



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:



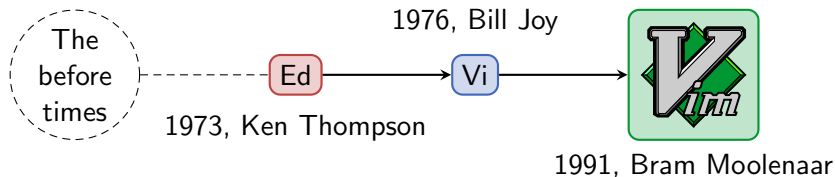
The Most Important Command

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

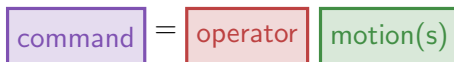
Vim has three **modes**:

| Name | Function | Key(s) |
|--------|-------------------------------|--|
| Normal | navigation and text editing | Default/ 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" [1]).

The most simple rule is

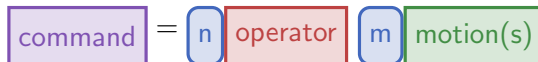


Example













| 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 |




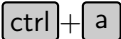
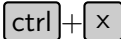


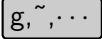
List of Motions

Main Vim motion (navigation) keys:

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

Registers hold yanked (copied) text. Similar to marks, they can be named `a` - `z` , but only using lowercase letters.

`"ay` will copy the selected text to the register `a` , while `"ap` will paste the content of `a` . All other relevant actions are possible too.

To display the content of registers `a` , `b` and `c` , use the command `:reg a b c` .

The unnamed register `"` holds the last copied/yanked text, i.e.

`"p` = `p` .

Registers

There are four special registers:

- `.` : last inserted text.
- `\%` : current file name.
- `:` : most recently executed command.
- `\#` : name of last (alternate) file (try `:h alternate-file` for more info).

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

A Vim buffer can be split horizontally by `:sp`, and vertically by `:vsp`. Adding a file name opens that file in the split.

Navigating between splits is done by `ctrl+w`+arrow.

The order of splits can be changed by `ctrl+w` `ctrl+r`.

Increasing a split size is done by `ctrl+w` `α` `+`, decreasing by `-` instead of `+`.

To make a horizontal split full width: `ctrl+w` `-` (`|` for vertical split). To make splits equal size: `ctrl+w` `=`.

Advanced Usage

More advance topics, such as: plugins, `.vimrc` config, vim scripts, custom syntax highlighting and colors, context-aware typing and more - in a future presentation.

Useful Sources for Further Reading

Useful Sources:

- Vim tutor: `:!vimtutor` - follow it!
- Vim tips wiki:
https://vim.fandom.com/wiki/Vim_Tips_Wikitext.
- Vim homepage: <https://www.vim.orgtext>.

Cited source:



Tom Cammann. *Vim Grammar*. Jan. 30, 2013. URL:
<https://takac.github.io/2013/01/30/vim-grammar/>.