggle
Error messages are preceded by a <bell>.

hardtabs ht
Default: hardtabs=8 Type: numeric
This option contains the value of hardware tabs in your terminal, or of software tabs expanded by the Unix system.

ignorecase ic
Default: noic Type: toggle
All upper case characters are mapped to lower case in regular expression matching.

lisp
Default: nolisp Type: toggle
Autoindent for **lisp** code. The commands **( ) [[** and **]]** are modified appropriately to affect s-expressions and functions.

list
Default: nolist Type: toggle
All printed lines have the <tab> and <nl> characters displayed visually.

magic
Default: magic Type: toggle
Enable the metacharacters for matching. These include **. * < > [string] [^string]** and **[<chr>-<chr>]**.

number nu
Default: nonu Type: toggle
Each line is displayed with its line number.

open
Default: open Type: toggle
When set, prevents entering open or visual modes from ex or edit. Not of interest from vi.

optimize opt
Default: opt Type: toggle
Basically of use only when using the **ex** capabilities. This option prevents automatic <cr>s from taking place, and speeds up output of indented lines, at the expense of losing typeahead on some versions of UNIX.

paragraphs para
Default: para=IPLPPPQPP bp Type: string
Each pair of characters in the string indicate **nroff** macros which are to be treated as the beginning of a paragraph for the **{** and **}** commands. The default string is for the **-ms** and **-mm** macros. To indicate one letter **nroff** macros, such as **.P** or **.H**, quote a space in for the second

character position. For example:

:set paragraphs=P\ bp<nl>

would cause **vi** to consider **.P** and **.bp** as paragraph delimiters.

prompt
Default: prompt Type: toggle
In **ex** command mode the prompt character **:** will be printed when **ex** is waiting for a command. This is not of interest from vi.

redraw
Default: noredraw Type: toggle
On dumb terminals, force the screen to always be up to date, by sending great amounts of output. Useful only at high speeds.

report
Default: report=5 Type: numeric
This sets the threshold for the number of lines modified. When more than this number of lines are modified, removed, or yanked, **vi** will report the number of lines changed at the bottom of the screen.

scroll
Default: scroll={1/2 window} Type: numeric
This is the number of lines that the screen scrolls up or down when using the <^U> and <^D> commands.

sections
Default: sections=SHNHH HU Type: string
Each two character pair of this string specify **nroff** macro names which are to be treated as the beginning of a section by the **]]** and **[[** commands. The default string is for the **-ms** and **-mm** macros. To enter one letter **nroff** macros, use a quoted space as the second character. See **paragraphs** for a fuller explanation.

shell sh
Default: sh=from environment SHELL or /bin/sh Type: string
This is the name of the **sh** to be used for "escaped" commands.

shiftwidth sw
Default: sw=8 Type: numeric
This is the number of spaces that a <^T> or <^D> will move over for indenting, and the amount < and > shift by.

showmatch sm
Default: nosm Type: toggle
When a **)** or **}** is typed, show the matching **(** or **{** by moving the cursor to it for one second if it is on the current screen.

slowopen slow
Default: terminal dependent Type: toggle
On terminals that are slow and unintelligent, this option prevents the updating of the screen some of the time to improve speed.

tabstop ts
Default: ts=8 Type: numeric
<tab>s are expanded to boundaries that are multiples of this value.

taglength tl
Default: tl=0 Type: numeric
If nonzero, tag names are only significant to this many characters.

term
Default: (from environment **TERM**, else dumb) Type: string
This is the terminal and controls the visual displays. It cannot be changed when in "visual" mode, you have to Q to command mode, type a set term command, and do ‘‘vi.’’ to get back into visual. Or exit vi, fix $TERM, and reenter. The definitions that drive a particular terminal type are found in the file **/etc/termcap**.

terse
Default: terse Type: toggle
When set, the error diagnostics are short.

warn
Default: warn Type: toggle
The user is warned if she/he tries to escape to the shell without writing out the current changes.

window
Default: window={8 at 600 baud or less, 16 at 1200 baud, and screen size − 1 at 2400 baud or more} Type: numeric
This is the number of lines in the window whenever **vi** must redraw an entire screen. It is useful

to make this size smaller if you are on a slow line.

w300, w1200, w9600
: These set window, but only within the corresponding speed ranges. They are useful in an EXINIT to fine tune window sizes. For example,

    set w300=4 w1200=12

causes a 4 lines window at speed up to 600 baud, a 12 line window at 1200 baud, and a full screen (the default) at over 1200 baud.

wrapscan ws
: Default: ws Type: toggle
Searches will wrap around the end of the file when is option is set. When it is off, the search will terminate when it reaches the end or the beginning of the file.

wrapmargin wm
: Default: wm=0 Type: numeric
**Vi** will automatically insert a <nl> when it finds a natural break point (usually a <sp> between words) that occurs within "wm" spaces of the right margin. Therefore with "wm=0" the option is off. Setting it to 10 would mean that any time you are within 10 spaces of the right margin **vi** would be looking for a <sp> or <tab> which it could replace with a <nl>. This is convenient for people who forget to look at the screen while they type. (In version 3, wrapmargin behaves more like nroff, in that the boundary specified by the distance from the right edge of the screen is taken as the rightmost edge of the area where a break is allowed, instead of the leftmost edge.)

writeany wa
: Default: nowa Type: toggle
**Vi** normally makes a number of checks before it writes out a file. This prevents the user from inadvertently destroying a file. When the "writeany" option is enabled, **vi** no longer makes these checks.