

Understand Vim Mappings and Create Your Own Shortcuts!

Get the right tools under your fingers.



Jonas B. Rossi

Follow

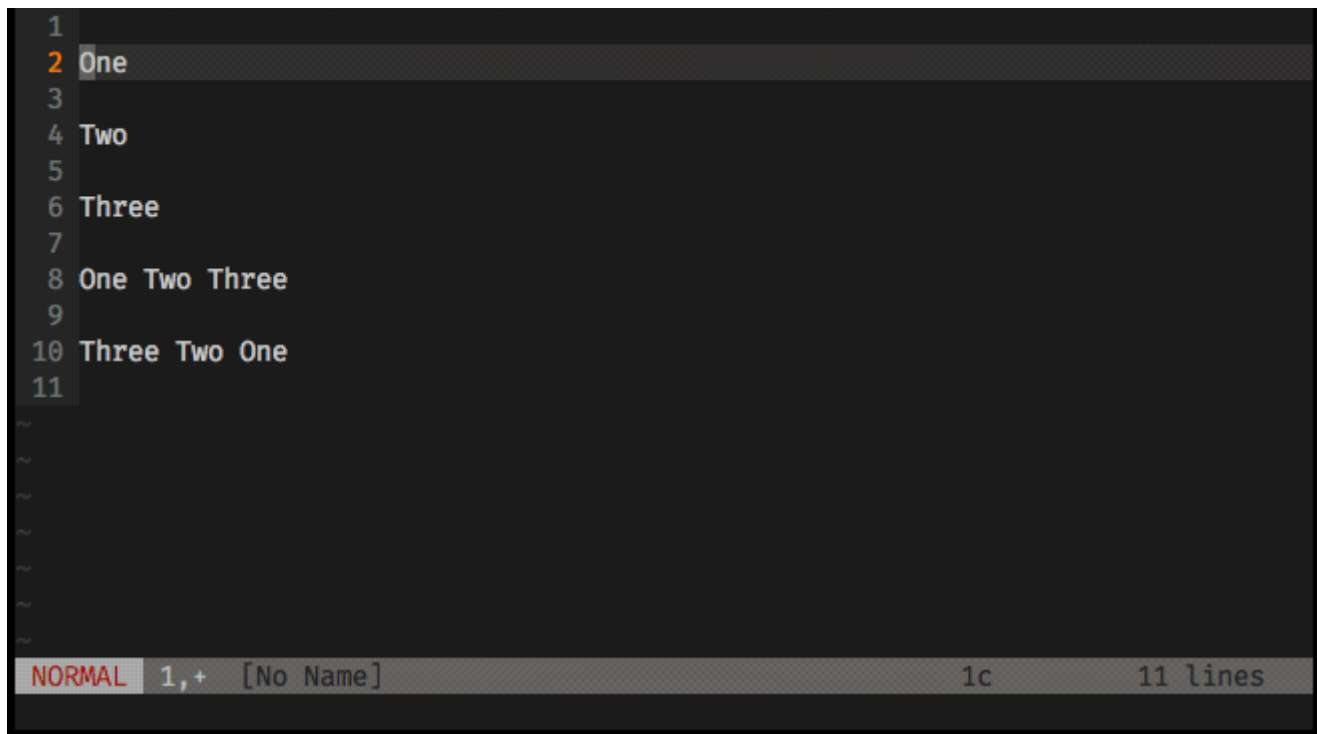
May 15, 2018 · 7 min read

```
nnooremap ,<space> :nohlsearch<CR>
```

When you start using vim you quickly find yourself searching for some small improvements on your keyboard commands. Maybe you want the **ESC** functionality a little bit closer to your fingers, or maybe you'd like to create a shortcut for a command sequence that you use very often.

There are good chances that you've checked stackoverflow looking for the easiest way to clear last search highlighting and found out that you need to type the command `:nohlsearch` or `:noh`, and then reading more, you get some tips about including a

mapping (a *shortcut*!) on your **.vimrc** file, to have the shortcut `,<space>` always under your fingers.



The screenshot shows a Vim editor window with a dark background. The first 11 lines of the file contain the following text: 1, 2 One, 3, 4 Two, 5, 6 Three, 7, 8 One Two Three, 9, 10 Three Two One, 11. The word 'One' on line 2 is highlighted in orange. The status bar at the bottom shows 'NORMAL', '1, +', '[No Name]', '1c', and '11 lines'.

When `,<space>` is pressed, the search highlighting is cleared

(from `:help mapping ...`)

Key mapping is used to change the meaning of typed keys. The most common use is to define a sequence of commands for a function key.

It's kinda easy to get going, to read the mapping that you've just pasted `nnoremap ,<space> :nohlsearch<CR>` and try to change it to something else... It's fun to play a little bit and to figure it out by yourself, and you should do it, but mapping is something very present on vim, and it's important to understand it.

As almost every single feature on vim, mapping is very powerful and it has tons of options and functionalities that handle many use cases but, as you might expect, on this post I'll try to expose the big picture of mapping, giving you a simple and clear understanding, bringing you the confidence to create your own mappings from scratch.

. . .

Understanding the key mapping structure

The structure is simple:

```
"          map-command {lhs} {rhs}

nnoimap ,<space> :nohlsearch<CR>
```

...and it's possible to use the special arguments:

```
"          map-command map-argument {lhs} {rhs}

nnoimap <silent> ,<space> :nohlsearch<CR>
```

map-arguments (one or more) are optional

Exploring the structure:

— map-command —

It's the first thing and, implicitly, the **map-command** defines:

- whether you're **adding new/removing/listing** map
- whether the mapping will be **recursive/non recursive**
- the **mode** where the mapping will be applied

From the example:

`nnoimap` defines that this is **adding a new mapping** to be applied in **normal mode** and it's **non recursive**.

— map-arguments —

It's **optional** and it must appear right after the **map-command**, you can use one or more arguments combined, and they can be used in any order.

From the example:

`<silent>` defines that when you execute this mapping pressing `,<space>`, the command `:nohlsearch` will not be echoed on the command line.

— {lhs} left-hand-side —

That's where you define the shortcut or the key(s) you're gonna use. It can be a single key like `,` or a sequence of keys like `,<space>`.

From the example:

`,<space>` is the shortcut, it's the sequence of keys that calls what you defined on **{rhs}**.

— {rhs} right-hand-side —

That's where you define what will your shortcut replace/execute when you press the key(s) defined at **{lhs}**. It can replace a key or execute a sequence of keys, it can call vim native commands (like previous example) or functions you created on vimscript language, things can be as complex as you wish.

From the example:

`:nohlsearch<CR>` is what will be executed, vim will type `:nohlsearch` followed by the **Enter** key that is defined by `<CR>`.

Now... All together!

```
noremap <silent> ,<space> :nohlsearch<CR>
```

... basically tells vim:

*Everytime I press `,` then `<space>` in **normal mode**, please type the following command `:nohlsearch` and please finish with **Enter** (`<CR>`) to execute the command. But...hey!*

The main commands:

1	COMMANDS			
2				
3	Add	Add	Remove	MODES IT WORKS
4	(or list)	Non-Recurs.	Mapping	
5	Mapping	Mapping		
6				
7	:map	:noremap	:unmap	Normal, Visual, Select, Operator-pending
8	:nmap	:nnoremap	:nunmap	Normal
9	:vmap	:vnoremap	:vunmap	Visual and Select
10	:smap	:snoremap	:sunmap	Select
11	:xmap	:xnoremap	:xunmap	Visual
12	:omap	:onoremap	:ounmap	Operator-pending
13	:map!	:noremap!	:unmap!	Insert and Command-line
14	:imap	:inoremap	:iunmap	Insert
15	:lmap	:lnoremap	:lunmap	Insert, Command-line, Lang-Arg
16	:cmap	:cnoremap	:cunmap	Command-line
17	:tmap	:tnoremap	:tunmap	Terminal-Job

Take a minute to stare at it and the syntax will start to make sense

At the first column you'll notice it's written *(or list)*, it's because the same command if used alone will list the mappings existent for that specific mode.

Special Arguments

The special arguments can be used in any order, but they must appear right after the map command, before any other arguments.

They are:

<buffer>

Keep the mapping restricted to work only on the current buffer where the command is being executed. It allows you to use the same key(s) with a different mapping on another buffer.

This argument can also be used with the command to remove a mapping from the current buffer.

<nowait>

It basically tells vim if you type the characters defined the **{lhs}**, vim must not wait for any more characters to be typed and it must execute this mapping right away!

It's important to know that if you do type more characters, it will be used and thrown after the mapping execution.

`<silent>`

As shown before, it prevents the mapping to be echoed on the command line.

`<special>`

It's used to indicate that you'll use the `<>` notation to define a mapping for special keys (example `:map <special> <F12> /Header<CR>`)

`<script>`

We're not going too deep on this one, you'll probably do that when you start to create your own plugins and scripts. Basically It's used to avoid interferences on mappings that are present on scripts.

`<expr>`

It defines that the `{lhs}` will be an expression, and this expression will be evaluated to obtain the `{rhs}`.

`<unique>`

Used to avoid overwriting a mapping that already exists for that specific key(s).

Now you know most of the basics that will improve your mapping skills but, if you hit some unexpected behaviour or if you feel that some mapping doesn't look that good, take an hour to read ‘`:help mapping`’ and try the examples, there's a few more magics!

. . .

Attention! Some advices before you start creating your own mappings!

- It makes sense to create your mappings and save it on your **.vimrc** file to have it always on hands.
- It's recommended to **always use non recursive mappings**, unless there's this special case where you need the recursive behavior.
- Vim has commands for almost every key so, when creating mappings, you must make good choices. Avoid changing the behavior of vim's most important keys/commands. Get the best orientation at `:help map-which-keys`.
- If you need some help about how to use some specific keys at **{lhs}** or **{rhs}**, check `:help key-notation` and `:help map-special-chars`.
- Improve your workflow but don't exaggerate! First, create mappings only for commands that are slowing your workflow, or for that key you need very often and it's hard to reach on your keyboard model. After being more used to it, or when you start to create some vim scripts and plugins, you'll probably dive a little bit deeper on mappings.

. . .

References:

- Vim Documentation (`:help`)
- "Mastering Vim Quickly" book a by Jovica Ilic. It's an awesome book, make sure you visit the website and subscribe to get free chapters by email.
- You can practice mappings from the very easiest ones to more complex ones, following the exercices on chapters 3, 4, 5 and 6 from the free ebook "Learn Vimscript the Hard Way", (by Steve Losh).

Footnotes:

- *If you have any doubts or tips I'd appreciate to know and discuss it on the comments section.*
- *Do you have any other vim tips and want to publish it here? Send me and email at 'vimdrops AT pm DOT me' with your medium username and I'll invite you as a writer!*

- *As English is not my first language, I apologize for errors. Corrections are welcome at ‘vimdrops AT pm DOT me’.*

[Vim](#)[Code Editor](#)[Programming](#)[Coding](#)[Learning To Code](#)[About](#)[Help](#)[Legal](#)