



Introduction to vim

Peleg Sapir

exocad GmbH

August 26, 2021






















The Most Important Command

Before we start, let's settle down the age-old question:



The Most Important Command

The most common exit and save commands:

Command	Keys
Simple exit	  
Save	  
Save and exit	   
Exit without save	   
Save and override	   
Command history	  

Just to confuse ;)

Vim Modes

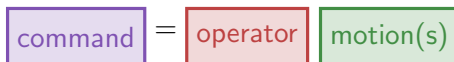
Vim has three **modes**:

Name	Function	Key(s)
Normal	navigation and text editing	Default/ Esc
Insert	inserting text	i
Normal	highlighting text/rows/blocks	v / V / Ctrl + v

Vim Grammar

Generally speaking, vim commands can be structure in different forms (here refered to as "rules" (CITE)).

The most simple rule is

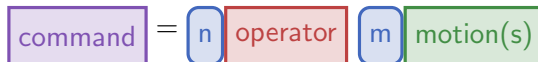


Example













Command	Syntax	Keys
Delete word	<div>del</div> <div>word</div>	<div>d</div> <div>w</div>
Copy until 'A'	<div>copy</div> <div>until 'A'</div>	<div>y</div> <div>f</div> <div>A</div>

Vim Grammar

Operators and motions can be preceded by repetitions, i.e.



Example

Command	Syntax	Keys
3× delete word	  	  
Delete 3 words	  	  

List of Operators

Main vim operator keys:

Key	Func.	Key	Func.
y	copy	c	change (delete + insert)
d	delete	x	delete single
p	paste	P	paste before
u	undo	Ctrl + r	redo
.	repeat action	;	repat motion




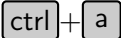
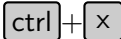


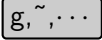
List of Motions

Main vim motion (navigation) keys:

Key	Func.	Key	Func.
<code>w</code>	next word	<code>b</code>	beginning of word
<code>e</code> / <code>g</code> <code>e</code>	end of word / prev	<code>n</code>	next find
<code>n</code>	next find	<code>N</code>	prev find
<code>f</code> <code>α</code>	next α	<code>F</code> <code>α</code>	prev α
<code>t</code> <code>α</code>	before next α	<code>T</code> <code>α</code>	after prev α

Operators without Motions

Some operators don't need motions:

Key	Func.	Key	Func.
Visual mode			
	make uppercase		make lowercase
	switch case		
Normal mode			
	increment int		decrement int
	make uppercase		make lowercase
	switch case		





















Operators without Motions

Some more operators without motions:

Key	Func.	Key	Func.
Normal mode			
<code>u</code>	undo	<code>Ctrl+r</code>	redo
<code>d</code> <code>d</code>	delete line	<code>c</code> <code>c</code>	replace line

Motions without Operators

Some motions aren't actually relevant to operators and repetitions:

Key	Func.	Key	Func.
 / 	left	 / 	down
 / 	up	 / 	right
	start of line		end of line
	start of line (text)		matched paranthesis
	top of view		bottom of view
	middle of view		bottom of view
 + 	scroll screen up	 + 	scroll screen down

Modifiers

The following two keys are **Modifiers**:

Key	Meaning
<code>i</code>	Inside an object
<code>a</code>	Around an object

Example

Command	Keys
Copy around '[]'	<code>y</code> <code>a</code> <code>[</code>
Delete inside '{}'	<code>d</code> <code>i</code> <code>{</code>

The Command Buffer

Marks

Points in a file can be marked for later use. There are 52 possible custom marks: all lowercase letters + all uppercase letters.

Adding a mark *a* is done by

`m` `a`

Navigating to a mark *a* is done by `'` `a` .

The combination `'` `a` will jump to the start of the line where the mark is.

Macros

A **macro** is a recording of a set of operations, which can be repeated as many times as needed.

A macro *a* is recorded by typing `q` `a` [set of operations] `q` .

A macro *a* is called by `@` `a` .

Registers

Find, Search & Replace, Regex

Searching for a string is done by pressing `/` (`?` for backwards search), and entering a regex-like search query.

Moving between matches can be done via `n` for forward search, and `N` for back search.

The word currently under the cursor can be searched by `*` for a forward search, and `#` for backwards search.

As with commands, the search history is searchable via `↑` and `↓`.

Find, Search & Replace, Regex

Replacing an expression is done by `:s` :

- `:s/foo/bar/g` changes each `foo` to `bar` in current line
- `:%s/foo/bar/g` change globally
- `:'<,>s/foo/bar/g` in visual mode: change in selected lines
- `:'<,>s/foo/bar/g` in visual mode: change in selected lines
- `:$s/foo/bar/g` change from here to end of file
- `:+Ns/foo/bar/g` change current line + N more lines

Find, Search & Replace, Regex

continuing:

- `: \%s/foo/bar/gc` change with confirmation per each change
- `: \%s/foo/bar/gci` change case insensitive
- `:g/^baz/s/foo/bar/g` change in lines starting with `baz`
- `:s//bar/g` use last searched pattern
- `: \%s/foo/<c-r><c-w>/g` replace with the word under the cursor (`Ctrl`+`r` `Ctrl`+`w`)
- `: \%s/foo/<c-r>a/g` replace with the content of register `a`
- `: \%/foo/\\=ae/g` replace using arithmetic expression `ae`

Splits