



# 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:



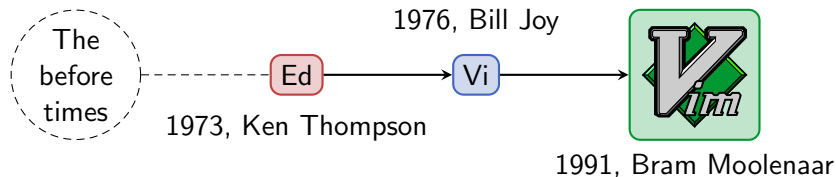
# The Most Important Command

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 ;)

# Brief History



# Vim Modes

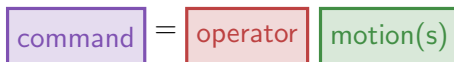
Vim has three **modes**:

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

# 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.

$$\boxed{\text{command}} = \boxed{n} \boxed{\text{operator}} \boxed{m} \boxed{\text{motion(s)}}$$

## Example

Command	Syntax			Keys
3× delete word	<div>3</div>	<div>del</div>	<div>word</div>	<div>3</div> <div>d</div> <div>w</div>
Delete 3 words	<div>del</div>	<div>3</div>	<div>word</div>	<div>d</div> <div>3</div> <div>w</div>

# 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






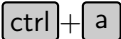
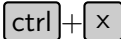


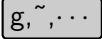
# List of Motions

Main Vim motion (navigation) keys:

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

# Operators without Motions

Some operators don't need motions:

Key	Func.	Key	Func.
<b>Visual mode</b>			
	make uppercase		make lowercase
	switch case		
<b>Normal mode</b>			
	increment int		decrement int
	make uppercase		make lowercase
	switch case		





















# Operators without Motions

Some more operators without motions:

Key	Func.	Key	Func.
<b>Normal mode</b>			
<code>u</code>	undo	<code>Ctrl+r</code>	redo
<code>d d</code>	delete line	<code>c 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
i	Inside an object
a	Around an object

## Example

Command	Keys
Copy around '[]'	y a [
Delete inside '{}'	d i {

# The Vim Command Line

Pressing `:` in normal mode summons the internal Vim command line (here referred to as *vc/*). Many, many different things can be done using this buffer - some of which will be shown later.

Any shell command can be executed via the *vc/*, by prepending it with `!`, e.g. `:!ls` will run `ls` in the shell that executed Vim.

# 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.

# Marks

The lowercase marks `a` - `z` exist only in currently open files (buffers), while the uppercase marks `A` - `Z` can be jumped to even in a closed file.

The command `:marks` gives a list of all marks.

Example motions and operations using marks:

Key	Function	Key	Function
<code>m</code> <code>a</code>	set <code>a</code>		
<code>'</code> <code>a</code>	go to line of <code>a</code>	<code>'</code> <code>a</code>	go to <code>a</code>
<code>d</code> <code>'</code> <code>a</code>	del until line of <code>a</code>	<code>d</code> <code>'</code> <code>a</code>	del until <code>a</code>



# Marks

Deleting the mark `a` is done via `:delmarks a`. One can delete a range of marks by `:delmarks a-d`, or a list of marks by e.g. `:delmarks afgv` (for the marks `a`, `f`, `g` and `v`). `:delmarks!` will delete all lowercase marks `a - z`.

There are several special marks:

- `.` : last change in current buffer.
- `"` : last exited current buffer.
- `0` : edited position in last edited file.
- `1 - 9` : edited position in previous `n`-th file.

# 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`
- `:%s/foo/\\=ae/g` replace using arithmetic expression `ae`

# Splits

# Useful Links for Further Reading