Introduction to vi

Before mice and menus, cut and paste, there was vi. Vi is a simple yet powerful text editor, which uses typed commands to navigate and edit files. Like touch typing, vi is quick once learned.

Vi has 3 modes: navigation, insertion, command.
To exit insertion mode, press <esc> (escape).
<esc> also aborts any half-typed command.
To use command mode, type ":", the command, and <rtn> (return).

Navigation

hjkl — act as "arrow" keys (left, down, up, right); h moves one character left, j one line down, k one line up, l one character right. Arrow keys also work; the space bar advances one character.

^F,B — move forward or back one screenful

nG — goto line n. Without n, goes to bottom.
"n" can be relative to current line (".");
".+10G" moves 10 lines ahead.

w,b — move one word forward or back.
nw, nb move n words forward or back.
W,B — same, but a "word" is now bounded only by spaces (not by symbols).

f,F — f<char> "finds" (moves to) next character <char> on the line; F finds previous character.
; — repeats last f or F

^,$ — ^ moves to first nonblank character on the line; $ moves to line end

m<char> — marks line; '<char> goes there.

Insert text

i — begin insert mode ("INSERT" appears at bottom). In insert mode, what you type is inserted, in front of cursor.
Exit insert mode with <esc>.
I — same, but inserts at first nonblank character.
a — "append"; same, but inserts just after cursor.
A — same, but inserts at line end.

Change text

ct<char> — change all characters on a line up to <char> (deletes them, and enters insert mode.)

ct$ changes everything to the end of the line.

ns — substitute n characters (deletes them and enters insert mode).

J — joins current and next line into one line.

Delete text

x — deletes ("x-es out") a character;
nx deletes n characters.

d — d<thing> deletes that thing.
dw deletes the next word; ndw deletes n words.
dt<char> deletes up to next character <char>,
d$ deletes to the end of the line

dd — delete a line. ndd deletes n lines.

Repeat change

. — repeats last insert, change, or delete command, at present cursor location.

Undo

u — undo last change (can be repeated).
^r — redo last undone change.
U — restore current line to previous state.

Put text from buffer

Text most recently deleted (whether one or many characters, words, or lines) is saved in a "buffer".

p — puts buffer contents into text. If buffer contains lines, contents go below the current line. If buffer contains words or characters, these insert just after the cursor.
P — same, but puts contents above current line (if lines) or before cursor (if words or characters).

np, nP — puts n copies of the buffer.

Yank text into buffer

y — y<thing> "yanks" text into the buffer
(like d, but puts text in buffer without deleting).
So yw yanks a word, nyw yanks n words,
yt<char> yanks up to next <char>.
y'<char> yanks from mark to current location.

Y — yanks an entire line; nY yanks n lines.

Named buffers

Besides the “unnamed” buffer into which text is normally yanked or deleted, there are “named” buffers. *Buffer contents survive when a new file is edited, and even after vi is exited.*

Named buffers are a good way to cut and paste within or between files, because contents survive and are not clobbered by other editing.

n”aY — yanks n lines into buffer named “a” (names are single letters).

n”add — deletes n lines into buffer “a”.

Block mode

^V — enters “block mode” for selecting, copying, and pasting blocks of text.
In block mode, use hjkl or arrow keys to highlight block, then y to yank or d to delete.

To paste a block from the buffer, use p or P.

Find

/<pattern> — finds next instance of <pattern> (patterns described below).

/ — finds next instance of last pattern searched for.

? — like /, but searches backwards.

Patterns

Pattern matching in vi is powerful, but takes practice to use effectively.

The simplest pattern is a literal string. “/dog” searches for “dog”.

^,$ — match the beginning or end of the line. “/^dog” only finds “dog” if it begins the line.

\<, \> — match the beginning or end of a word. “/\<dog” matches “dog” in “ hound dog” but not in “hounddog”.

vi supports “wildcard” pattern matching. Here are some common examples:

. — matches any single character
[0-9] — matches any digit
(can use other ranges, e.g., [1-5])
[a-z] — matches any lower case letter
(can use other ranges, e.g., [a-e])
[A-Z] — matches any upper case letter
* — matches one or more repeats of previous matched character; hence [a-z]* matches any lower case word
[.!?] — matches any one of these three characters
([ ] list can include anything)

To match characters like ., *, \ with special meaning in patterns, preface with backslash (“\”)

Command mode

To access commands that operate on the entire file, type “:”, the command, and <rtn>.

:f — file info. Reports number of lines, current line and character #.

:w — write (save) the file. Changes to a file are not saved until written.
“w <filename>” writes the open file to <filename>.
To write changes and quit, :wq.

:vi <filename> — quit editing current file, edit <filename>.

:q — quit vi. To quit after editing without saving changes, q!.

vi can be launched with wildcard file specification; “vi *.log” opens all files matching *.log, one after the other. When editing multiple files:

:n — edit the next file

Substitute (find and replace)

Find and replace in vi is powerful, but takes practice to use effectively.

:s/<findPattern>/<replacePattern>/ substitutes the first instance of <findPattern> with <replacePattern>, on the current line.

Most often, <findPattern> and <replacePattern> are literal strings; “s/dog/cat/“ replaces the first instance of “dog” with “cat”.

If followed by g (global), as in “s/dog/cat/g”, all instances on the line are replaced.

If preceded by a line specification (or “linespec”), substitute acts on a range of lines. Examples are
m,n — lines m to n
.,.+10 — current line to 10 lines ahead
1,$ — first line to last line ($)

<findPattern> can use pattern matching special characters, just as in searching with “/“ and “?”

Substitute with wildcards

<findPattern> can also use wildcards, like [0-9], ., and *. In such cases, we usually want to replace with a pattern based on what was matched.
If we bracket part of <findPattern> by \(…\), the characters matched are saved in a buffer.
Then in <replacePattern>, we include those characters by invoking the buffer — \1, \2, … for the first or second bracketed part matched.

For example, to reverse order of names in a list
John Doe
Mary Moe
Richard Roe
…
we replace
1,$s/\([a-zA-Z]*\) \([a-zA-Z]*\)/\2, \1/

The pattern [a-zA-Z] matches any single upper or lower case letter; hence [a-zA-Z]* matches any number of letters. The first bracket matches first name, followed by a space (matches the space between names), followed by the second bracket (last name). The replacement reverses the names, and adds a comma.