Patrick Newman
Very Incomplete Vim Reference September 30, 2019

There are 3 main modes: navigation (moving around the page), insert (entering text), and command (sort of like menus, but also many fancy commands). This guide does not even touch on a great many things (eg: changing settings).

1 Navigation Mode

Arrow keys work as you'd expect. Depending on your terminal, page up/down may or may not. HJKL work as left down up right for reasons that made sense on a specific keyboard in the 1970s. (seriously)

Typing a number and then a direction (with the arrow keys) moves you that many characters/lines

number G go to line number, as will number gg G go to end

ĝo to start of line

[go to start of paragraph

] go to end of paragraph

control + u	go up/back half a page/screen
control + f	go forward/down a full page/screen
control + b	go up/back a full page/screen
control + d	go forward/down half a page/screen

/string searches the text for string use n to go to the next instance and N to go to the previous

1.1 Cut/Copy/Paste

dd cut current line 5dd cut five lines (current plus next 4) D or d$ cut rest of current line (everything to the right of the cursor, not including the end of the line itself) d% if the cursor is over a set of paratheses, etc, cut the parens and everything between them. yy "yank" (copy) current line to buffer. y2y or 2yy yank 2 lines. p paste whatever is currently in the buffer. 4p paste 4 times. This can get spammy.

1.2 Macros

You can automate a bunch of tasks for eg: editing a file and using various navigation mode commands with these. qq start recording in register q. You can use any letter for the second choice to record to a different register (eg:

qt to record to register t). @q play back the macro in register q. 2@a play back the macro in register a twice.

2 Insertion Mode

2.1 Entering

i begin at the start/just before the character under the cursor.
a begin at the end/just after the character under the cursor.
I begin at the start of the current line.
A begin at the end of the current line.
s begin by cutting the character under the cursor.
S begin by cutting the current line.

2.2 Digraphs

Type control + k then a 2 character combination. Typically a letter and then a modifier or diacritic:

letter+punctuation eg: a' or n	letter with relevant diacritics eg: á or ñ
letter+asterisk eg: a* b* G*	greek letter eg: α β Γ

<- -< ← also >- -> → , => etc
for a full list, enter :digraphs or :dig

3 Command Mode

typing a colon (:) in navigation mode enters command mode, escape exits, and enter runs the command. Typing in the first few characters of a command and then pressing tab will attempt to complete it. Repeatedly pressing tab will cycle through the possibilities. If the first few characters of a command are unique, you can Command history can be scrolled through (up/down arrow), and is kept even if one quits/restarts vim. There are far too many commands (including changing settings) to go over here, but :help can give a start. I will not be including the starting colons for the rest of this section, but they should be clearly implicit.

3.1 shell stuff

Once you're in command mode, !command lets you run various shell comands. eg: !ls will list files in the current (working) directory. Perhaps more usefully you can use !python3 current_file.py or !pdflatex current_file.tex

3.2 file and buffer management

w saves (well, writes to disk)

w! force save. Usually used to overwrite a file that is not the one you opened.

w filename save as the specified filename (use w! if the file already exists)

e filename open/begin editing filename in a new buffer. If the file does not exist, an empty file is created

q quit (exit vim)

q! force quit. Useful if you made changes that you want to throw out

qa quit all. If you have multiple buffers open...

qa! force quit all. Nice if you "accidentally" opened a bunch of files and don't actually want to go through all of them.

bn next buffer.

bp previous buffer.

bd delete current buffer/close file. This and the previous two may sometimes need to be forced (bd! etc) if you want to go through buffers with unsaved changes.

ls list buffers.

3.3 find and replace

In practice, these are an introduction to regular expressions, and can be used (with sometimes slightly different syntax) in sed, etc. There are a great deal of possible combinations, so going with examples:

s searches so you can do find and replace. By default all of these are case sensitive.

s/foo/bar/ replaces the first instance of “foo” on the current line with “bar”. Most characters can be used as-is, but []\|/.*+& need to be prefixed with a \.

.,+s/foo/bar/ replaces the first instance of “foo” with “bar” on each of the current and next lines.

2,25s/foo/bar/g on lines 3 through 25, replace every instance of “foo” with “bar”.

.,$s/foo/bar/ replaces the first instance of “foo” with “bar” from the current line through the end of the buffer.

%s/[123]/x/g searches the entire buffer, and replaces instance of the digits 1, 2, or 3 with an x.

7s/[a-z]/\t/g On line 7, replaces every instance of a lower case letter in the latin alphabet (a through z) with a tab.

g/foo/ searches every line (by default, lines can be specified) for pattern “foo” that can be fed into other things.

g/foo/s/bar/baz/ replaces “bar” with “baz” on lines that contain “foo”.

v/foo/ searches every line which does not have pattern “foo”.

v/sporks/d deletes every line that does not contain the word “sporks”.

s/Kitten/cat/gi Replaces every instance of ”kitten” on the current line with ”cat”, and ignores case.

%s/\n/\r/ replace every line break with a line break (assuming *nix formatted file)

%s/.*/ls -lh &\r wc &/ takes every line and converts it to two lines, one with “ls -lh ” followed by the original text, and then a second one with “wc ” followed by the original text. This example is explicitly the sort of thing you would do to put together a shell script.