

Extracted from:

Practical Vim

Edit Text at the Speed of Thought

This PDF file contains pages extracted from *Practical Vim*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printer versions; the content is otherwise identical.

Copyright © 2012 The Pragmatic Programmers, LLC.

All rights reserved.

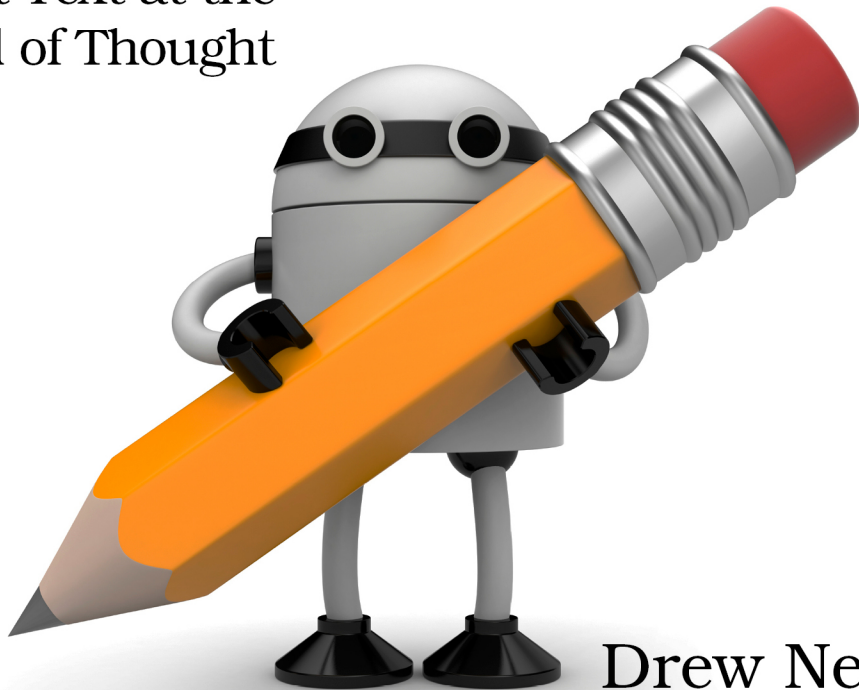
No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

Practical Vim

Edit Text at the
Speed of Thought



Drew Neil

Foreword by Tim Pope

Edited by Kay Keppler



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at <http://pragprog.com>.

The team that produced this book includes:

Kay Keppler (editor)
Potomac Indexing, LLC (indexer)
Molly McBeath (copyeditor)
David J. Kelly (typesetter)
Janet Furlow (producer)
Juliet Benda (rights)
Ellie Callahan (support)

Copyright © 2012 The Pragmatic Bookshelf.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

Printed in the United States of America.
ISBN-13: 978-1-934356-98-2
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—September 2012


Meet the Search Command

In this tip, we'll cover the basics of using the search command, including how to specify the direction of a search, repeat (or reverse) the last search, and work with the search history.

Execute a Search

From Normal mode, the `/` key brings up Vim's search prompt. Here we can enter the pattern, or literal text, that we want to search for. Vim does nothing until we press the `<CR>` key to execute the search. If we press the `<Esc>` key instead, the search prompt will be dismissed and we'll return to Normal mode.

When we execute a search, Vim scans forward from the current cursor position, stopping on the first match that it finds. If nothing is found before the end of the document, Vim informs us “search hit BOTTOM, continuing at TOP.” This means that in some circumstances, a forward search can take us backward. That's not as disorienting as it might sound. Just remember that the search command wraps around the document, and it'll make sense.

If you ever need to search from the current cursor position to the end of the document *without* wrapping around, you can disable the 'wrapscan' option (see `:h 'wrapscan'` .

Specify the Search Direction

When a search is initiated with the `/` key, Vim scans the document forward. If we use the `?` key to bring up the search prompt, Vim searches backward instead. The search prompt always begins with either the `/` or `?` character, which indicates in what direction the search will scan.

Repeat the Last Search

The `n` command jumps to the next match, and the `N` command jumps to the previous match. We can easily navigate between matches in the current document with the `n` and `N` commands. But the definition of “next match” depends on context.

The `n` command preserves the direction as well as any offsets that were applied to the previous search (we'll meet offsets in [Tip 83, on page ?](#)). So if we execute a forward search using `/`, then `n` will continue searching forward.

Whereas if we used `?` for the original search, then `n` will continue backward. Meanwhile, the `N` command always goes in the opposite direction from the last search.

Sometimes we might want to repeat a search using the same pattern but changing the direction or offset. In this case, it's useful to know that if we execute a search without providing a pattern, Vim will just reuse the pattern from the previous search. [Table 22, Options for Repeating a Search, on page 7](#) summarizes the matrix of options for repeating a search:

Suppose that we use `?` to initiate a search. Having jumped backward to the previous match, we then decide that we want to skip forward through the remainder of the matches. We could do it with the `N` key, but somehow that makes everything feel upside down. Instead, we could execute `/<CR>`. This executes a forward search, reusing the same pattern. Now, we can use the `n` key to skip forward through the rest of the matches in the document.

In [Tip 84, on page ?](#), we'll meet a more complex example where we change the offset each time we repeat a search.

Recall Historical Searches

Vim records our search patterns so we can easily recall them. When the search prompt is visible, we can scroll through the previous searches by pressing the `<Up>` key. In fact, the interface for browsing the search history is just the same as for browsing the command-line history. We covered this in more depth in [Tip 34, on page ?](#). We'll put these techniques into action in [Tip 85, on page ?](#).

Tip 80

Highlight Search Matches

Vim can highlight search matches, but this feature is not enabled by default. Learn how to enable it, and (just as importantly) how to mute it for those times when the highlighting takes over.

The search command allows us to jump quickly between matches, but by default, Vim does nothing to make them stand out visually. We can fix this by enabling the `'hlsearch'` option, (see `:h 'hlsearch'` ⓘ), which causes all matches to be highlighted throughout the active document as well as in any other open split windows.

Command	Effect
n	Jump to next match, preserving direction and offset
N	Jump to previous match, preserving direction and offset
/<CR>	Jump forward to next match of same pattern
?<CR>	Jump backward to previous match of same pattern

Table 22—Options for Repeating a Search

Mute Search Highlighting

Search highlighting is a useful feature, but sometimes it can make itself unwelcome. If we search for a common string, for example, or a pattern with hundreds of matches, we'll soon find that our workspace is riddled with yellow (or whatever hue the active color scheme uses).

In this scenario, we could run `:set nohlsearch` to disable the feature entirely (`:set nohls` and `:set hls!` also work). But when we come to execute another search, we might wish to reenable the feature again.

Vim has a more elegant solution. The `:nohlsearch` command can be used to mute the search highlighting temporarily (see `:h :noh` ⓘ). It will stay muted until the next time you execute a new or repeat search command. See [Create a Shortcut to Mute Highlighting, on page 7](#), for a suggested mapping.

Create a Shortcut to Mute Highlighting

Typing `:noh<CR>` to mute search highlighting is laborious. We can speed things up by creating a mapping such as this:

```
nnoremap <silent> <C-l> :<C-u>nohlsearch<CR><C-l>
```

Normally, `<C-l>` clears and redraws the screen (see `:h CTRL-L` ⓘ). This mapping builds on top of the usual behavior by muting search highlighting.

Tip 81

Preview the First Match Before Execution

Vim's search command is much more useful when the incremental search feature is enabled. Here are a couple of ways that this option can improve your workflow.

By default, Vim sits idle as we prepare our search pattern, only springing into action when we press `<CR>`. My favorite enhancement is enabled with the 'incsearch' setting (see `:h 'incsearch'` ⓘ). This tells Vim to show a preview of the first match based on what has been entered so far into the search field. Each time we enter another character, Vim instantly updates the preview. This table illustrates how it works:

Keystrokes	Buffer Contents
{start}	T he car was the color of a carrot.
/car	The car was the color of a carrot.
/carr	The car was the color of a carrot .
/carr<CR>	The car was the color of a carrot .

After typing “car” into the search field, Vim highlights the first match, which in this case is the word “car” itself. As soon as we enter the next “r” character, our preview ceases to match, and Vim skips forward to the next matching word. This time, it’s “carrot.” If we were to press the `<Esc>` key at this point, the search prompt would be dismissed and our cursor restored to its original position at the start of the line. But instead, we press `<CR>` to execute the command, causing our cursor to jump to the start of the word “carrot.”

This instant feedback lets us know when we’ve hit our target. If our intention was simply to move the cursor to the start of the word “carrot,” then there’s no need to type the full word into the search field. In this case, `/carr<CR>` is enough. Without the 'incsearch' feature enabled, we wouldn’t know whether or not our pattern would hit the target until we executed the search.

Check for the Existence of a Match

In our example, we have two partial matches for “car” on the same line. But imagine if the words “car” and “carrot” were separated by several hundred words. When we updated our search field from “car” to “carr,” Vim would have to scroll the document to bring the word “carrot” into view. And that is exactly what happens.

Suppose that we just want to check if the word “carrot” is present in the current document without moving our cursor. With the 'incsearch' option enabled, we would simply have to dial up the search prompt and then type as many characters of the word “carrot” as it takes to bring the first occurrence of the word into view. If the word is found, we can just press `<Esc>`, and we’ll end up right back where we started. No need to interrupt our train of thought.

Autocomplete the Search Field Based on Preview Match

In that last example, we executed the search command before completing the word “carrot.” That’s good enough if our intention was simply to move our cursor to the first match. But suppose that we needed our pattern to match the entire word “carrot”: for example, if we were planning to follow the search command with a substitute command.

Of course, we could simply type out the “carrot” in full. But here’s a handy shortcut: `<C-r><C-w>`. This autocompletes the search field using the remainder of the current preview match. If we used this command after entering “carr” into the search field, it would append “ot,” causing the match to encompass the entire word “carrot.”

Note that the `<C-r><C-w>` autocompletion is slightly brittle in this context. If you prefix your search with the `\v` item, then `<C-r><C-w>` will complete the entire word under the cursor (creating `\vcarrcarrot<CR>`, for example) instead of the remainder of the word. As long as you are searching for words and not patterns, the autocomplete feature of incremental search can be a nice little time-saver.

Tip 82

Count the Matches for the Current Pattern

This quick tip shows how to count the number of matches for a pattern.

There’s no way of making the search command report the number of matches in the current document. But here’s a command that does:

```
⇒ :%s///gn
```

```
◀ 117 matches on 65 lines
```

We’re actually calling the `:substitute` command, but the `n` flag suppresses the usual behavior. Instead of replacing each match with the target, it simply counts the number of matches and then echoes the result below the command line. By leaving the search field blank, we instruct Vim to use the current search pattern. The replacement field is ignored anyway (because of the `n` flag), so we can leave it blank too.

Whatever you do, don’t omit any of the `/` characters. Running `:%s//gn` would replace every match with “gn”! If it seems like too much typing, you might want to create a mapping for it. I’ll leave this as an exercise for you.