

# TeXnical Vim

---

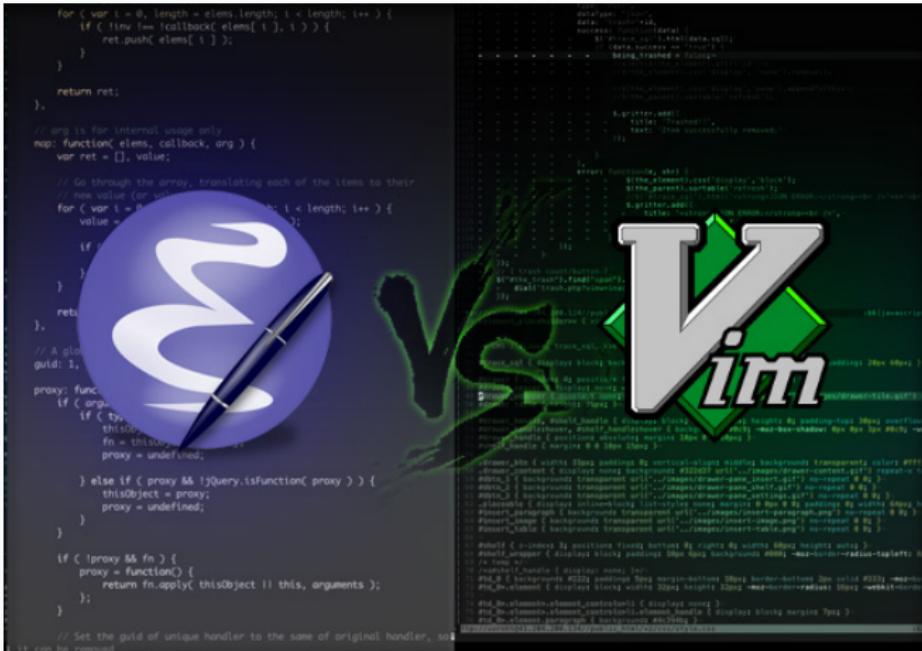
이재호

2020년 2월 15일

서울대학교 전기·정보공학부 / KTUG



# Editor War (1985~현재)



그렇다면 어떤 에디터를 쓰는게 제일 좋을까요?

그렇다면 어떤 에디터를 쓰는게 제일 좋을까요?

여러분의 취향에 맞는 에디터를 쓰세요!

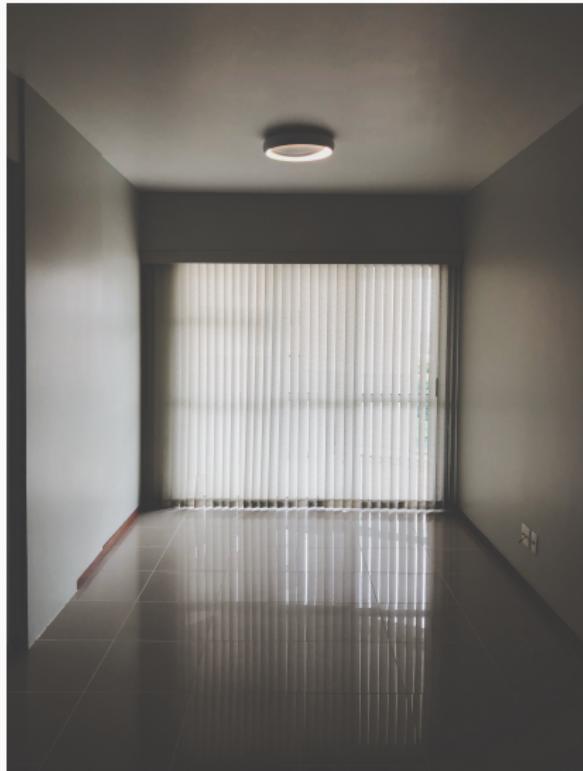
그렇다면 어떤 에디터를 쓰는게 제일 좋을까요?

여러분의 취향에 맞는 에디터를 쓰세요!

한 가지 취향과 가능성을 소개해드리려고 합니다.

Vim의 가장 큰 무기는 원하는대로 만들 수 있다는 것입니다.

Vim의 가장 큰 무기는 원하는대로 만들 수 있다는 것입니다.



Vim의 가장 큰 무기는 원하는대로 만들 수 있다는 것입니다.

그러나, 이는 동시에  
진입장벽이기도 합니다.



Vim의 가장 큰 무기는 원하는대로 만들 수 있다는 것입니다.



그러나, 이는 동시에  
진입장벽이기도 합니다.

# 가능성

Technical-Vim

I II III IV

texnical-vim.tex

```
1: texnical-vim.tex >
  h Toggle help text
  t Toggle sorted TODOs
  +/- Decrease/increase ToC depth
  f/F Apply/clear filter
  s Hide numbering

Layers: L label
        I include
        T todo
        C content

Preamble
Frame
Frame
Frame
Frame

3 Why Vim?
Frame: 첫 번째 관문
Frame: 본 발표는...
Frame: 그러나 본 발표는...
Frame: Vim이 뭘까요?
Frame: 제 X 대신 Vim인가요?
Frame: (Neo)Vim의 뜻 모습
Frame: \TeXnical Vim 설정
Frame

Table of contents (vintex) -> texnical-vim.tex          tex > (c) 428W < 51% ≡ 119/230 ≈ : 36
1 texnical-vim.tex[173 warning] Overfull \vbox (377.6639pt too high) detected at line 173$
```

Quickfix Vintex errors (LaTeX logfile)

```
:resize +5
[tex] @:nvim* 1:nvim-
```

buffers

imux

66% 21:48

texnical-vim.pdf (page 2 of 16)

Previous Next Page Back/Forward Zoom Tool Mode

TExnical Vim

이제호  
2020년 2월 15일  
서울대학교 전기·정보공학부/KTUG

MyPhygmyed  
MyPhygmyed

manus

TeXnical Vim

Atom

B

Emacs

Sublime Text

Notepad++

VS Code

gEdit

TeXShop

TeX

Vim

VS Code

Page 2 of 16

# 본 발표는...

## T<sub>E</sub>X 사용자 중

1. Vim에 대해서 들어봤지만 시도하기가 막막하셨던 분들,

# 본 발표는...

## T<sub>E</sub>X 사용자 중

1. Vim에 대해서 들어봤지만 시도하기가 막막하셨던 분들,
2. Vim을 쓸 줄 알지만 자료가 별로 없어서 T<sub>E</sub>X을 Vim에서 쓰지 않았던 분들,

# 본 발표는...

## T<sub>E</sub>X 사용자 중

1. Vim에 대해서 들어봤지만 시도하기가 막막하셨던 분들,
2. Vim을 쓸 줄 알지만 자료가 별로 없어서 T<sub>E</sub>X을 Vim에서 쓰지 않았던 분들,
3. Vim(혹은 Emacs+AUCT<sub>E</sub>X)으로 T<sub>E</sub>X을 쓰지만 다른 사람들의 작업 환경이 궁금하신 분들

# 본 발표는...

## T<sub>E</sub>X 사용자 중

1. Vim에 대해서 들어봤지만 시도하기가 막막하셨던 분들,
2. Vim을 쓸 줄 알지만 자료가 별로 없어서 T<sub>E</sub>X을 Vim에서 쓰지 않았던 분들,
3. Vim(혹은 Emacs+AUCT<sub>E</sub>X)으로 T<sub>E</sub>X을 쓰지만 다른 사람들의 작업 환경이 궁금하신 분들

Vim으로 T<sub>E</sub>X 문서를 작성하고 편집하는 효율적인 세팅!



vim tex



elle.korea.ac.kr > ~ywkim > mawiki > wiki > TeXH... ▾ Translate this page

### [attachment:KCmenu.jpg \[\[TableOfContents\]\] CategoryTeX ...](#)

[[http://math.korea.ac.kr/~ywkim/kkk/tex/tex\\_ywk3.pdf](http://math.korea.ac.kr/~ywkim/kkk/tex/tex_ywk3.pdf) TeX에서 우리 말을 쓸 때 주의할 점] 2003년 7  
... {{#if:vim tex |makeatletter \def\markboth{\#1\#2}% \begingroup ...  
You visited this page on 2/11/20.

teeve.sim24.club > sty-latex ▾ Translate this page

### [Sty Latex 설치 | teeve.sim24.club](#)

[Vim-LaTeX] 원도우즈에서 Vim으로 LaTeX 사용하기. MiKTeX Vim으로 논문을 쓰면 뭔가 기분이 좋다. 근  
데 원도우즈에서 Vim과 LaTeX를 연결하는 게 만만치가 ...  
You visited this page on 2/11/20.

sodocumentation.net > latex ▾ Translate this page

### [latex - 라텍스 시작하기 | latex Tutorial - SO Documentation](#)

latex documentation: 라텍스 시작하기, ... Emacs 나 Vim의 속련 된 사용자가 에디터 (다른 곳에서는 사용  
할 수없는 기능을 제공하는 플리그인)를 고수하고 싶어 할 ...  
You visited this page on 2/11/20.

woonizzooni.tistory.com > entry > MSYS2-설치-및... ▾ Translate this page

### [MSYS2 설치 및 패키지 설치 - woonizzooni - 티스토리](#)

Jul 10, 2019 - pacman -S --needed base-devel gcc cmake git vim :: 56 개의 구성원이 ... swig 51)  
texinfo 52) texinfo-tex 53) tycrc 54) unrar 55) wget 56) xmito ...  
You visited this page on 2/11/20.

ememoho.com > bbs > tb.php > freeboard ▾ Translate this page

### [gvim - ememoho.com](#)

1988년 Stevie라는 vi clone 에디터를 Bram (VIM 주 개발자)이 Amiga에 VIM ... 가령 :set  
makeprg=latex 하고, ex.tex 파일을 작성후 :make 하면 'latex ex.tex'가 수행 ...  
You visited this page on 2/11/20.



1 2 Next

Why Vim?

Vim Setup

Hello, Vim!

TEXnical Vim

$+\alpha$

## Why Vim?

---

# 첫 번째 관문

T<sub>E</sub>X을 처음 깔았을 때 누구나 마주하는 문제:

# 첫 번째 관문

T<sub>E</sub>X을 처음 깔았을 때 누구나 마주하는 문제:

T<sub>E</sub>X 문서를 어떤 도구로 만들까?

# 첫 번째 관문

T<sub>E</sub>X을 처음 깔았을 때 누구나 마주하는 문제:

T<sub>E</sub>X 문서를 어떤 도구로 만들까?

가장 기본적인 선택지:

- T<sub>E</sub>Xworks (Windows)
- T<sub>E</sub>XShop (macOS)

Why Vim?  
○○●○○○

Vim Setup  
○○○

Hello, Vim!  
○○○○○○○○○○○○

T<sub>E</sub>Xnical Vim  
○○○○○○○○○○○○○○

+α  
○○○

# Vim이 뭔가요?

# Vim이 뭔가요?

Vim은 modal(모달) 에디터입니다.

# Vim이 뭔가요?

Vim은 modal(모달) 에디터입니다.

- Normal, Insert, Visual, Command-line 모드
  - ▶ 와 Ex, Select 모드

## Vim이 뭔가요?

Vim은 modal(모달) 에디터입니다.

- Normal, Insert, Visual, Command-line 모드
    - ▶ 와 Ex, Select 모드
  - Editor라는 이름에 걸맞게, 편집하는 일에 특화
    - ▶ 반복 작업 및 영역 편집

# Vim이 뭔가요?

Vim은 modal(모달) 에디터입니다.

- Normal, Insert, Visual, Command-line 모드
  - ▶ 와 Ex, Select 모드
- Editor라는 이름에 걸맞게, 편집하는 일에 특화
  - ▶ 반복 작업 및 영역 편집
- 기본적으로는 터미널에서 사용
  - ▶ 손은 키보드 위에서만

## Vim이 뭔가요?

Vim은 modal(모달) 에디터입니다.

- Normal, Insert, Visual, Command-line 모드
    - ▶ 와 Ex, Select 모드
  - Editor라는 이름에 걸맞게, 편집하는 일에 특화
    - ▶ 반복 작업 및 영역 편집
  - 기본적으로는 터미널에서 사용
    - ▶ 손은 키보드 위에서만
  - 매우 유연한 설정
    - ▶ 그만큼 초기 설정은 매우 적음

# 왜 X 대신 Vim인가요?

- X = Electron 계열의 에디터 (Atom, Visual Studio Code)

# 왜 X 대신 Vim인가요?

- X = Electron 계열의 에디터 (Atom, Visual Studio Code)
  - ▶ 장점
    - GUI로 동작하므로 진입장벽이 낮음

# 왜 X 대신 Vim인가요?

■ X = Electron 계열의 에디터 (Atom, Visual Studio Code)

▶ 장점

- GUI로 동작하므로 진입장벽이 낮음
- Language server를 통한 강력한 문법 강조 및 자동 완성

# 왜 X 대신 Vim인가요?

## ■ X = Electron 계열의 에디터 (Atom, Visual Studio Code)

### ▶ 장점

- GUI로 동작하므로 진입장벽이 낮음
- Language server를 통한 강력한 문법 강조 및 자동 완성

### ▶ 단점

- (텍스트 편집을 위해서!) 웹 브라우저(Chromium)의 부하

# 왜 X 대신 Vim인가요?

## ■ X = Electron 계열의 에디터 (Atom, Visual Studio Code)

### ▶ 장점

- GUI로 동작하므로 진입장벽이 낮음
- Language server를 통한 강력한 문법 강조 및 자동 완성

### ▶ 단점

- (텍스트 편집을 위해서!) 웹 브라우저(Chromium)의 부하
- Language Server Protocol (LSP)의 등장

# 왜 X 대신 Vim인가요?

## ■ X = Electron 계열의 에디터 (Atom, Visual Studio Code)

### ▶ 장점

- GUI로 동작하므로 진입장벽이 낮음
- Language server를 통한 강력한 문법 강조 및 자동 완성

### ▶ 단점

- (텍스트 편집을 위해서!) 웹 브라우저(Chromium)의 부하
- Language Server Protocol (LSP)의 등장

## ■ X = Emacs

# 왜 X 대신 Vim인가요?

## ■ X = Electron 계열의 에디터 (Atom, Visual Studio Code)

### ▶ 장점

- GUI로 동작하므로 진입장벽이 낮음
- Language server를 통한 강력한 문법 강조 및 자동 완성

### ▶ 단점

- (텍스트 편집을 위해서!) 웹 브라우저(Chromium)의 부하
- Language Server Protocol (LSP)의 등장

## ■ X = Emacs

### ▶ 장점

- TUI로 동작함

# 왜 X 대신 Vim인가요?

## ■ X = Electron 계열의 에디터 (Atom, Visual Studio Code)

### ▶ 장점

- GUI로 동작하므로 진입장벽이 낮음
- Language server를 통한 강력한 문법 강조 및 자동 완성

### ▶ 단점

- (텍스트 편집을 위해서!) 웹 브라우저(Chromium)의 부하
- Language Server Protocol (LSP)의 등장

## ■ X = Emacs

### ▶ 장점

- TUI로 동작함
- More than an editor

# 왜 X 대신 Vim인가요?

## ■ X = Electron 계열의 에디터 (Atom, Visual Studio Code)

### ▶ 장점

- GUI로 동작하므로 진입장벽이 낮음
- Language server를 통한 강력한 문법 강조 및 자동 완성

### ▶ 단점

- (텍스트 편집을 위해서!) 웹 브라우저(Chromium)의 부하
- Language Server Protocol (LSP)의 등장

## ■ X = Emacs

### ▶ 장점

- TUI로 동작함
- More than an editor

### ▶ 단점

# 왜 X 대신 Vim인가요?

## ■ X = Electron 계열의 에디터 (Atom, Visual Studio Code)

### ▶ 장점

- GUI로 동작하므로 진입장벽이 낮음
- Language server를 통한 강력한 문법 강조 및 자동 완성

### ▶ 단점

- (텍스트 편집을 위해서!) 웹 브라우저(Chromium)의 부하
- Language Server Protocol (LSP)의 등장

## ■ X = Emacs

### ▶ 장점

- TUI로 동작함
- More than an editor

### ▶ 단점

- TUI로 동작함
- More than an editor

# 2020년의 Vim

■ User Friendly!

# 2020년의 Vim

## ■ User Friendly!

- ▶ LSP의 혜택으로 강력한 문법 강조, 자동 완성, 문법 검증

# 2020년의 Vim

## ■ User Friendly!

- ▶ LSP의 혜택으로 강력한 문법 강조, 자동 완성, 문법 검증
  - LSP가 너무 무겁다면, 기존 방식의 문법팩 사용 가능
- ▶ 모든 것이 사용자화 가능

# 2020년의 Vim

## ■ User Friendly!

- ▶ LSP의 혜택으로 강력한 문법 강조, 자동 완성, 문법 검증
  - LSP가 너무 무겁다면, 기존 방식의 문법팩 사용 가능
- ▶ 모든 것이 사용자화 가능
- ▶ 직관적이고 편한 명령어 (동사 + 명사)

# 2020년의 Vim

## ■ User Friendly!

- ▶ LSP의 혜택으로 강력한 문법 강조, 자동 완성, 문법 검증
  - LSP가 너무 무겁다면, 기존 방식의 문법팩 사용 가능
- ▶ 모든 것이 사용자화 가능
- ▶ 직관적이고 편한 명령어 (동사 + 명사)
- ▶ 40년이 넘는 자료와 사용자 그룹

# 2020년의 Vim

## ■ User Friendly!

- ▶ LSP의 혜택으로 강력한 문법 강조, 자동 완성, 문법 검증
  - LSP가 너무 무겁다면, 기존 방식의 문법팩 사용 가능
- ▶ 모든 것이 사용자화 가능
- ▶ 직관적이고 편한 명령어 (동사 + 명사)
- ▶ 40년이 넘는 자료와 사용자 그룹
- ▶ 한 가지 일을 하고, 그것을 잘함 (UNIX Philosophy: Do One Thing and Do It Well)

# 2020년의 Vim

## ■ User Friendly!

- ▶ LSP의 혜택으로 강력한 문법 강조, 자동 완성, 문법 검증
  - LSP가 너무 무겁다면, 기존 방식의 문법팩 사용 가능
- ▶ 모든 것이 사용자화 가능
- ▶ 직관적이고 편한 명령어 (동사 + 명사)
- ▶ 40년이 넘는 자료와 사용자 그룹
- ▶ 한 가지 일을 하고, 그것을 잘함 (UNIX Philosophy: Do One Thing and Do It Well)

## ■ Developer Friendly!

# 2020년의 Vim

## ■ User Friendly!

- ▶ LSP의 혜택으로 강력한 문법 강조, 자동 완성, 문법 검증
  - LSP가 너무 무겁다면, 기존 방식의 문법팩 사용 가능
- ▶ 모든 것이 사용자화 가능
- ▶ 직관적이고 편한 명령어 (동사 + 명사)
- ▶ 40년이 넘는 자료와 사용자 그룹
- ▶ 한 가지 일을 하고, 그것을 잘함 (UNIX Philosophy: Do One Thing and Do It Well)

## ■ Developer Friendly!

- ▶ Lua를 내장한 Neovim의 등장으로 Lua로 스크립팅이 가능해짐 (LuaTeX?)

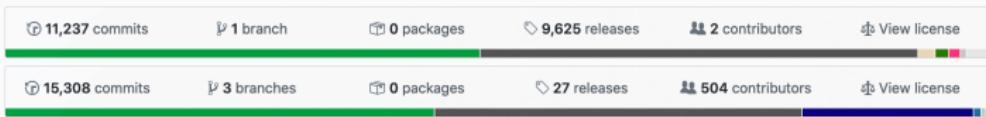
# 2020년의 Vim

## ■ User Friendly!

- ▶ LSP의 혜택으로 강력한 문법 강조, 자동 완성, 문법 검증
  - LSP가 너무 무겁다면, 기존 방식의 문법팩 사용 가능
- ▶ 모든 것이 사용자화 가능
- ▶ 직관적이고 편한 명령어 (동사 + 명사)
- ▶ 40년이 넘는 자료와 사용자 그룹
- ▶ 한 가지 일을 하고, 그것을 잘함 (UNIX Philosophy: Do One Thing and Do It Well)

## ■ Developer Friendly!

- ▶ Lua를 내장한 Neovim의 등장으로 Lua로 스크립팅이 가능해짐 (LuaTeX?)
- ▶ 커뮤니티 기반 (무엇이 Vim이고 Neovim일까요?)



Why Vim?  
oooo●

Vim Setup  
○○○

Hello, Vim!  
○○○○○○○○○○

TExnical Vim  
○○○○○○○○○○○○

+α  
○○○



# neovim

# Vim Setup

---

# TeXnical Vim 설정 (for macOS) i

macOS용 package manager인 Homebrew를 사용합니다.

1. <https://brew.sh/>를 참고하여 Homebrew를 설치합니다.
2. iTerm2, MacTeX, Skim, 그리고 Neovim을 설치합니다.

```
brew tap homebrew/cask
brew cask install item2 mactex skim
brew install neovim
```

**iTerm2** Color scheme, option key mapping,  
powerline glyphs, ...

**MacTeX** TeXLive의 macOS용 재배포

**Skim** SyncTeX support, automatic refresh, ...

# TeXnical Vim 설정 (for macOS) ii

### 3. Python을 설치합니다.

- ▶ 평소에 Python으로 프로젝트를 하시는 분들은 pyenv과 virtualenv 등으로 관리하시는 것을 권장드립니다.

```
brew install pyenv pyenv-virtualenv # and setup
→ Python, or simply
brew install python
```

### 4. neovim-remote(nvr)을 설치합니다.

```
pip3 install neovim-remote
```

- ▶ 이는 Skim과 같은 외부 프로그램이 Neovim을 원격으로 조작하여 reverse search를 가능하게 합니다.

## TeXnical Vim 설정 (for macOS) iii

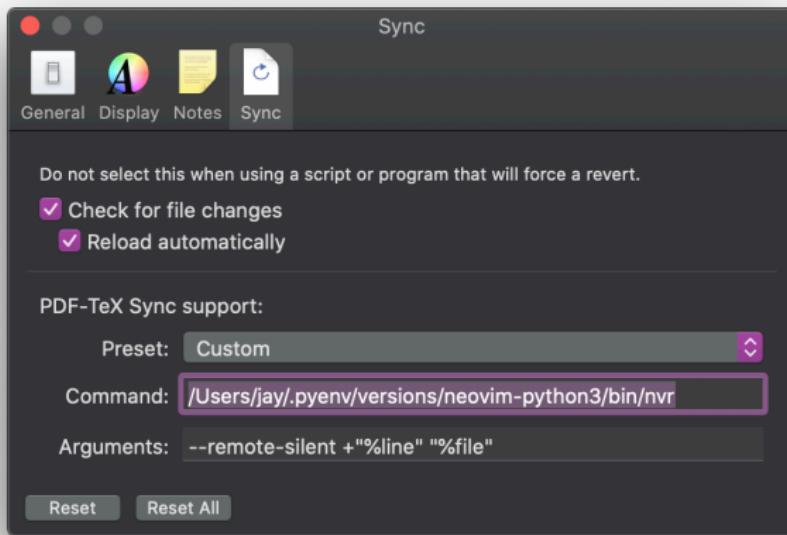
5. <https://github.com/junegunn/vim-plug#unix-1>을 참고하여 Vim plugin manager vim-plug를 설치합니다.
6. (Vim을 아신다면) ~/.config/nvim/init.vim에 다음과 같이 작성합니다:

```
1 call plug#begin('~/local/share/nvim/plugged')
2 Plug 'lervag/vimtex'
3 call plug#end()
```

그 후 (다시 source하거나) 파일을 다시 열어서 :PlugInstall합니다.

# TeXnical Vim 설정 (for macOS) iv

7. Skim의 Preferences를 열어 다음과 같이 설정합니다:



# TeXnical Vim 설정 (for Windows) i

Windows용 package manager인 Chocolatey 사용을 하며, 이하 명령어들은 PowerShell에서 실행합니다. (**Windows 설정은 아직 완벽하지 않습니다.**)

1. <https://chocolatey.org/install>을 참고하여 Chocolatey를 설치합니다. (명령어가 너무 길어 링크로 대체)
2. KTUG Wiki를 참고하여, install-tl.zip의 압축을 풀고 install-tl-windows.bat을 실행하여 설치합니다.
3. Terminal, SumatraPDF, 그리고 Neovim을 설치합니다.

```
choco install microsoft-windows-terminal sumatrapdf  
→ neovim
```

# T<sub>E</sub>Xnical Vim 설정 (for Windows) ii

**Windows Terminal** Color scheme (that supports iTerm themes!), **option key mapping**, powerline glyphs, ...

**SumatraPDF** SyncT<sub>E</sub>X support, automatic refresh, ...

# TeXnical Vim 설정 (for Windows) iii

## 4. Python을 설치합니다.

- ▶ Python 가상 환경은 Windows용으로 pyenv-win이라는 것 있다고 합니다.

```
choco install python3
```

## 5. neovim-remote(nvr)을 설치합니다.

```
pip3 install neovim-remote
```

- ▶ 이는 Skim과 같은 외부 프로그램이 Neovim을 원격으로 조작하여 reverse search를 가능하게 합니다.

# TeXnical Vim 설정 (for Windows) iv

6. [https://github.com/junegunn/vim-plug#  
windows-powershell-1](https://github.com/junegunn/vim-plug#windows-powershell-1)을 참고하여 Vim plugin manager  
vim-plug를 설치합니다.
7. (Vim을 아신다면) ~\AppData\Local\nvim\init.vim에 다음과  
같이 작성합니다:

```
1 call plug#begin('~/AppData/Local/nvim/plugged')
2 Plug 'lervag/vimtex'
3 call plug#end()
```

그 후 (다시 source하거나) 파일을 다시 열어서  
:PlugInstall합니다.

# T<sub>E</sub>Xnical Vim 설정 (for Windows) v

8. <https://github.com/lervag/vimtex/issues/1585> 참고

# Hello, Vim!

---

## How do I exit the Vim editor?

Asked 7 years, 6 months ago Active 1 month ago Viewed 2.0m times

I'm stuck and cannot escape. It says:

3624 "type :quit<Enter> to quit VIM"

But when I type that it simply appears in the object body.

887 vim vi

share edit flag

edited May 14 '19 at 22:49



Peter Mortensen

24.9k ▾ 20 ▾ 90 ▾ 118

asked Aug 6 '12 at 12:25



jdancy

34.1k ▾ 5 ▾ 21 ▾ 29

124 Are you just trying to quit VIM ? If this is the case, press "escape" and then type ':q' – [Pop](#) Aug 6 '12 at 12:28 ✓

46 Don't forget the colon! You should type :quit and then hit the [ENTER] key. – [Farahmand](#) Mar 4 '14 at 18:33 ✓

81 It's really easy to learn the basics of vim, and it's built right into your system. In terminal type "vimtutor". 25 minutes later you will be going faster than your usual text editor! – [Mark Robson](#) Jan 26 '15 at 12:11

4 Check [here](#) more commands. – [Tori](#) Aug 7 '15 at 15:36

74 To prevent git commit sending you to vim in the future: git config --global core.editor="nano". – [Tom Kelly](#) May 24 '17 at 3:19 ✓

[add a comment](#) | [show 4 more comments](#)

[start a bounty](#)



I Am Developer

@iamdeveloper

Following

I've been using Vim for about 2 years now, mostly because I can't figure out how to exit it.

RETWEETS 14,083 LIKES 8,154



3:26 PM - 17 Feb 2014

314 14K 8.2K

# Exit Vim

<Esc>를 두어번 누른 후 (normal 모드로 확실하게 가기),

## Exit Vim

<Esc>를 두어번 누른 후 (normal 모드로 확실하게 가기),

- :q를 친다. (저장 안 된 내용 있을시 오류)

# Exit Vim

<Esc>를 두어번 누른 후 (normal 모드로 확실하게 가기),

- :q를 친다. (저장 안 된 내용 있을시 오류)
- :q!를 친다. (저장 안 된 내용 있을시 버리기)

# Exit Vim

<Esc>를 두어번 누른 후 (normal 모드로 확실하게 가기),

- :q를 친다. (저장 안 된 내용 있을시 오류)
- :q!를 친다. (저장 안 된 내용 있을시 버리기)
- :wq를 친다. (저장하고 종료)

# Exit Vim

<Esc>를 두어번 누른 후 (normal 모드로 확실하게 가기),

- :q를 친다. (저장 안 된 내용 있을시 오류)
- :q!를 친다. (저장 안 된 내용 있을시 버리기)
- :wq를 친다. (저장하고 종료)
- :wq!를 친다. (읽기 전용이더라도 저장하고 종료)

# Exit Vim

<Esc>를 두어번 누른 후 (normal 모드로 확실하게 가기),

- :q를 친다. (저장 안 된 내용 있을시 오류)
- :q!를 친다. (저장 안 된 내용 있을시 버리기)
- :wq를 친다. (저장하고 종료)
- :wq!를 친다. (읽기 전용이더라도 저장하고 종료)
- :x를 친다. (저장 안 된 내용 있을시 저장하고 종료)

# Exit Vim

<Esc>를 두어번 누른 후 (normal 모드로 확실하게 가기),

- :q를 친다. (저장 안 된 내용 있을시 오류)
- :q!를 친다. (저장 안 된 내용 있을시 버리기)
- :wq를 친다. (저장하고 종료)
- :wq!를 친다. (읽기 전용이더라도 저장하고 종료)
- :x를 친다. (저장 안 된 내용 있을시 저장하고 종료)

이 때 q는 **quit**, w는 **write**, x는 **exit**의 약자입니다.

# Exit Vim

<Esc>를 두어번 누른 후 (normal 모드로 확실하게 가기),

- :q를 친다. (저장 안 된 내용 있을시 오류)
- :q!를 친다. (저장 안 된 내용 있을시 버리기)
- :wq를 친다. (저장하고 종료)
- :wq!를 친다. (읽기 전용이더라도 저장하고 종료)
- :x를 친다. (저장 안 된 내용 있을시 저장하고 종료)

이 때 q는 **quit**, w는 **write**, x는 **exit**의 약자입니다. 혹은 normal 모드에서 ZZ를 눌러 저장하고 종료하거나 ZQ를 눌러 변경 사항을 버리고 종료할 수 있습니다.

# The Normal, the Insert, and the Command-line

Vim에는 크게 normal, insert, 그리고 command-line 모드가 있습니다. (Normal 모드를 command 모드, command-line 모드를 ex 모드라고도 부릅니다.)

# The Normal, the Insert, and the Command-line

Vim에는 크게 normal, insert, 그리고 command-line 모드가 있습니다. (Normal 모드를 command 모드, command-line 모드를 ex 모드라고도 부릅니다.)

모드가 분리되어 있다는 것의 의미:

- GUI 텍스트 에디터에서는 글을 쓰다가, 마우스로 드래그 하다가, 메뉴바에서 작업을 고르고 하는 일

# The Normal, the Insert, and the Command-line

Vim에는 크게 normal, insert, 그리고 command-line 모드가 있습니다. (Normal 모드를 command 모드, command-line 모드를 ex 모드라고도 부릅니다.)

모드가 분리되어 있다는 것의 의미:

- GUI 텍스트 에디터에서는 글을 쓰다가, 마우스로 드래그 하다가, 메뉴바에서 작업을 고르고 하는 일 → 키보드 위에서 모드를 바꿔가며 모든 작업을 하는 것
- Normal 모드에서는 글을 쓰려고 해도 안 써진다?

# The Normal, the Insert, and the Command-line

Vim에는 크게 normal, insert, 그리고 command-line 모드가 있습니다. (Normal 모드를 command 모드, command-line 모드를 ex 모드라고도 부릅니다.)

모드가 분리되어 있다는 것의 의미:

- GUI 텍스트 에디터에서는 글을 쓰다가, 마우스로 드래그 하다가, 메뉴바에서 작업을 고르고 하는 일 → 키보드 위에서 모드를 바꿔가며 모든 작업을 하는 것
- Normal 모드에서는 글을 쓰려고 해도 안 써진다? → Normal 모드에서는 cmd, ctrl, alt, fn 등의 불필요한 조합을 (거의) 하지 않고 명령어를 입력

# The Normal, the Insert, and the Command-line

Vim에는 크게 normal, insert, 그리고 command-line 모드가 있습니다. (Normal 모드를 command 모드, command-line 모드를 ex 모드라고도 부릅니다.)

모드가 분리되어 있다는 것의 의미:

- GUI 텍스트 에디터에서는 글을 쓰다가, 마우스로 드래그 하다가, 메뉴바에서 작업을 고르고 하는 일 → 키보드 위에서 모드를 바꿔가며 모든 작업을 하는 것
- Normal 모드에서는 글을 쓰려고 해도 안 써진다? → Normal 모드에서는 cmd, ctrl, alt, fn 등의 불필요한 조합을 (거의) 하지 않고 명령어를 입력
- 작성(insert), 수정(normal), 작업(command-line)을 분리

Why Vim?  
○○○○○

Vim Setup  
○○○

Hello, Vim!  
○○○●○○○○○

T<sub>E</sub>Xnical Vim  
○○○○○○○○○○○○

+α  
○○○

hjkl == ← ↓ ↑ →

Insert 모드에서는 방향키로 이동하지 않습니다.

hjkl == ← ↓ ↑ →

Insert 모드에서는 방향키로 이동하지 않습니다.

Normal 모드에서는 방향키로 이동하지 않습니다.

hjkl == ← ↓ ↑ →

Insert 모드에서는 방향키로 이동하지 않습니다.

Normal 모드에서는 방향키로 이동하지 않습니다.

Normal 모드에서는 hjkl로 좌하상우로 이동할 수 있습니다.

hjkl == ← ↓ ↑ →

Insert 모드에서는 방향키로 이동하지 않습니다.

Normal 모드에서는 방향키로 이동하지 않습니다.

Normal 모드에서는 hjkl로 좌하상우로 이동할 수 있습니다.

Insert 모드에서는 이동하지 않습니다.

hjkl == ← ↓ ↑ →

Insert 모드에서는 방향키로 이동하지 않습니다.

Normal 모드에서는 방향키로 이동하지 않습니다.

Normal 모드에서는 hjkl로 좌하상우로 이동할 수 있습니다.

Insert 모드에서는 이동하지 않습니다.

Vim에서 모든 일은 normal 모드에서 이뤄집니다.

hjkl == ← ↓ ↑ →

Insert 모드에서는 방향키로 이동하지 않습니다.

Normal 모드에서는 방향키로 이동하지 않습니다.

Normal 모드에서는 hjkl로 좌하상우로 이동할 수 있습니다.

Insert 모드에서는 이동하지 않습니다.

Vim에서 모든 일은 normal 모드에서 이뤄집니다.

문서에서 이동하거나 작업을 할 때에는 insert 모드에서 <Esc>를 눌러 나온 후, normal 모드 혹은 command-line 모드에서 작업해야 합니다.

# 글쓰기

Normal 모드에서 insert 모드로 들어가는 방법은 많이 있지만,  
다음이 가장 많이 쓰입니다:

- i - 현재 커서 위치에서 insert 모드로

# 글쓰기

Normal 모드에서 insert 모드로 들어가는 방법은 많이 있지만,  
다음이 가장 많이 쓰입니다:

- i - 현재 커서 위치에서 insert 모드로
- a - 다음 커서 위치에서 insert 모드로

# 글쓰기

Normal 모드에서 insert 모드로 들어가는 방법은 많이 있지만,  
다음이 가장 많이 쓰입니다:

- i - 현재 커서 위치에서 insert 모드로
- a - 다음 커서 위치에서 insert 모드로
- I - 공백이 아닌 현재 줄의 시작에서 insert 모드로

# 글쓰기

Normal 모드에서 insert 모드로 들어가는 방법은 많이 있지만,  
다음이 가장 많이 쓰입니다:

- i - 현재 커서 위치에서 insert 모드로
- a - 다음 커서 위치에서 insert 모드로
- I - 공백이 아닌 현재 줄의 시작에서 insert 모드로
- A - 현재 줄의 끝에서 insert 모드로

# 글쓰기

Normal 모드에서 insert 모드로 들어가는 방법은 많이 있지만,  
다음이 가장 많이 쓰입니다:

- i - 현재 커서 위치에서 insert 모드로
- a - 다음 커서 위치에서 insert 모드로
- I - 공백이 아닌 현재 줄의 시작에서 insert 모드로
- A - 현재 줄의 끝에서 insert 모드로
- o - 새로운 아랫줄에 insert 모드로
- O - 새로운 윗줄에 insert 모드로

# 글쓰기

Normal 모드에서 insert 모드로 들어가는 방법은 많이 있지만,  
다음이 가장 많이 쓰입니다:

- i - 현재 커서 위치에서 insert 모드로
- a - 다음 커서 위치에서 insert 모드로
- I - 공백이 아닌 현재 줄의 시작에서 insert 모드로
- A - 현재 줄의 끝에서 insert 모드로
- o - 새로운 아랫줄에 insert 모드로
- O - 새로운 윗줄에 insert 모드로

iI, aA, oO는 각각 **input**, **append**, **open**의 약자입니다.

## 글에서 이동하기

Normal 모드에서 hjkl만으로 이동하는 것은 마치 ‘1단’ 기어와 같아서, 이것만으로 글을 이동하려면 곤란합니다.

- w - 다음 단어 시작으로

## 글에서 이동하기

Normal 모드에서 hjkl만으로 이동하는 것은 마치 ‘1단’ 기어와 같아서, 이것만으로 글을 이동하려면 곤란합니다.

- w - 다음 단어 시작으로
- b - 이전 단어 시작으로

## 글에서 이동하기

Normal 모드에서 hjkl만으로 이동하는 것은 마치 ‘1단’ 기어와 같아서, 이것만으로 글을 이동하려면 곤란합니다.

- w - 다음 단어 시작으로
- b - 이전 단어 시작으로
- W - 공백 기준 다음 단어 시작으로
- B - 공백 기준 이전 단어 시작으로

# 글에서 이동하기

Normal 모드에서 hjkl만으로 이동하는 것은 마치 ‘1단’ 기어와 같아서, 이것만으로 글을 이동하려면 곤란합니다.

- w - 다음 단어 시작으로
- b - 이전 단어 시작으로
- W - 공백 기준 다음 단어 시작으로
- B - 공백 기준 이전 단어 시작으로
- e - 다음 단어 끝으로
- ge - 이전 단어 끝으로
- E - 공백 기준 다음 단어 끝으로
- gE - 공백 기준 이전 단어 끝으로

# 글에서 이동하기

Normal 모드에서 hjkl만으로 이동하는 것은 마치 ‘1단’ 기어와 같아서, 이것만으로 글을 이동하려면 곤란합니다.

- w - 다음 단어 시작으로
- b - 이전 단어 시작으로
- W - 공백 기준 다음 단어 시작으로
- B - 공백 기준 이전 단어 시작으로
- e - 다음 단어 끝으로
- ge - 이전 단어 끝으로

- E - 공백 기준 다음 단어 끝으로
- gE - 공백 기준 이전 단어 끝으로
- ^ - 공백이 아닌 현재 줄의 시작으로
- 0 - 현재 줄의 시작으로
- \$ - 현재 줄의 끝으로
- gg - 문서 시작으로

# 글에서 이동하기

Normal 모드에서 hjkl만으로 이동하는 것은 마치 ‘1단’ 기어와 같아서, 이것만으로 글을 이동하려면 곤란합니다.

- w - 다음 단어 시작으로
- b - 이전 단어 시작으로
- W - 공백 기준 다음 단어 시작으로
- B - 공백 기준 이전 단어 시작으로
- e - 다음 단어 끝으로
- ge - 이전 단어 끝으로

- E - 공백 기준 다음 단어 끝으로
- gE - 공백 기준 이전 단어 끝으로
- ^ - 공백이 아닌 현재 줄의 시작으로
- 0 - 현재 줄의 시작으로
- \$ - 현재 줄의 끝으로
- gg - 문서 시작으로
- G - 문서 끝으로

# 글에서 이동하기

Normal 모드에서 hjkl만으로 이동하는 것은 마치 ‘1단’ 기어와 같아서, 이것만으로 글을 이동하려면 곤란합니다.

- w - 다음 단어 시작으로
- b - 이전 단어 시작으로
- W - 공백 기준 다음 단어 시작으로
- B - 공백 기준 이전 단어 시작으로
- e - 다음 단어 끝으로
- ge - 이전 단어 끝으로

wW, bB, eE, gG는 각각 word, back, end, goto의 약자입니다. ^와 \$는 regex의 그것에서 따온 것 같습니다.

- E - 공백 기준 다음 단어 끝으로
- gE - 공백 기준 이전 단어 끝으로
- ^ - 공백이 아닌 현재 줄의 시작으로
- 0 - 현재 줄의 시작으로
- \$ - 현재 줄의 끝으로
- gg - 문서 시작으로
- G - 문서 끝으로

## count \* (operator + motion)

위에서 보았던 w, e 등은 motion이라고 부릅니다. 이는 operator와 결합되어 작용합니다:

- c - 수정 (change)

## count \* (operator + motion)

위에서 보았던 w, e 등은 motion이라고 부릅니다. 이는 operator와 결합되어 작용합니다:

- c - 수정 (change)
- d - 삭제 (delete)

## count \* (operator + motion)

위에서 보았던 w, e 등은 motion이라고 부릅니다. 이는 operator와 결합되어 작용합니다:

- c - 수정 (change)
- d - 삭제 (delete)
- y - 복사 (yank)

## count \* (operator + motion)

위에서 보았던 w, e 등은 motion이라고 부릅니다. 이는 operator와 결합되어 작용합니다:

- c - 수정 (change)
- d - 삭제 (delete)
- y - 복사 (yank)
- g~ - 대소문자 바꾸기

## count \* (operator + motion)

위에서 보았던 w, e 등은 motion이라고 부릅니다. 이는 operator와 결합되어 작용합니다:

- c - 수정 (change)
- d - 삭제 (delete)
- y - 복사 (yank)
- g~ - 대소문자 바꾸기
- gu - 소문자 바꾸기

## count \* (operator + motion)

위에서 보았던 w, e 등은 motion이라고 부릅니다. 이는 operator와 결합되어 작용합니다:

- c - 수정 (change)
- d - 삭제 (delete)
- y - 복사 (yank)
- g~ - 대소문자 바꾸기
- gu - 소문자 바꾸기
- gU - 대문자 바꾸기 (Uppercase)

## count \* (operator + motion)

위에서 보았던 w, e 등은 motion이라고 부릅니다. 이는 operator와 결합되어 작용합니다:

- c - 수정 (change)
- d - 삭제 (delete)
- y - 복사 (yank)
- g~ - 대소문자 바꾸기
- gu - 소문자 바꾸기
- gU - 대문자 바꾸기 (Uppercase)

예를 들어, d(삭제)와 \$(줄 끝으로)을 같이 써서 d\$를 쓰면 줄 끝까지 지우고, 여기에 5를 앞에 붙여서 5d\$를 쓰면 이를 5회 반복합니다.

# Text Object

마지막으로, **text object**에 대해서 살펴봅시다. `apple`이라는 단어에서 커서가 1에 있었고, 해당 단어를 지우려면 지금까지 살펴본 것으로는

# Text Object

마지막으로, **text object**에 대해서 살펴봅시다. `apple`이라는 단어에서 커서가 1에 있었고, 해당 단어를 지우려면 지금까지 살펴본 것으로는 `a`로 가기 위해 `b`, 그리고 단어를 지우기 위해 `dw`를 해야 했습니다.

# Text Object

마지막으로, **text object**에 대해서 살펴봅시다. `apple`이라는 단어에서 커서가 `l`에 있었고, 해당 단어를 지우려면 지금까지 살펴본 것으로는 `a`로 가기 위해 `b`, 그리고 단어를 지우기 위해 `dw`를 해야 했습니다.

그러나 단어 ‘안’, 나아가 어떤 범위에서 편집을 쉽게 하기 위해서 Vim 은 **text object**를 제공합니다.

구조는 `a` 혹은 `i`와, `", ', (, )`, `w`, `W` 등을 결합한 것입니다.

# Text Object

마지막으로, **text object**에 대해서 살펴봅시다. `apple`이라는 단어에서 커서가 `l`에 있었고, 해당 단어를 지우려면 지금까지 살펴본 것으로는 `a`로 가기 위해 `b`, 그리고 단어를 지우기 위해 `dw`를 해야 했습니다.

그러나 단어 ‘안’, 나아가 어떤 범위에서 편집을 쉽게 하기 위해서 Vim은 **text object**를 제공합니다.

구조는 `a` 혹은 `i`와, `", ', (, )`, `w, W` 등을 결합한 것입니다.

예를 들어, `iw`은 ‘단어 안’, `aw`은 ‘(띄어쓰기를 포함한) 단어’입니다. 각각 “inner word”, “a word”라고 읽으시면 됩니다.

# Text Object

마지막으로, **text object**에 대해서 살펴봅시다. `apple`이라는 단어에서 커서가 `l`에 있었고, 해당 단어를 지우려면 지금까지 살펴본 것으로는 `a`로 가기 위해 `b`, 그리고 단어를 지우기 위해 `dw`를 해야 했습니다.

그러나 단어 ‘안’, 나아가 어떤 범위에서 편집을 쉽게 하기 위해서 Vim 은 **text object**를 제공합니다.

구조는 `a` 혹은 `i`와, `,`, `',`, `(`, `)`, `w`, `W` 등을 결합한 것입니다.

예를 들어, `iw`은 ‘단어 안’, `aw`은 ‘(띄어쓰기를 포함한) 단어’입니다.  
각각 “inner word”, “a word”라고 읽으시면 됩니다.

따라서, 이를 **operator**와 결합하여 `cis`와 같이 쓸 수 있습니다.

## 더 읽을거리

- neovim.io
- Vim Awesome
- Why, oh WHY, do those #?! nutheads use vi?
- Vimcasts.org

# TeXnical Vim

---

# Neovim + Vimtex + Coc.nvim + TexLab = ❤

**Neovim** 모던 Vim (The Engine)

**Vimtex**  $\text{\TeX}$  문서를 Vim에서 쉽게 쓸 수 있도록 motion, text objects, 기타 mapping을 제공하며, 문서 navigation, PDF 뷰어들과의 연동을 담당

**Coc.nvim** Vim 8/Neovim을 위한 IntelliSense engine입니다.  
LSP를 지원하며, 나아가 VS Code용 plugin을 Vim에서 쓸 수 있게 합니다.

**TexLab** Crossplatform  $\text{\LaTeX}$  / Bib $\text{\TeX}$  language server입니다.

# Language Server Protocol (LSP)란?



출처: <https://langserver.org/>

자동 완성, 문법 강조, 문서 등의 언어 분석 기능을 가진 language server와 IDE 및 에디터 (client) 간의 규약으로, Microsoft가 VS Code를 만들면서 공개하였습니다.

## Last Vim Setup

TexLab은 macOS의 경우 brew install texlab으로, Windows의 경우 <https://github.com/latex-lsp/texlab/releases>에서 설치할 수 있습니다.

Vimtex과 coc.nvim은 vim-plug로 설치하면 되는데, init.vim에 아래와 같이 추가하면 됩니다:

```
1 call plug#begin('~/local/share/nvim/plugged')
2 Plug 'lervag/vimtex'
3 Plug 'neoclide/coc.nvim', { 'branch': 'release' }
4 call plug#end()
5 let g:vimtex_view_method = 'skim'
6 let g:coc_global_extensions = ['coc-texlab']
```

Neovim을 껐다 켠 후 :PlugInstall 해줍니다. 이외 coc.nvim의 예시 설정: <https://github.com/neoclide/coc.nvim>

## init.vim / .vimrc

지금까지는 Vim을 조작하는 방법에 대해서 알아보았는데요, 지금은 간단히 Vim의 설정을 바꾸어 그냥 ‘검은 화면’을 바꾸어 봅시다.

## init.vim / .vimrc

지금까지는 Vim을 조작하는 방법에 대해서 알아보았는데요, 지금은 간단히 Vim의 설정을 바꾸어 그냥 ‘검은 화면’을 바꾸어 봅시다.

이런 설정은 Vim을 실행하여 command-line mode에서 :set nu 등과 같이 바꿀 수 있습니다. 그런데 이를 매번 실행할 수는 없으므로, 매번 Vim이 확인해서 실행하는 파일이 바로 Vim의 .vimrc 혹은 Neovim의 init.vim (XDG 기반)입니다.

## init.vim / .vimrc

지금까지는 Vim을 조작하는 방법에 대해서 알아보았는데요, 지금은 간단히 Vim의 설정을 바꾸어 그냥 ‘검은 화면’을 바꾸어 봅시다.

이런 설정은 Vim을 실행하여 command-line mode에서 :set nu 등과 같이 바꿀 수 있습니다. 그런데 이를 매번 실행할 수는 없으므로, 매번 Vim이 확인해서 실행하는 파일이 바로 Vim의 .vimrc 혹은 Neovim의 init.vim (XDG 기반)입니다.

call plug#end() 밑에 아래와 같이 추가하면 Vim 옆에 상대적인 줄번호와 색상이 추가된 것을 볼 수 있습니다:

```
1 set number
2 set relativenumber
3 colorscheme slate
```

# 자동 완성, 추가된 Text Objects

Vimtex과 TexLab 모두 자동 완성 기능을 제공하며, 각 추천어들은 <C-n>과 <C-p>로 선택할 수 있으며 <CR> (return) 키를 통해 snippet을 확장할 수도 있습니다.

# 자동 완성, 추가된 Text Objects

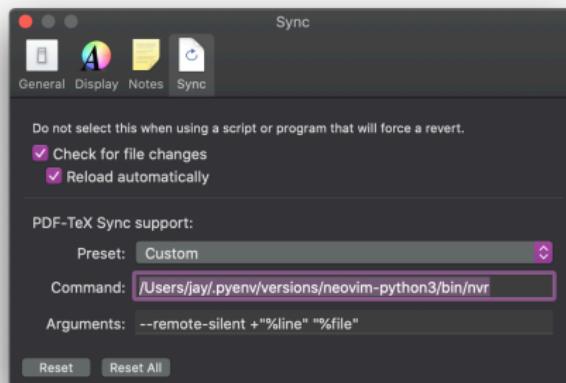
Vimtex과 TexLab 모두 자동 완성 기능을 제공하며, 각 추천어들은 <C-n>과 <C-p>로 선택할 수 있으며 <CR> (return) 키를 통해 snippet을 확장할 수도 있습니다.

Text object의 경우 Vimtex이 제공하는데, command나 환경을 쉽게 바꾸고 \*를 넣거나 빼기가 굉장히 편합니다. 예를 들어서 tsc (“toggle surrounding command”)는 명령어에 \*를 넣고 빼주며, dae (“delete an environment”)는 현재 환경 \begin{env} ... \end{env}을 지웁니다.

# 편한 Mappings

- \ll - 연속 조판 (저장할 때마다 조판하기)
- \lv - Forward search
- \le - 조판 오류 보기
- \lc - 부수 파일 삭제
- \lc - 부수 파일 및 PDF 삭제
- \lt - 문서 목차 보기
- \lt - 문서 목차 toggle
- K - texdoc 없이 package의 개요 보기

# Forward / Backward Search (SyncTeX)



Neovim은 NVIM\_LISTEN\_ADDRESS=/tmp/nvimsocket nvim로 시작하면 됩니다. 매번 칠 수 없으므로 alias를 만듭니다:

```
alias v='NVIM_LISTEN_ADDRESS=/tmp/nvimsocket nvim'
```

# P.S. Neovim에 버그가 있는 것 같아요...

The screenshot shows a GitHub issue page for a bug in Neovim's vimtex plugin. The URL is <https://github.com/lervag/vimtex/issues/1595>. The title of the issue is "[Neo)Vim 'w' behaves like 'W' in a certain scenario - Issue #1595 - lervag/vimtex". The page displays several comments from users clason, lervag, and Zeta611. The first comment by clason describes a bug where `ctrl-w` and `ctrl-W` are not mapped in vimtex and are not related to vimtex. The second comment by clason indicates they can reproduce the issue on Neovim nightly on Catalina. The third comment by lervag asks if the user can reproduce the issue in Vim. The fourth comment by clason states that they forgot how to use vim. The fifth comment by Zeta611 confirms that Vim works fine, suggesting it is a Neovim bug.

(Neo)Vim 'w' behaves like 'W' in a certain scenario - Issue #1595 - lervag/vimtex

[Neo)Vim 'w' behaves like 'W' in a certain scenario #1595

clason commented 6 minutes ago

Indeed, I can reproduce this (neovim nightly on Catalina) -- the `w` issue, I mean. Seeing as this is likely a neovim bug, it would be good to bisect it.

lervag commented 3 minutes ago

Ok, good! That's progress! Can you reproduce the issue in Vim as well?

clason commented 2 minutes ago • edited

No, because I apparently forgot how to use vim (the minimal vimrc doesn't work on vim for me), but at least I could reproduce on Linux (openSUSE Tumbleweed, again neovim head).

Zeta611 commented 1 minute ago

@lervag Vim does work fine! So apparently this is a Neovim bug..

<https://github.com/lervag/vimtex/issues/1595>

Why Vim?  
○○○○○

Vim Setup  
○○○

Hello, Vim!  
○○○○○○○○○○

T<sub>e</sub>Xnical Vim  
○○○○○○○○●○

+α  
○○○

감사합니다.

# 부록

발표 자료:

<https://github.com/Zeta611/technical-vim-kts-conf-2020>

Neovim 설정: <https://github.com/Zeta611/dotfiles>



$+\alpha$



## 추가 자료

발표가 끝난 후, ‘어떤 자료가 있으면 좋을 것 같다’는 의견이 있어서 자료를 추가합니다. 내용을 중간에 보충하려다가 업데이트된 부분을 쉽게 보실 수 있도록 뒤에 추가하여 정리합니다.

# 1. 자동 한/영 전환 i

Vim에서 한글 T<sub>E</sub>X 문서를 작성하신다면, normal 모드의 명령어를 영어로 입력하다가 insert 모드에서 한글로 입력을 하기 위해 한/영 전환을 하는 것이 상당히 불편합니다.

이를 위해서 (macOS에서) `input-source-switcher`와 Vim plugin인 `vim-xkbswitch`을 사용합니다.

# 1. 자동 한/영 전환 ii

다음과 같이 input-source-switcher을 설치합니다. cmake에 -DCMAKE\_INSTALL\_PREFIX:PATH=\$HOME을 준 이유는 홈 디렉터리에 설치하기 위함입니다. 옵션을 안 주면 기본 설치 경로에 설치됩니다. 터미널에서 다음과 같이 실행합니다:

```
git clone
→ https://github.com/vovkasm/input-source-switcher.git
cd input-source-switcher
mkdir build && cd build
cmake -DCMAKE_INSTALL_PREFIX:PATH=$HOME ..
make
make install
cd ../../..
```

# 1. 자동 한/영 전환 iii

이제 Vim에서 vim-xkbswitch를 설정하면 됩니다. `init.vim`에 다음과 같이 추가합니다:

```
1 call plug#begin('~/local/share/nvim/plugged')
2 Plug 'lyokha/vim-xkbswitch'
3 call plug#end()

4
5 let g:XkbSwitchEnabled = 1
6 let g:XkbSwitchLib =
7   ↪ expand '~/lib/libInputSourceSwitcher.dylib'
8 " 기본 경로에 설치했다면 아래와 같이 설정합니다:
9 " let g:XkbSwitchLib =
10   ↪ '/usr/local/lib/libInputSourceSwitcher.dylib'
```