Appendix: character functions

This appendix gives the uses the editor makes of each character. The characters are presented in their order in the ASCII character set: Control characters come first, then most special characters, then the digits, upper and then lower case characters.

For each character we tell a meaning it has as a command and any meaning it has during an insert. If it has only meaning as a command, then only this is discussed. Section numbers in parentheses indicate where the character is discussed; a ‘f’ after the section number means that the character is mentioned in a footnote.

ˆ@	Not a command character. If typed as the first character of an insertion it is replaced with the last text inserted, and the insert terminates. Only 128 characters are saved from the last insert; if more characters were inserted the mechanism is not available. A **ˆ@** cannot be part of the file due to the editor implementation (7.5f).
ˆA	Unused.
ˆB	Backward window. A count specifies a new window size. Two lines of continuity are kept if possible (2.1, 6.1, 7.2).
ˆC	Unused.
ˆD	As a command, scrolls down a half-window of text. A count gives the number of (logical) lines to scroll, and is remembered for future **ˆD** and **ˆU** commands (2.1, 7.2). During an insert, backtabs over *autoindent* white space at the beginning of a line (6.6, 7.5); this white space cannot be backspaced over.
ˆE	Unused.
ˆF	Forward window. A count specifies a new window size. Two lines of continuity are kept if possible (2.1, 6.1, 7.2).
ˆG	Equivalent to **:f**ESC, printing the current file, whether it has been modified, the current line number and the number of lines in the file, and the percentage of the way through the file that you are. A count specifies a new default window size (2.2).
ˆH (BS**)**	Backspaces one character on the current line. With a count backs up that many characters. The character **h** is a synonym for **ˆH** which is convenient on terminals without backspace keys (3.2). During an insert, eliminates the last input character, backing over it but not erasing it; it remains so you can see what you typed if you wish to type something only slightly different (3.1, 7.5).
ˆI (TAB**)**	Not a command character. When inserted it prints as some number of spaces. When the cursor is at a tab character it rests at the last of the spaces which represent the tab. The spacing of tabstops is controlled by the *tabstop* option (4.1, 6.6).
ˆJ (LF**)**	A linefeed advances to the next line, in the same column or as near this column as possible. A sequence of linefeeds will attempt to move vertically and stay in the original position horizontally. A count causes an advance over several lines (2.3).
ˆK	Unused.
ˆL	The ASCII formfeed character, this causes the screen to be cleared and redrawn. This is useful after a transmission error, if characters typed by a program other than the editor scramble the screen, or after output is stopped by an interrupt (5.4, 7.2f).
ˆM (RETURN**)**	A carriage return advances to the next line, at the first non-white position in the line. Given a count, it advances that many lines (2.3). During an insert, a RETURN causes the insert to continue onto another line (3.1).
ˆN	Advances to the next line, in the same column (see **ˆJ** for more details on how this works). A count advances that many lines (2.3).
ˆO	Unused.
ˆP	Retreats to the previous line, in the same column (see **ˆJ** for more details). A count retreats that many lines (2.3).

ˆQ	Not a command character. During an insert, quotes the next character so that it is possible to insert non-printing and special characters into the file (4.2, 7.5).
ˆR	Redraws the current screen, eliminating logical lines not corresponding to physical lines (lines with only a single @ character on them). On hardcopy terminals in *open* mode, retypes the current line (5.4, 7.2, 7.8).
ˆS	Suspends output temporarily; another **ˆS** restarts it. This works at all times (7.5f).
ˆT	Not a command character. During an insert, with *autoindent* set and at the beginning of the line, inserts *shiftwidth* white space (6.6, 7.5).
ˆU	Scrolls the screen up, inverting **ˆD** which scrolls down. Counts work as they do for **ˆD**, and the previous scroll amount is common to both. On a dumb terminal, **ˆU** will often necessitate clearing and redrawing the screen further back in the file (2.1, 7.2).
ˆV	Unused.
ˆW	Not a command character. During an insert, backs up as **b** would in command mode; the deleted characters remain on the display (see **ˆH**) (7.5).
ˆX	Unused.
ˆY	Unused.
ˆZ	Unused.
ˆ[(ESC**)**	Cancels a partially formed command, such as a **z** when no following character has yet been given; terminates inputs on the last line (read by commands such as **: /** and **?**); ends insertions of new text into the buffer. If an ESC is given when quiescent in command state, the editor rings the bell or flashes the screen. You can thus hit ESC if you don't know what is happening till the editor rings the bell. If you don't know if you are in insert mode you can type ESC**a**, and then material to be input; the material will be inserted correctly whether or not you were in insert mode when you started (1.5, 3.1, 7.5).
ˆ\	Unused.
ˆ]	Searches for the word which is after the cursor as a tag. Equivalent to typing **:ta**, this word, and then an ESC. Mnemonically, this command is ‘‘go right to’’ (7.3).
ˆ↑	Equivalent to **:e #**ESC, returning to the previous position in the last edited file, or editing a file which you specified if you got a 'No write since last change diagnostic' and do not want to have to type the file name again (7.3). (You have to do a **:w** before ˆ↑ will work in this case. If you do not wish to write the file you should do **:e! #**ESC instead.)
ˆ_	Unused.
SPACE	Advances to the next character in the current line. With a count, advances that many characters.
!	An operator, which processes lines from the buffer with reformatting commands. Follow **!** with the object to be processed, and then the command name terminated by ESC. Doubling **!** and preceding it by a count causes count lines to be filtered; otherwise the count is passed on to the object after the **!**. Thus **2!}***fmt*ESC reformats the next two paragraphs by running them through the program *fmt*. If you are working on LISP, the command **!%***grind*ESC, given at the beginning of a function, will run the text of the function through the LISP grinder (6.7, 7.3). To read a file or the output of a command into the buffer use **:r** (7.3). To simply execute a command use **:!** (7.3).
"	Precedes a named buffer specification. There are named buffers **1–9** used for saving deleted text and named buffers **a–z** into which you can place text (4.3, 6.3)
#	Unused. If this is your erase character, it will delete the last character you typed in input mode, and must be preceded with a \ to insert it, since it normally backs over the last input character you gave.

$	Moves to the end of the current line. If you **:se list**ESC, then the end of each line will be shown by printing a **$** after the end of the displayed text in the line. Given a count, advances to the count'th following end of line; thus **2$** advances to the end of the following line.
%	Moves to the parenthesis or brace **{ }** which balances the parenthesis or brace at the current cursor position.
&	Unused.
´	When followed by a ´ returns to the previous context at the beginning of a line. The previous context is set whenever the current line is moved in a non-relative way. When followed by a letter **a**–**z**, returns to the line which was marked with this letter with a **m** command, at the first non-white character in the line. A count before a ´ gives a new window size for the next screen redraw (2.2, 5.3). When used with an operator such as **d**, the operation takes place over complete lines; if you use \`, the operation takes place from the exact marked place to the current cursor position within the line.
(	Retreats to the beginning of a sentence, or to the beginning of a LISP s-expression if the *lisp* option is set. A sentence ends at a **. !** or **?** which is followed by either the end of a line or by two spaces. Any number of closing **)] "** and ´ characters may appear after the **. !** or **?**, and before the spaces or end of line. Sentences also begin at paragraph and section boundaries (see **{** and **[[** below). A count advances that many sentences (4.2, 6.8).
)	Advances to the beginning of a sentence. A count repeats the effect. See **(** above for the definition of a sentence (4.2, 6.8).
*****	Unused.
+	Advances to the next line at the first non-white character. A count repeats the effect. A RETURN is equivalent to a **+**. If the line moved to is not on the screen, the screen is scrolled, or cleared and redrawn if a large amount of scrolling would be necessary (2.3).
,	Reverse of the last **f F t** or **T** command, looking the other way in the current line. Especially useful after hitting too many **;** characters. A count repeats the search.
–	Retreats to the previous line at the first non-white character. This is the inverse of **+** and RETURN. If the line moved to is not on the screen, the screen is scrolled, or cleared and redrawn if this is not possible. If a large amount of scrolling would be required the screen is also cleared and redrawn, with the current line at the center (2.3).
.	Repeats the last command which changed the buffer. Especially useful when deleting words or lines; you can delete some words/lines and then hit **.** to delete more and more words/lines. Given a count, it passes it on to the command being repeated. Thus after a **2dw**, **3.** deletes three words (3.3, 6.3, 7.2, 7.4).
/	Reads a string from the last line on the screen, and scans forward for a line containing this string. The normal input editing sequences may be used during the input on the bottom line; an returns to command state without ever searching. The search begins when you hit ESC to terminate the pattern; the cursor moves to the beginning of the last line to indicate that the search is in progress; the search may then be terminated with a DEL or RUB, returning the cursor to its initial position. Searches normally wrap end-around to find a string anywhere in the buffer. A count to the command specifies the a new window size. When used with an operator entire lines are always affected. In this case it is sometimes convenient to specify an offset from the line matched by the pattern. You can do this by giving a closing **/** and then an offset *+n* or *–n*. If you wish to afffect the region from where you were to where the serach terminates, use \`\`.

	To include the character / in the search string, you must escape it with a preceding \. A ↑ at the beginning of the pattern forces the match to occur at the beginning of a line only; this speeds the search. A **$** at the end of the pattern forces the match to occur at the end of a line only. More extended pattern matching is available, see section 7.4 (1.5, 2,2, 6.1, 7.2, 7.4).
0	Moves to the first character on the current line. Also used, in forming numbers, after an initial **1–9**.
1–9	Used to form numeric arguments to commands (2.3, 7.2).
:	A prefix to a set of commands for file and option manipulation and escapes to the system. Input is given on the bottom line and terminated with an ESC, and the command then executed. You can return to where you were by hitting DEL or RUB if you hit **:** accidentally (see primarily 6.2 and 7.3).
;	Repeats the last single character find which used **f F t** or **T**. A count iterates the basic scan (4.1).
<	An operator which shifts lines left one *shiftwidth*, normally 8 spaces. Like all operators, affects lines when repeated, as in <<. Counts are passed through to the basic object, thus **3<<** shifts three lines (6.6, 7.2).
=	Reindents line for LISP, as though they were typed in with *lisp* and *autoindent* set (6.8).
>	An operator which shifts lines right one *shiftwidth*, normally 8 spaces. Affects lines when repeated as in >>. Counts repeat the basic object (6.6, 7.2).
?	Scans backwards, the opposite of **/**. See the **/** description above for details on scanning (2.2, 6.1, 7.4).
@	Unused. If this is your kill character, you must escape it with a \ to type it in during input mode, as it normally backs over the input you have given on the current line (3.1, 3.4, 7.5).
A	Appends at the end of line, a synonym for **$a** (7.2).
B	Backs up a word, where words are composed of non-blank sequences, placing the cursor at the beginning of the word. A count repeats the effect (2.4).
C	Changes the rest of the text on the current line; a synonym for **c$**.
D	Deletes the rest of the text on the current line; a synonym for **d$**.
E	Moves forward to the end of a word, defined as blanks and non-blanks, like **B** and **W**. A count repeats the effect.
F	Finds a single following character, backwards in the current line. A count repeats this search that many times (4.1).
G	Goes to the line number given as preceding argument, or the end of the file if no preceding count is given. The screen is redrawn with the new current line in the center if necessary (7.2).
H	Homes the cursor to the top line on the screen. If a count is given, then the cursor is moved to the count'th line on the screen. In any case the cursor is moved to the first non-white character on the line. If used as the target of an operator, full lines are affected (2.3, 3.2).
I	Inserts at the beginning of a line; a synonym for ↑**i**.
J	Joins together lines, supplying appropriate white space: one space between words, two spaces after a **.**, and no spaces at all if the first character of the joined on line is **)**. A count causes that many lines to be joined rather than the default two (6.5, 7.1f).
K	Unused.

L	Moves the cursor to the first non-white character of the last line on the screen. With a count, to the first non-white of the count'th line from the bottom. Operators affect whole lines when used with **L** (2.3).
M	Moves the cursor to the middle line on the screen, at the first non-white position on the line (2.3).
N	Scans for the next match of the last pattern given to **/** or **?**, but in the reverse direction; this is the inverse of **n**.
O	Opens a new line above the current line and inputs text there up to an ESC. A count can be used on dumb terminals to specify a number of lines to be opened; this is generally obsolete, as the *slowopen* option works better (3.1).
P	Puts the last deleted text back before/above the cursor. The text goes back as whole lines above the cursor if it was deleted as whole lines. Otherwise the text is inserted between the characters before and at the cursor. May be preceded by a named buffer specification **"***x* to retrieve the contents of the buffer; buffers **1**–**9** contain deleted material, buffers **a**–**z** are available for general use (6.3).
Q	Quits from *vi* to *ex* command mode. In this mode, whole lines form commands, ending with a RETURN. You can give all the **:** commands; the editor supplies the **:** as a prompt (7.7).
R	Replaces characters on the screen with characters you type (overlay fashion). Terminates with an ESC.
S	Changes whole lines, a synonym for **cc**. A count substitutes for that many lines. The lines are saved in the numeric buffers, and erased on the screen before the substitution begins.
T	Takes a single following character, locates the character before the cursor in the current line, and places the cursor just after that character. A count repeats the effect. Most useful with operators such as **d** (4.1).
U	Restores the current line to its state before you started changing it (3.5).
V	Unused.
W	Moves forward to the beginning of a word in the current line, where words are defined as sequences of blank/non-blank characters. A count repeats the effect (2.4).
X	Deletes the character before the cursor. A count repeats the effect, but only characters on the current line are deleted.
Y	Yanks a copy of the current line into the unnamed buffer, to be put back by a later **p** or **P**; a very useful synonym for **yy**. A count yanks that many lines. May be preceded by a buffer name to put lines in that buffer (7.4).
Z	Unused.
[[	Backs up to the previous section boundary. A section begins at each macro in the *sections* option, normally a '.NH' or '.SH' and also at lines which which start with a formfeed **^L**. Lines beginning with **{** also stop **[[**; this makes it useful for looking backwards, a function at a time, in C programs. If the option *lisp* is set, stops at each **(** at the beginning of a line, and is thus useful for moving backwards at the top level LISP objects. A count gives a new window size for the next screen redraw (4.2, 6.1, 6.6, 7.2).
\	Unused.
]]	Forward to a section boundary, see **[[** for a definition (4.2, 6.1, 6.6, 7.2).
↑	Moves to the first non-white position on the current line (4.4).
_	Unused.

`	When followed by a ` returns to the previous context. The previous context is set whenever the current line is moved in a non-relative way. When followed by a letter **a**–**z**, returns to the position which was marked with this letter with a **m** command. A count before a ` gives a new window size for the next screen redraw. When used with an operator such as **d**, the operation takes place from the exact marked place to the current position within the line; if you use ´, the operation takes place over complete lines (2.2, 5.3).
a	Appends arbitrary text after the current cursor position; the insert can continue onto multiple lines by using RETURN within the insert. A count causes the inserted text to be replicated, but only if the inserted text is all on one line. The insertion terminates with an ESC (3.1, 7.2).
b	Backs up to the beginning of a word in the current line. A word is a sequence of alphanumerics, or a sequence of special characters. A count repeats the effect (2.4).
c	An operator which changes the following object, replacing it with the following input text up to an ESC. If more than part of a single line is affected, the text which is changed away is saved in the numeric named buffers. If only part of the current line is affected, then the last character to be changed away is marked with a **$**. A count causes that many objects to be affected, thus both **3c)** and **c3)** change the following three sentences (7.4).
d	An operator which deletes the following object. If more than part of a line is affected, the text is saved in the numeric buffers. A count causes that many objects to be affected; thus **3dw** is the same as **d3w** (3.3, 3.4, 4.1, 7.4).
e	Advances to the end of the next word, defined as for **b** and **w**. A count repeats the effect (2.4, 3.1).
f	Finds the first instance of the next character following the cursor on the current line. A count repeats the find (4.1).
g	Unused.
h	Backspaces a single character in the current line; a synonym for **ˆH**. A count repeats the effect (3.1, 7.5).
i	Inserts text before the cursor, otherwise like **a** (7.2).
j	Unused.
k	Unused.
l	Unused.
m	Marks the current position of the cursor in the mark register which is specified by the next character **a**–**z**. Return to this position or use with an operator using ` or ´ (5.3).
n	Repeats the last **/** or **?** scanning commands (2.2).
o	Opens new lines below the current line; otherwise like **O** (3.1).
p	Puts text after/below the cursor; otherwise like **P** (6.3).
q	Unused. In a *vi* command executed from within *ex* this returns to *ex* command mode.
r	Replaces the single character at the cursor with a single character you type. The new character may be a RETURN; this is the easiest way to split lines. A count replaces each of the following count characters with the single character given; see **R** above which is the more usually useful iteration of **r** (3.2).
s	Changes the single character under the cursor to the text which follows up to an ESC; given a count, that many characters from the current line are changed. The last character to be changed is marked with **$** as in **c** (3.2).
t	Advances the cursor upto the character before the next character typed. Most useful with operators such as **d** and **c** to delete the characters up to a following character.

	You can use **.** to delete more if this doesn't delete enough the first time (4.1).
u	Undoes the last change made to the current buffer. If repeated, will alternate between these two states, thus is its own inverse. When used after an insert which inserted text on more than one line, the lines are saved in the numeric named buffers (3.5).
v	Unused.
w	Advances to the beginning of the next word, as defined by **b** (2.4).
x	Deletes the single character under the cursor. With a count deletes deletes that many characters forward from the cursor position, but only on the current line (6.5).
y	An operator, yanks the following object into the unnamed temporary buffer. If preceded by a named buffer specification, **"***x*, the text is placed in that buffer also. Text can be recovered by a later **p** or **P** (7.4).
z	Redraws the screen with the current line placed as specified by the following character: RETURN specifies the top of the screen, **.** the center of the screen, and **–** at the bottom of the screen. A count may be given after the **z** and before the following character to specify the new screen size for the redraw. A count before the **z** gives the number of the line to place in the center of the screen instead of the default current line.
{	Retreats to the beginning of the beginning of the preceding paragraph. A paragraph begins at each macro in the *paragraphs* option, normally '.IP', '.LP', '.PP', '.QP' and '.bp'. A paragraph also begins after a completely empty line, and at each section boundary (see **[[** above) (4.2, 6.8, 7.6).
|	Places the cursor on the character in the column specified by the count (7.1, 7.2).
}	Advances to the beginning of the next paragraph. See **{** for the definition of paragraph (4.2, 6.8, 7.6).
~	Unused.
^? (DEL**)**	Interrupts the editor, returning it to command accepting state (1.5, 7.5)