

VIM basics

Piotr Grabarski

Warszawa 2020-04-19

Plan prezentacji

- 1 Wstęp
- 2 Podstawy
- 3 Zaawansowana edyc
- 4 Konfigurac
- 5 Język VIM'a poza edytorem
- 6 Zakończenie

1. Wstęp

VIM - Ulepszona wersja edytora VI

- wysoce konfigurowalny
- szybki
- zajmuje mało pamięci
- uniwersalny - jest dostępny na każdym unixowym systemie
- używa języka spójnego z innymi programistycznymi narzędziami

Trzy pytania

- Czy każdy zna VIM'a?
- Kto lubi VIM'a?
- Kto używa VIM'a?

2. Podstawy

Podstawy poruszania się i edycji tekstu

- Tryby
- Poruszanie się po pliku
- Operatory
- Obiekty tekstowe
- Liczniki
- Poruszanie się po tekście - motions
- Połączmy je
- Inne skróty

2.1 Tryby

Normal mode - *Esc*

Najważniejszy, domyślny tryb. Wykonujemy w nim większość akcji

Visual mode - *v*, *V*, *ctrl + v*

Tryb służący do zaznaczania oraz wykonywania akcji na zaznaczonym tekście

Insert mode - *i*, *I*, *A*, *a*, *c*, *C*

Tryb służący do pisania - odblokowuje “normalną” klawiaturę

Command mode - *:*

Tryb ten służy do wywoływania komend we wbudowanym wierszu poleceń

Select oraz *ex*

Rzadko używane tryby - nie wymagane do średnio-zaawansowanej edycji plików

2.2 Poruszanie się po pliku

h, j, k, l

Można używać strzałek, ale nie jest to zalecane i wydajne, bo znajdują się one daleko od centrum klawiatury.

- **h** - w lewo
- **j** - w dół
- **k** - w górę
- **l** - w prawo

Inne skróty klawiszowe

- **gg** i **G** - początek i koniec pliku
- **ctrl + Y** - przewijanie w górę o jedną linię
- **ctrl + U** - przewijanie w górę o połowę okna
- **ctrl + B** - przewijanie w górę o całe okno
- **ctrl + E** - przewijanie w dół o jedną linię
- **ctrl + D** - przewijanie w dół o połowę okna
- **ctrl + F** - przewijanie w dół o całe okno

2.3 Operator

Operatory pozwalają na edycję tekstu poza trybem *Insert*

Podstawowe operatory

- **d** - “delete” - usuń
- **c** - “change” - usuń i wejdź do trybu pisania
- **y** - “yank” - skopiuj do rejestru

W Vimie musimy uwzględniać wielkość liter

- **D** - usuń całą linię
- **C** - usuń całą linię i wejdź do trybu pisania
- **Y** - skopiuj całą linię do rejestru
- **gu** - zmień z wielkiej litery na małą
- **gU** - zmień z małej litery na wielką

2.4 Obiekty tekstowe

W Vimie musimy patrzeć na plik jak na zestaw obiektów
“a” oznacza “a lub all”, czyli cały obiekt tekstowy
“i” oznacza “inner”, czyli jedynie wewnątrz obiektu

Podstawowe obiekty tekstowe

- **aw, iw** - “word” - oznacza ciąg znaków zakończony znakiem specjalnym np. “.(),-”
- **aW, iW** - “Word” - oznacza ciąg znaków zakończony spacją
- **ap, ip** - “paragraph” - Oznacza tekst nieprzedzielony pustą linią
- **a{, i{** - obszar wewnątrz nawiasów klamrowych
- **a(, i(** - analogicznie
- **a[, i[** - analogicznie
- **a“, i”** - analogicznie
- **a', i'** - analogicznie

Vim pozwala na zwielokrotnianie wykonywanych akcji poprzez poprzedzenie ich liczbą całkowitą

- **20j** - przejdź 20 linii w dół
- **5w** - przejdź o 5 słów do przodu
- **5b** - przejdź o 5 słów do tyłu
- **10.** - powtórz poprzednią akcję 10 razy
- **10@q** - wykonaj makro 10 razy

2.6 Poruszanie się po tekście

Poruszanie się po liniach

- **“liczba” + gg** - wybrana linia w pliku
- **ctrl + o** - powrót
- **0 i \$** - początek i koniec linii
- **^** - pierwszy niepusty znak w linii
- **%** - następny nawias zamykający

Poruszanie się po słowach - motions

- **w, W** - przejście do początku następnego “słowa/Słowa”
- **e, E** - przejście do końca bieżącego słowa
- **b, B** - przejście do początku bieżącego słowa
- **f, F** - używane w połączeniu z jakimś innym znakiem. Małe “f” przechodzi do następnego wystąpienia wybranego znaku, a wielkie “F” do poprzedniego
- **t, T** - analogicznie do “f” tylko, że przenosi kursor przed szukany znak

2.7 Połączmy je

[Licznik] [operator] [obiekt tekstowy / motion]

- **gUiW** - Zamiana całego "Słowa" na pisane wielkimi literami
- **dap** - Usuń cały paragraf wraz z pustą linią nad nim
- **4dw** - Usuń cztery następne słowa.
- **ci{** - usuń wszystko w najbliższych nawiasach klamrowych - np. ciało funkcji i przejdź do trybu pisania
- **2dt+** - Usuń wszystko do drugiego znalezionej znaku "+" bez samego "+" (znak szukany jest od miejsca wywołania do końca linii)

2.8 Inne skróty

vi / vim graphical cheat sheet

Esc														normal mode													
~ toggle case	! external filter	@. play macro	# prev ident	\$ eol	% goto match	^ "soft" bol	& repeat :s	* next ident	(begin sentence) end sentence	"soft" bol down	+ next line															
. goto mark	1	2	3	4	5	6	7	8	9	0 "hard" bol	- prev line	= auto-format															
Q ex mode	W next WORD	E end WORD	R replace mode	T back 'till	Y yank line	U undo line	I insert at bol	O open above	P paste before	{ begin parag.	}	end parag.															
q record macro	w next word	e end word	r replace char	t 'till	y yank	u undo	i insert mode	o open below	p paste after	[misc]	misc															
A append at eol	S subst line	D delete to eol	F "back" find ch	G eof/ goto ln	H screen top	J join lines	K help	L screen bottom	.	ex cmd line	" reg. spec	bol/ goto col															
a append	s subst char	d delete	f find char	g extra cmds	h ←	j ↓	k ↑	l →	.	repeat t/T/f/F	' goto mk. bol	\ not used!															
Z quit	X back-space	C change to eol	V visual lines	B prev WORD	N prev (find)	M screen mid'l	< un-indent	> indent	?. find (rev.)																		
Z extra cmds	X delete char	c change	V visual mode	b prev word	n next (find)	m set mark	, reverse t/T/f/F	.	repeat cmd	/ find																	

motion	moves the cursor, or defines the range for an operator
command	direct action command, if red , it enters insert mode
operator	requires a motion afterwards, operates between cursor & destination
extra	special functions, requires extra input
q.	commands with a dot need a char argument afterwards

bol = beginning of line, eol = end of line, mk = mark, yank = copy

words: `quux(foo, bar, baz);`
 WORDs: `quux(foo, bar, baz);`

Main command line commands ('ex'):

:w (save), :q (quit), :q! (quit w/o saving)
 :e f (open file f),
 :%s/x/y/g (replace 'x' by 'y' filewide),
 :h (help in vim), :new (new file in vim),

Other important commands:

CTRL-R: redo (vim),
 CTRL-F/-B: page up/down,
 CTRL-E/-Y: scroll line up/down,
 CTRL-V: block-visual mode (vim only)

Visual mode:

Move around and type operator to act on selected region (vim only)

Notes:

- (1) use "x before a yank/paste/del command to use that register ('clipboard') (x=a..z,*) (e.g.: "ay\$ to copy rest of line to reg 'a')
- (2) type in a number before any action to repeat it that number of times (e.g.: 2p, d2w, 5j, d4j)
- (3) duplicate operator to act on current line (dd = delete line, >> = indent line)
- (4) ZZ to save & quit, ZQ to quit w/o saving
- (5) zt: scroll cursor to top, zb: bottom, zz: center
- (6) gg: top of file (vim only), gf: open file under cursor (vim only)

3. Bardziej zaawansowana edycja

Zaawansowana edycja projektów

- Wyszukiwanie
 - szukanie w bieżącym buforze
 - szukanie wybranego słowa
 - szukanie w całym projekcie
- Makra
- Tabby
- Okna

3.1 Wyszukiwanie

Szukanie w bieżącym buforze

Szukamy za pomocą `/` - następnie za pomocą `n` przechodzimy do następnego wystąpienia, a `N` do poprzedniego

Szukanie wybranego słowa

- za pomocą `"*"` - przechodzi do następnego wystąpienia
- za pomocą `"#"` - przechodzi do poprzedniego wystąpienia

Szukanie po całym projekcie

- Można szukać po otwartych buforach
- Można użyć komendy **vimgrep**
- Można użyć znanego narzędzia **Silver searcher** inaczej **ag**.
Najłatwiej jest to osiągnąć dodając plugin fzf.

3.2 Makra

Czym jest makro?

- Makro to zapisana/nagrana sekwencja wciśnięć klawiszy.
- Są one bardzo często wykorzystywane i potrafią znacznie przyspieszyć pracę. Piszemy program by napisać inny program
- Tworzenie ich jest ciekawym zajęciem - zaczynamy myśleć jak zrobić coś szybciej za pomocą makra

Jak nagrać makro?

- Wciskamy **q** + **rejestr** - Rejestrem może być dowolny znak.
Teraz wszystko co zrobimy zostanie nagrane
- Kończymy nagrywanie **q**
- Uruchamiamy makro - **@** + **rejestr**

Operator “.”

- “.” - operator kropki jest najprostrzym makrem - powtarza poprzednią operację np. **diw** czyli usunięcie słowa.

3.2 Makra

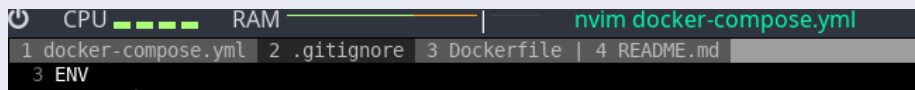
Przykład makra

Chcemy w każdym z 50 plików, mających po kilkadziesiąt funkcji, zmienić każdą nazwę funkcji z snake_case na camelCase i zapisać nazwę każdej zmienionej nazwy w Changelogu.

- Przyjmijmy, że język to Python. Szukamy więc “def” np. za pomocą **vimgrep** będąc w pliku zawierającym Changelog
- Włączamy nagrywanie i idziemy do pierwszego znalezionej rezultatu
- Kopiujemy całą linię (**Y**) i wracamy do miejsca skąd wykonaliśmy skok (**ctrl + o**)
- Wklejamy w Changelogu i wracamy (**ctrl + i**)
- Zamieniamy nazwę funkcji na camelCase - `:s#_\(\l\)\#\u\l#g`
- Kopiujemy nową nazwę, cofamy się do pliku z Changelogiem i wklejamy
- Kończymy nagrywanie makra
- Uruchamiamy makro dowolną liczbę razy

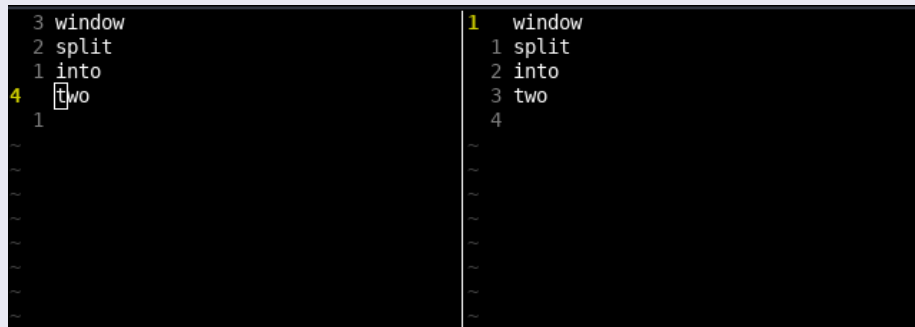
3.3 Taby i Okna

Taby



A terminal window with a dark background. At the top, there's a status bar showing 'CPU' with four green bars, 'RAM' with a green-to-red gradient bar, and 'nvim docker-compose.yml' in green text. Below this, a tab bar shows four tabs: '1 docker-compose.yml', '2 .gitignore', '3 Dockerfile', and '4 README.md'. The active tab is '1 docker-compose.yml', which shows line 3 with the text 'ENV'.

Okna



A terminal window with a dark background, split into two panes. The left pane shows a sequence of commands: '3 window', '2 split', '1 into', and '4 |t|wo' (where the cursor is on the 't' of 'two'). The right pane shows the result of these commands: '1 window', '1 split', '2 into', '3 two', and '4'. Both panes have line numbers on the left margin.

4. Konfiguracja

VIM jako IDE

- Vimrc
- Pluginy
 - Plugin managers
 - NerdTree
 - Powerline
 - Fugitive
 - fzf
- LSP

4.1 Vimrc

- Vimrc jest to plik konfiguracyjny Vima.
- Pozwala na kontrolę wersji naszego IDE - config w gicie
- Umożliwia zmienianie wbudowanych ustawień za pomocą komendy **set**
Np. **set number** - Włączenie numerowania linii
- Umożliwia definiowanie własnych skrótów klawiszowych i makr Np.
dodanie przenoszenia linii w dół - `nnoremap <M-j> :m .+1<CR>==`
- Pozwala na zmianę wyglądu edytora - kolory, colorschemes
- Pozwala na dodawanie pluginów

4.2 Pluginy

Menadżery pluginów

- vim-plug
- Vundle
- Pathogen

NerdTree

```
▼ lectures/
  ► BCYB/
  ► BEST/
  ► GIS/
  ► PORR/
  ► SD1/
  ► TASS/
  README.md
  ► publication-graph-analysis/
  ▼ seminarium/
    nerdtree.png
    presentantion1.md
  ▲ 33 # 3.3 Taby i Okna
    32
    31 ## Taby
    30
    29 ![tabs](tabs.png)
    28
    27 ## Okna
    26
    25 ![windows](window_split.png)
    24
  ▲ 23 # 4. Konfiguracja
    22
```

4.2 Pluginy

Powerline/Lightline

```
NORMAL master | AUTHORS.md utf-8 | markdown 20% 1:1
```

Fugitive - integracja z gitem

fzf - Fuzzy Finder

```
19
18
17
16
15
14
13
12
11
10 > presentation1.md
9  fzf.png
8  nerdtree.png
7  tabs.png
6  vim-sheet.png
5  line.png
4  window_split.png
3  presentation1.pdf
2  presentation1.pdf
1  9/9
0 ~/D/s/seminarium/
> fzf
:call FzfGitFilesIfPossible()
```

```
4 author:
5 - Piotr Grabarski
6 theme:
7 - Madrid
8 date:
9 - Warszawa 2020-04-13
10 ---
11
12 # Plan prezentacji
13
14 1. Wstęp
15
16 2. Podstawy
17
18 3. Zaawansowana edycja
19
20 4. Konfiguracja
21
22 5. Język VIM'a poza edytorem
```

4.3 LSP

Language Server Protocol

LSP to protokół oparty na JSON'ie opracowany przez firmę Microsoft. Początkowo był tworzony tylko dla Visual Studio Code, ale obecnie jest otwartym standardem. Można go używać w Vimie do auto uzupełniania, podpowiedzi, znajdowania definicji funkcji, znajdowania i zamieniania tekstu itp.

Neovim i CoC

- Neovim nie różni się mocno od zwykłego Vim'a, ale jest na pewno lepszy.
Wszystkie komendy zawarte w tej prezentacji działają w obu edytorach.
- Conqueror of Completion - Plugin do NeoVima pozwalający na łatwą integrację z LSP

5. Język VIM'a poza edytorem

Skróty vimowe są obecne w wielu programistycznych narzędziach, których używamy na co dzień, takich jak **less**, **tig**, czy **sed** (składnia wyszukiwania taka jak w vimie).

Inne narzędzia mające domyślne skróty vimowe:

- **Ranger i cfiles** - Terminalowe menadżery plików
- **Zathura** - PDF viewer
- **Tig** - Narzędzie ułatwiające pracę z gitem
- **SC-IM** - Terminalowy excel
- **Termite** - Emulator terminala
- **Inne** - <https://vim.reversed.top/>

5. Język VIM'a poza edytorem

Jest również wiele programów, które domyślnie nie mają włączonych skrótów vimowych ale pozwalają na ich łatwą zmianę.

Przeglądarka

Przeglądarki oferują wiele rozszerzeń vimowych np.: Vimium

Inne IDE

Np.: Emacs - Evil Mode oraz pochodne emacsa - spacemacs, doom-emacs

Shell

Domyślnie shell (np.: bash, zsh) jest zwykle w trybie emacs, ale można przełączyć go w tryb VIM

Suckless - <https://suckless.org/>

Oprogramowanie minimalistyczne, które za cel stawia sobie minimalizację zużycia zasobów i prostotę wykonania oraz działania

Bibliografia

- Dokumentacja VIM'a - <http://vimdoc.sourceforge.net/html/doc/>
- VIM - Porady i tipy - https://vim.fandom.com/wiki/Vim_Tips_Wiki

Dziękuję za uwagę

- Moja konfiguracja - <https://github.com/Gr4b4rz/vim-config>
- Polecam na start - **vimtutor** - terminalowy samouczek
- Do pracy z Vimem polecam **tmux** lub tiling window manager p.: **i3**, **xmonad**, **dwm**