Vim Key Mapping

Bill Odom (billodom@gmail.com)
St. Louis UNIX Users Group

What?

- Maps bind a series of keystrokes to a key sequence
- Better for defining behavior (vs. abbreviations)
- Better for long-term customization (vs. macros)
- Can be defined across all of Vim's modes
- **Fundamental to customizing Vim**

Abbreviations are most useful for content expansion, like correcting typos and inserting snippets/templates.

By "long-term customization," I mean things you want to use beyond your current session. Macros are great, but if you have one you like, convert it to a map and stick it in your .vimrc.

Map Command Structure

{cmd} {attr} {lhs} {rhs}

map command attributes left-hand side right-hand side

Map Command Structure

```
" Example map command
map <silent> <F2> :echo 'Hello!'<CR>
```


<silent> is an attribute, and <F2> is the key being mapped.

Map Commands

`omap`	`nmap`
`cmap`	`imap`
`lmap`	`vmap`
`map`	`xmap`
`map!`	`smap`

These are the mapping commands that Vim supports, but you probably won't use all of them, and you usually don't want to use them in this form.

Map Commands

```
nnoremap
inoremap
vnoremap
xnoremap
snoremap
onoremap
cnoremap
```


These are the ones you'll use most of the time.

I've left out :map and :map! since Vim's mode-specific commands are more precise and understandable. The :lmap command is missing, too, since it's very rarely used. (See :help language-mapping if you're curious.)

But what's with the "nore" that's shown up in the middle of the command names?

Map Commands

```
nnoremap
inoremap
vnoremap
xnoremap
snoremap
onoremap
cnoremap
```


The usual Vim map commands (without “nore” in the middle) create maps that allow **remapping**. That is, any mapped keystrokes on the right-hand side get expanded when the mapped key sequence (the left-hand side) is pressed. This is often very useful, but it can be a problem if it happens unexpectedly.

The “nore” -- a.k.a. “non-remapping” or “non-recursive” -- versions don’t allow remapping; keys mentioned on the right-hand side have their default, built-in meaning. I use the non-remapping commands as a safe default, and only use the remapping versions when I specifically **want** the right-hand side of a map to expand other embedded maps. (We’ll see examples of this in the Sample Maps.)

Map Attributes

<silent>
<buffer>
<expr>
<unique>
<script>
<special>

<silent> Map will not be echoed on the command line
<buffer> Map is local to the current buffer
<expr> Right-hand side is result of an expression evaluated each time the map is invoked
<unique> Command fails if the left-hand side is not unique
<script> Remapping allowed, but only for <SID> maps local to a script
<special> Allows use of symbolic key names, even when cpoptions doesn't contain <

:help :map-argument

Available Keys

- Function keys and shifted function keys
- **<Leader>** and **<LocalLeader>** sequences (especially in plugins and scripts)

This is why <F2> is such a popular example key in map documentation. (<F1> triggers :help.)

Plugins and scripts often define maps with <Leader> (or <LocalLeader> for buffer-local maps). The user can specify the leader keys with

```
:let mapleader = {key}
:let maplocalleader = {key}
```

This makes it easy to avoid conflicts between a user's personal maps and plugin/script maps, just by selecting the appropriate <Leader> and <LocalLeader>.

Sorta-Available Keys

- Alt / Option / Meta Keys (safest in the GUI)
- Commands you don't use
- Command "synonyms" you don't use
- Comma
- Semicolon
- Underscore

Console Vim is at the mercy of the terminal when it comes to the keys it can see, so be sure to test any alt/option/meta key sequences on the platforms you use.

GUI Vim processes keystrokes natively, so there are many more keys available. Go nuts.

Control+N / Control+P is what I mean by a command synonym.

Comma, semicolon, and underscore all have predefined behaviors in Vim, but (a) they all have reasonable synonyms, and (b) lots of people don't use them anyway.

Finding Keys to Map

```
:help map-which-keys
```

```
:help index.txt
```

```
:help {key} Ctrl D
```

Displaying Maps

:nmap

:nmap *{key}*

:mkvimrc ~/scratch.vim

All the map commands work this way. :nmap is just an example.

I never actually use :mkvimrc to make .vimrc, since I'm pretty attached to the one I've got. Instead, it's useful for capturing a current Vim session's options and settings into a file, then cherry-picking what you want from it.

Removing Maps

`:unmap` *{key}*

`:verbose map` *{key}*

:unmap isn’t permanent. It only affects the current session.

:verbose map doesn’t remove a map, but it’s great when you need to know where a map is defined (so you can, for example, remove it).

Sample Maps

```
" Overload Control+L to also clear
" search highlighting
"
nnoremap <C-L>  :nohlsearch<CR><C-L>
```


:nohlsearch lets you keep search highlighting enabled, but turn it off for the most recent search. It's better than toggling 'hlsearch' on and off with :set.

This is a good example of where "nore" avoids creating a recursive map. With just :nmap, hitting Control+L turns off highlighting (which is good), but then **invokes itself again** (which is bad).

The :nnoremap version does the right thing, turning off highlighting, then invoking the built-in Control+L to redraw the screen.

Sample Maps

```
" Toggle screen wrapping of long lines,
" then display value of ‘wrap‘ option
"
nnoremap <Leader>w
\   :set invwrap<Bar>set wrap?<CR>
```


Notice the <CR> at the end of the command line. Without it, the cursor is left sitting on the Ex command line, just after `set wrap?`.

The \ is how Vim does line continuation. I’m only using it here to make the command fit on screen comfortably. It’s not part of the map.

I like and use the “toggle-then-display” idiom a lot.

Sample Maps

```
" Make up/down arrows move by display
" lines instead of physical lines
"
nnoremap <Up>     gk
nnoremap <Down>   gj
```


This makes dealing with soft-wrapped text **so** much easier, and lets the up/down arrow keys do something useful.

I don't recommend remapping j and k to do this, since that's likely to break some plugins.

Sample Maps

```
" Reformat current paragraph
"
nnoremap Q  gqap
```


This is a good way to take a chunk of mangled text and tidy it up. gq respects the current textwidth setting, and understands > and >> email formatting. It’s not as smart as fmt or par, but it’s pretty good.

Sample Maps

```
" Indent/unindent highlighted block
" (and maintain highlight)
"
vnoremap <Tab>     >gv
vnoremap <S-Tab>   <gv
```


Very useful.

Sample Maps

```
" Disable paste-on-middle-click
"
inoremap <MiddleMouse>  <Nop>
```


A single <Nop> is how you disable Vim features you don’t want. I don’t want to unexpectedly paste my clipboard into random files because I accidentally pressed my scrollwheel too hard.

Sample Maps

```
" Center display line after searches
"
nnoremap n   nzz
nnoremap N   Nzz
nnoremap *   *zz
nnoremap #   #zz
nnoremap g*  g*zz
nnoremap g#  g#z
```


Great for noisy searches when you've lost your cursor -- just look in the middle of the screen.

Sample Maps

```
" Edit vimrc in new tab
"
nnoremap ,ev  :tabedit $MYVIMRC<CR>
```


A good example of how tab pages can be used to quickly get in and get out of something you need to do, but that isn't the main thing you're working on at the moment. I update my .vimrc all the time, but I hate to mess up my current window arrangement just to tweak my editor configuration.

Sample Maps

```
" Delete to end of line, bash-style
"
inoremap <C-K> <C-O>D
```


You get used to something in one environment, you want it in another.

Sample Maps

```
" Control+Up/Down move lines & selections up and down.
" (Based on http://vim.wikia.com/wiki/VimTip646)
"
" Define maps for Normal and Visual modes, then re-use
" them for Insert and Select modes.
"
nnoremap <silent>  <C-Up>     :move -2<CR>
nnoremap <silent>  <C-Down>   :move +<CR>
xnoremap <silent>  <C-Up>     :move '<-2<CR>gv
xnoremap <silent>  <C-Down>   :move '>+<CR>gv
imap     <silent>  <C-Up>     <C-O><C-Up>
imap     <silent>  <C-Down>   <C-O><C-Down>
smap     <silent>  <C-Up>     <C-G><C-Up><C-G>
smap     <silent>  <C-Down>   <C-G><C-Down><C-G>
```


I can’t count the number of times I use these every day. Highly recommended.

This contains a good example of when **not** to use the “nore” versions of map commands. The four :imap and :smap commands at the bottom allow remapping, since they depend on the Normal and Visual maps above them. If I’d used :inoremap and :snoremap instead, they wouldn’t work.