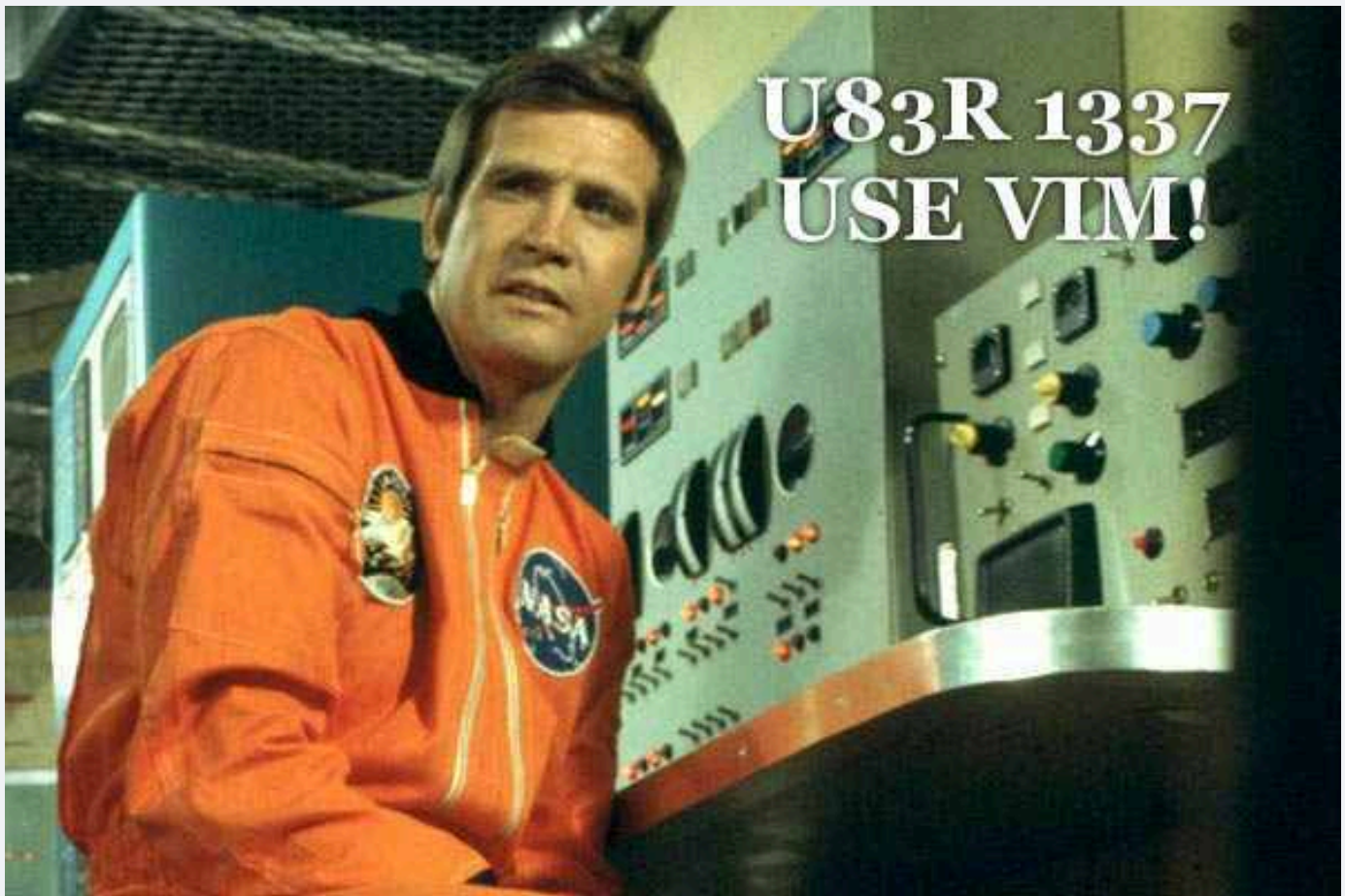




Learn Vim Progressively



tl;dr: You want to teach yourself vim (the best text editor known to human kind) in the fastest way possible. This is my way of doing it. You start by learning the minimal to survive, then you integrate all the tricks slowly.

Vim the Six Billion Dollar editor

| Better, Stronger, Faster.

Learn vim and it will be your last text editor. There isn't any better text editor that I know of. It is hard to learn, but incredible to use.

I suggest you teach yourself Vim in 4 steps:

1. Survive
2. Feel comfortable
3. Feel Better, Stronger, Faster
4. Use superpowers of vim

By the end of this journey, you'll become a vim superstar.

But before we start, just a warning. Learning vim will be painful at first. It will take time. It will be a lot like playing a musical instrument. Don't expect to be more efficient with vim than with another editor in less than 3 days. In fact it will certainly take 2 weeks instead of 3 days.

1st Level – Survive

0. Install vim
1. Launch vim
2. DO NOTHING! Read.

In a standard editor, typing on the keyboard is enough to write something and see it on the screen. Not this time. Vim is in *Normal* mode. Let's go to *Insert* mode. Type the letter *i*.

You should feel a bit better. You can type letters like in a standard editor. To get back to *Normal* mode just press the ESC key.

You now know how to switch between *Insert* and *Normal* mode. And now, here are the commands that you need in order to survive in *Normal* mode:

- *i* → *Insert* mode. Type ESC to return to *Normal* mode.
- *x* → Delete the char under the cursor
- *:wq* → Save and Quit (*:w* save, *:q* quit)
- *dd* → Delete (and copy) the current line
- *p* → Paste

Recommended:

- *h j k l* (highly recommended but not mandatory) → basic cursor move (*← ↓ ↑ →*). Hint: *j* looks like a down arrow.
- *:help <command>* → Show help about *<command>*. You can use *:help* without a *<command>* to get general help.

Only 5 commands. That is all you need to get started. Once these command start to become natural (maybe after a day or so), you should move on to level 2.

But first, just a little remark about *Normal mode*. In standard editors, to copy you have to use the Ctrl key (Ctrl-c generally). In fact, when you press Ctrl, it is as if all of your keys change meaning. Using vim in normal mode is a bit like having the editor automatically press the Ctrl key for you.

A last word about notations:

- instead of writing Ctrl-λ, I'll write <C-λ>.
- commands starting with : end with <enter>. For example, when I write :q, I mean :q<enter>.

2nd Level – Feel comfortable

You know the commands required for survival. It's time to learn a few more commands. These are my suggestions:

1. Insert mode variations:

- a → insert after the cursor
- o → insert a new line after the current one
- O → insert a new line before the current one
- cw → replace from the cursor to the end of the word

2. Basic moves

- 0 → go to the first column
- ^ → go to the first non-blank character of the line
- \$ → go to the end of line
- g_ → go to the last non-blank character of line
- /pattern → search for pattern

3. Copy/Paste

- P → paste before, remember p is paste after current position.
- yy → copy the current line, easier but equivalent to ddP

4. Undo/Redo

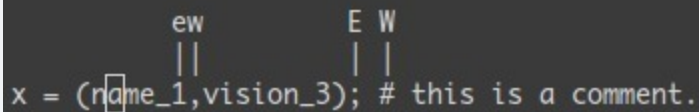
- u → undo
- <C-r> → redo

5. Load/Save/Quit/Change File (Buffer)

- :e <path/to/file> → open
- :w → save
- :saveas <path/to/file> → save to <path/to/file>
- :x, ZZ or :wq → save and quit (:x only save if necessary)
- :q! → quit without saving, also: :qa! to quit even if there are modified hidden buffers.
- :bn (resp. :bp) → show next (resp. previous) file (buffer)

Take the time to learn all of these command. Once done, you should be able to do every thing you are able to do in other editors. You may still feel a bit awkward. But follow me to the next level and you'll see why vim is worth the extra work.

3rd Level – Better. Stronger. Faster.



```
x = (name_1, vision_3); # this is a comment.
```

Now let's talk about very efficient moves:

- % : Go to the corresponding (, {, [.
- * (resp. #) : go to next (resp. previous) occurrence of the word under the cursor

Believe me, the last three commands are gold.

Faster

Remember about the importance of vi moves? Here is the reason. Most commands can be used using the following general format:

<start position><command><end position>

For example : 0y\$ means

- 0 → go to the beginning of this line
- y → yank from here
- \$ → up to the end of this line

We also can do things like ye, yank from here to the end of the word. But also y2/foo yank up to the second occurrence of "foo".

But what was true for y (yank), is also true for d (delete), v (visual select), gU (uppercase), gu (lowercase), etc...

4th Level – Vim Superpowers

With all preceding commands you should be comfortable using vim. But now, here are the killer features. Some of these features were the reason I started to use vim.

Move on current line: 0 ^ \$ g_ f F t T , ;

- 0 → go to column 0
- ^ → go to first character on the line
- \$ → go to the last column
- g_ → go to the last character on the line
- fa → go to next occurrence of the letter a on the line. , (resp. ;) will find the next (resp. previous) occurrence.
- t, → go to just before the character ,.
- 3fa → find the 3rd occurrence of a on this line.

- F and T → like f and t but backward.

```

0  ^      fi    t)    4fi      g_  $
|  |      |     |     |       |   |
x = (name_1,vision_3); #this is a comment.

```

A useful tip is: dt" → remove everything until the " .

Zone selection <action>a<object> or <action>i<object>

These command can only be used after an operator in visual mode. But they are very powerful. Their main pattern is:

<action>a<object> and <action>i<object>

Where action can be any action, for example, d (delete), y (yank), v (select in visual mode). The object can be: w a word, W a WORD (extended word), s a sentence, p a paragraph. But also, natural character such as ", ',), },].

Suppose the cursor is on the first o of (map (+) ("foo")).

- vi" → will select foo .
- va" → will select "foo" .
- vi) → will select "foo" .
- va) → will select ("foo") .
- v2i) → will select map (+) ("foo")
- v2a) → will select (map (+) ("foo"))

```

----- v2a)
~~~~~ v2i)
(map (+) ("foo"))
      --- vi"
      +++++ va" and vi)
      ===== va)

```

Select rectangular blocks: <C-v> .

Rectangular blocks are very useful for commenting many lines of code.
Typically: 0<C-v><C-d>I-- [ESC]

- ^ → go to the first non-blank character of the line
- <C-v> → Start block selection
- <C-d> → move down (could also be jjj or %, etc...)
- I-- [ESC] → write -- to comment each line

```
-- Screen size
type Size = (Int,Int)
type Point = (Int,Int)
type TimePoint = (Int,Int,Int)
screenSize = (25,79)

type Drawer = Size -> Int -> [Float]

fabs x | x < 0 = -x
```

Note: in Windows you might have to use <C-q> instead of <C-v> if your clipboard is not empty.

Completion: <C-n> and <C-p>.

In Insert mode, just type the start of a word, then type <C-p>, magic...

```
- Lovcraft
- LyX
- LaTeX
- XeLaTeX
```

Now I'll type: X<C-p>, L<C-n><C-n>.

```


```

```
~
```

```
~
```

Macros : qa do something q, @a, @@

qa record your actions in the *register* a. Then @a will replay the macro saved into the register a as if you typed it. @@ is a shortcut to replay the last executed macro.

Example

On a line containing only the number 1, type this:

- qaYp<C-a>q →
 - qa start recording.
 - Yp duplicate this line.
 - <C-a> increment the number.
 - q stop recording.
- @a → write 2 under the 1
- @@ → write 3 under the 2
- Now do 100@@ will create a list of increasing numbers until 103.



Visual selection: v, V, <C-v>

We saw an example with <C-v>. There is also v and V. Once the selection has been made, you can:

- J → join all the lines together.
- < (resp. >) → indent to the left (resp. to the right).
- = → auto indent


```
// AUTOINDENT with = example => S-V$%=
// S-V => Visual select lines
// $ => go to end of line
// % => go to end of function (corresponding { )
// = => Auto indent the selection
int allcases ( char prefix[], int n, char *list[] ) {
char *c;
char **next_list = list;
ylog(prefix);
19,1 26%
```

Add something at the end of all visually selected lines:

- <C-v>
- go to desired line (jjj or <C-d> or /pattern or % etc...)
- \$ go to the end of the line
- A, write text, ESC.

```
#!/usr/bin/env zsh
echo "Hello"
echo "I will add something"
echo "in the end of each line"
echo "Lets do it"
echo "C-vG$A >&2[ESC]"
~
~
2,0-1 All
```

Splits: :split and vsplit.

These are the most important commands, but you should look at :help split.

- :split → create a split (:vsplit create a vertical split)
- <C-w><dir> : where dir is any of hjkl or ←↑→ to change the split.
- <C-w>_ (resp. <C-w>|) : maximise the size of the split (resp. vertical split)
- <C-w>+ (resp. <C-w>-) : Grow (resp. shrink) split

```
enddiv
```

```
[Vim] the Six Billion Dollar editor
```

```
> Better, Stronger, Faster.
```

```
Learn [vim] and it will be your last text editor.  
There isn't any better text editor I know.  
Hard to learn, but it will pay a billion times.
```

```
:q
```

```
23,0-1
```

```
7%
```

Conclusion

That was 90% of the commands I use every day. I suggest that you learn no more than one or two new commands per day. After two to three weeks you'll start to feel the power of vim in your hands.

Learning Vim is more a matter of training than plain memorization. Fortunately vim comes with some very good tools and excellent documentation. Run vimtutor until you are familiar with most basic commands. Also, you should read this page carefully: `:help usr_02.txt`.

Then, you will learn about `!`, folds, registers, plugins and many other features. Learn vim like you'd learn piano and all should be fine.

If you liked this article, there is a follow up: [Vim as IDE](#)

[RSS](#) · [Tweet](#) · [FB](#)

[These social sharing links preserve your privacy](#)

[Home](#) | [Blog](#) | [Softwares](#) | [About](#)

[↑ Top ↑](#)

Published on 2011-08-25

Follow [@yogsototh@ieji.de](#)

[Yann Esposito](#)©

Done with [Vim](#) [spacemacs](#) & [nanoe](#) [Hakyll](#)