

# おすすめVimプラグイン紹介

## @横田研Vim勉強会

横田理央研究室 修士2年 岩瀬 駿

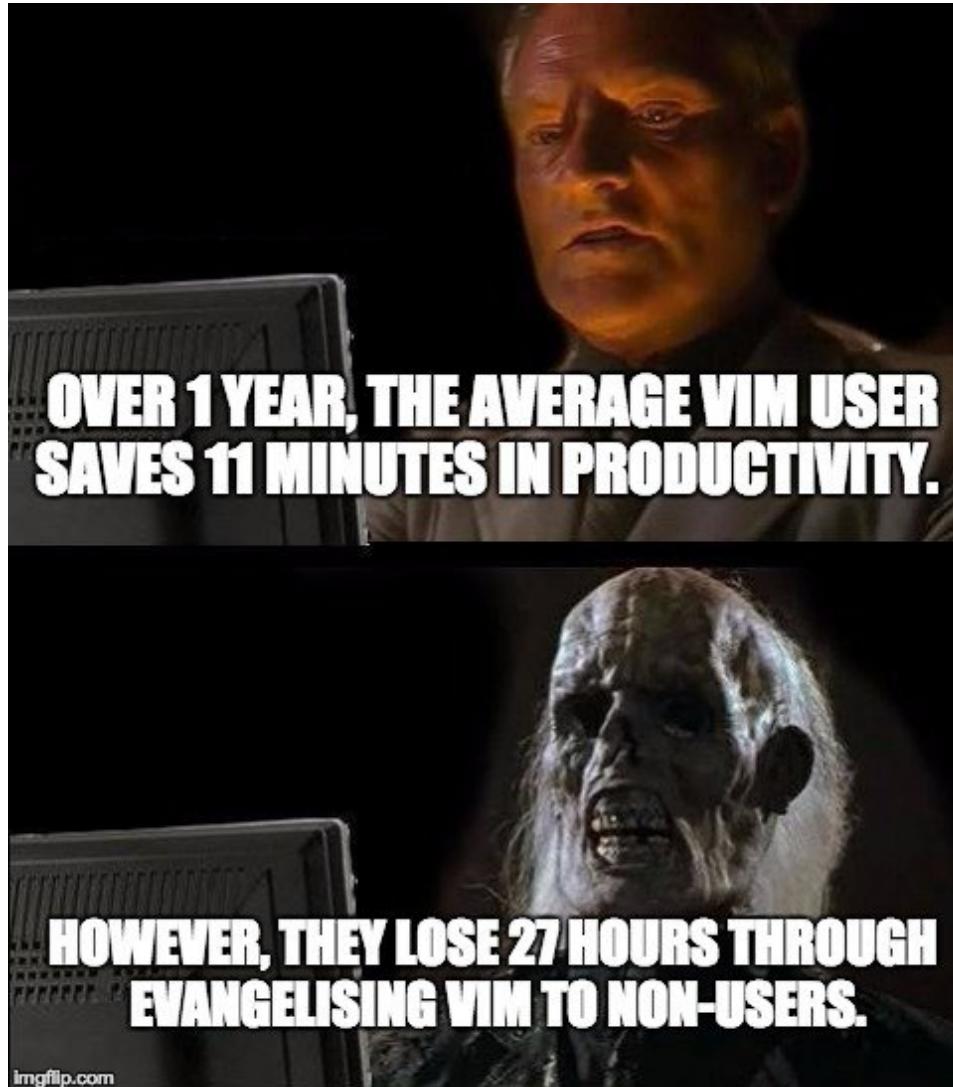


東京工業大学  
Tokyo Institute of Technology

# 自己紹介

- 氏名
  - 岩瀬 駿 (いわせ しゅん)
- 所属
  - 横田理央研究室 修士2年
  - コンピュータビジョン関連の研究をしています
- 趣味
  - Vimでコーディング
  - Vimプラグイン探し
  - Vimの布教活動
  - Emacs, IDE(Eclipse, Xcode etc...)のネガキャン

# 自己紹介

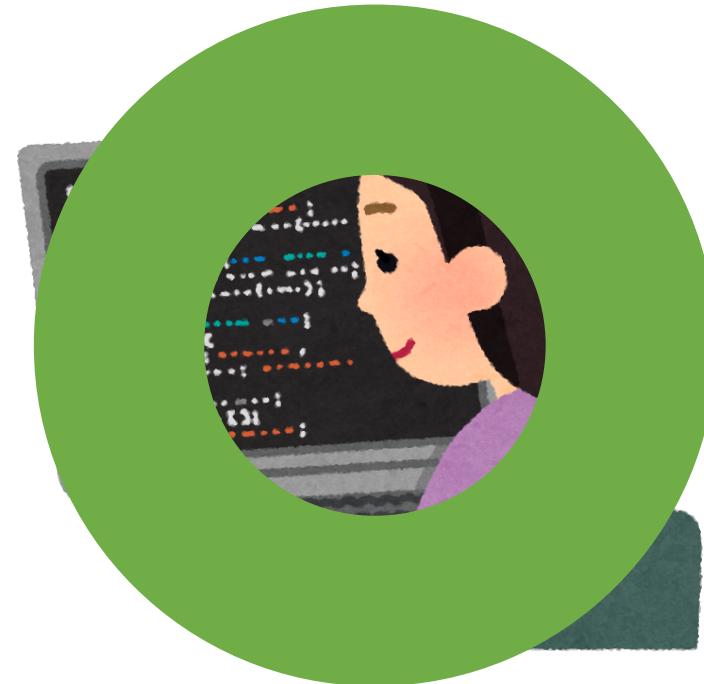


# Vimプラグインとは？

# Vimプラグインとは?

Vimをちゃんと使いこなせていますか？

初期設定のまま使っていませんか？



# Vimプラグインとは?

Vimプラグインを使えば、  
自分だけの快適な編集環境を作れる！

# Vimプラグインとは?

そういうけど…



設定めんどくさくないの?  
ゼロから書かないといけないの?  
プラグイン全部調べなきゃいけない?  
~~AtomやVisual Studioの方がいいじゃん?~~

# Vimプラグインとは?

vimrcに設定を書くだけで、  
どこでも同じ環境を簡単に再現可能！

Githubには先人のノウハウが詰まった  
すごいvimrcがたくさん落ちてる 😊

```
63 if dein#load_state(s:dein_dir)
64   call dein#begin(s:dein_dir)
65
66   call dein#add('Shougo/dein.vim')
67
68 " Plugin async loader
69   call dein#add('Shougo/vimproc.vim', {
70     \ 'build' : [
71       \ 'windows' : 'tools\\update-dll-mingw',
72       \ 'cygwin' : 'make -f make_cygwin.mak',
73       \ 'mac' : 'make -f make_mac.mak',
74       \ 'unix' : 'make -f make_unix.mak',
75     ],
76   })
77
78 " Completion
79   call dein#add('prabirshrestha/asynccomplete.vim')
80   call dein#add('prabirshrestha/async.vim')
81   call dein#add('prabirshrestha/vim-lsp')
82   call dein#add('prabirshrestha/asynccomplete-lsp.vim')
83   call dein#add('Shougo/echodoc.vim')
84   call dein#add('jiangmiao/auto-pairs')
85
86 " Linter
87   call dein#add('w0rp/ale')
88
89 " Calendar
90   call dein#add('itchyny/calendar.vim')
91
92 " Funny plugins
93   call dein#add('ashisha/image.vim')
94   call dein#add('osyo-manga/vim-nyaaancat')
```

vimrcの一例

# Vimプラグインとは?

**Vim Script**によって書かれた拡張機能  
(NeovimだとPythonでも書ける)

# Vimプラグインで何ができるの?

- ・ファイル・文字列の検索
- ・シンタックスハイライト
- ・Linter機能
- ・ファイル一覧表示
- ・補完機能
- ・他にもいろいろたくさん...

# Vimプラグインとは?

IDEでできることは**ほぼ全部**できる！！！！

全部ターミナルで作業が完結！！最高！！

# プラグイン管理

# プラグイン管理

プラグインを管理するためのプラグイン

- **dein.vim**
- NeoBundle
- Vundle
- vim-pathogen
- vim-plug

など

# プラグイン管理

日本だと暗黒美夢王(Shougo)の開発した **dein.vim** を使ってる人が多い  
速度の差は多少あるが、使い心地はどれも大差ない（個人の意見です）

```
" Syntax highlight
call dein#add('fatih/vim-go')
call dein#add('pangloss/vim-javascript')
call dein#add('mxw/vim-jsx')
call dein#add('maxmellon/vim-jsx-pretty')
call dein#add('wlangstroth/vim-racket')
call dein#add('godlygeek/tabular')

" Completion
call dein#add('prabirshrestha/asyncomplete.vim')
call dein#add('prabirshrestha/async.vim')
call dein#add('prabirshrestha/vim-lsp')
call dein#add('prabirshrestha/asyncomplete-lsp.vim')
call dein#add('Shougo/echodoc.vim')
call dein#add('jiangmiao/auto-pairs')
```

# ファイル・文字列の検索

# ファイル・文字列の検索

とりあえず, **FZF**と**Ripgrep**を入れよう

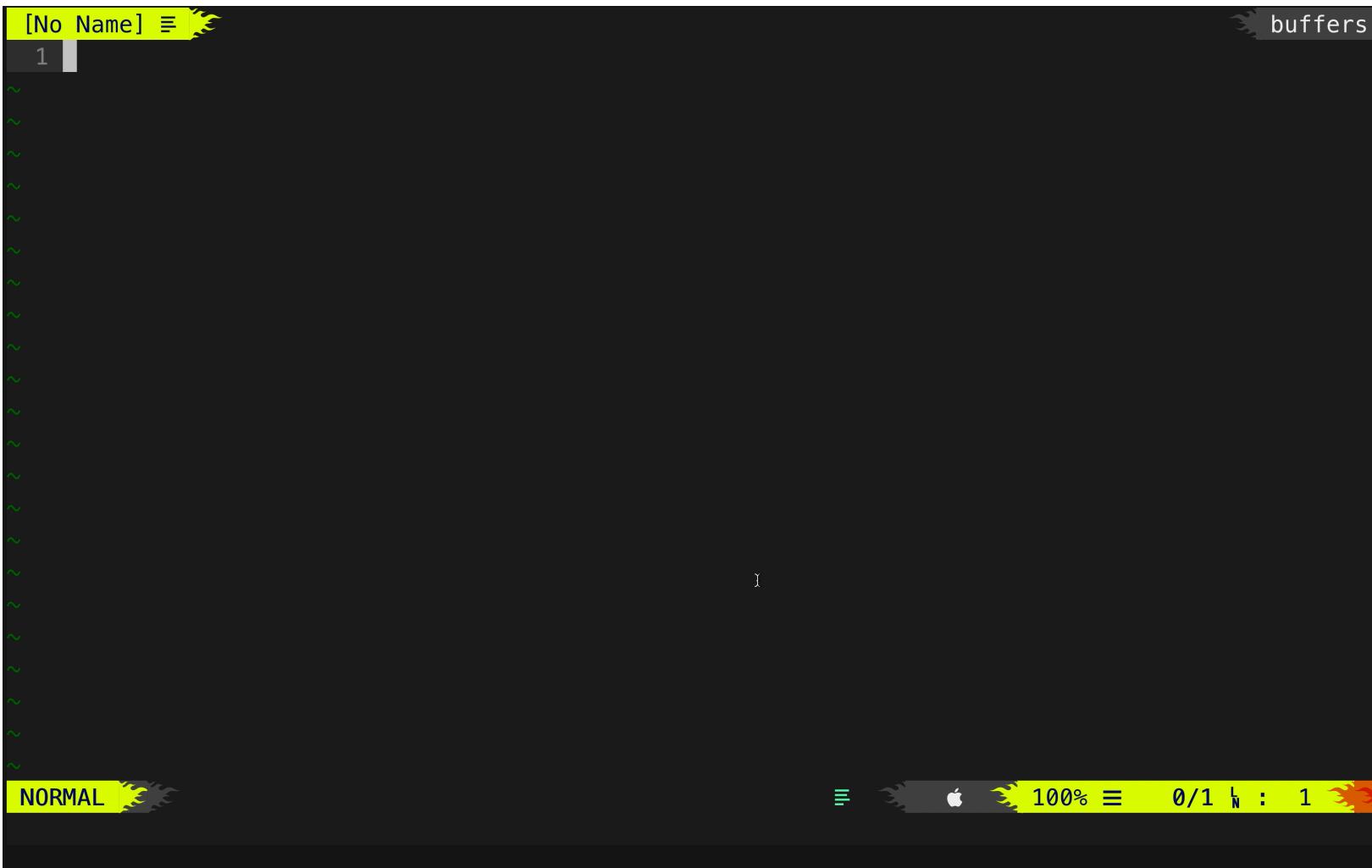
# ファイル・文字列の検索

- **FZF**は絞り込み検索のためのライブラリ
- **Ripgrep**は最も高速に動作するgrep(文字列の検索のためのコマンド)

```
"fzf.vim
if dein#tap('fzf.vim')
    command! -bang -nargs=* Rg
        \ call fzf#vim#grep(
        \   'rg -g "!node_modules/*" --column --line-number --no-heading --color=always '.shellescape(<q-args>), 0,
        \   fzf#vim#with_preview(
        \     {'options': '--exact --delimiter : --nth 3.. --preview "rougify {2..-1} | head -'.&lines.'"}, 'right:50%')
    nnoremap <silent> <C-t> :call Fzf_dev()<CR>
    nnoremap <silent> ,g :Rg<CR>
endif
```

“Ctrl + t” でファイル検索, “,g”で文字列検索ができるようにするための設定例

# ファイル・文字列の検索



# シンタクスハイライト

# シンタックスハイライト

- 可読性が向上する
- 言語ごとに異なるプラグインが提供されてる ("言語名 *highlight vim*"で検索)
  -  :syntax on とすれば初期設定でもある程度ハイライトされる
- あえて有効化しないという猛者もいるらしい？
- 詳しくは次の発表で！！

Before

```
6 int main() {  
7     string s;  
8     cin >> s;  
9  
10    int result1 = 0;  
11    int result2 = 0;  
12  
13    for (int i = 0; i < s.size(); i++) {  
14        int bit1 = i % 2;  
15        int bit2 = (i+1) % 2;  
16        int s_int = s[i] - '0';  
17    }
```



After

```
7 int main() {  
8     string s;  
9     cin >> s;  
10  
11    int result1 = 0;  
12    int result2 = 0;  
13  
14    for (int i = 0; i < s.size(); i++) {  
15        int bit1 = i % 2;  
16        int bit2 = (i+1) % 2;  
17        int s_int = s[i] - '0';
```

# シンタックスハイライト

## Python

- vim-python/python-syntax
- numirias/semshi (Semantic highlight for Neovim)

## C++

- octol/vim-cpp-enhanced-highlight
- bfrg/vim-cpp-modern

# シンタックスハイライト

## Javascript

- pangloss/vim-javascript
- mxw/vim-jsx (For JSX)

## Golang

- fatih/vim-go

# Linter

# Linterとは？

## 文法のチェックをしてくれる機能

- タイポでおかしな変数名に代入している
- if文の最後にコロンを忘れがち (Python)
- 動的型付け言語に汚染されると型定義忘れがち (C, C++, Golang)
- 定数変数に代入しようとしてる (Javascript等)

# Linterとは？

Linterがあるだけで、開発速度が全然変わる  
(人間は小さなミスになかなか気づけない...)

```
68 if __name__ == '__main__':
69     name = input().strip()
E> 70     if name == 'sh8'
>> 71         print('Hi! %s' % name)
72     else:
73         print('Good Bye! %s' % name)
NORMAL test.py[+] python
E999: SyntaxError: invalid syntax
```

```
18 int main() {
>> 19     n = 10;
>> 20     max_height = 0;
21     int result = 0;
>> 22     cin >> n;
NORMAL <90413/b/main.cpp[+] cpp
use of undeclared identifier 'max_height'
```

昔は**Syntastic**というLintエンジンが  
よく使われていた

# Linterとは？

**Syntastic**はファイル保存時に  
同期的にLintを行う

# Linterとは？

Lintの実行中は他の操作ができない

# Linterとは？

でも今は**ALE**がある！  
(Asynchronous Lint Engine)

# ALEとは？

非同期でLintを実行してくれる  
Vimプラグイン

# ALEとは？

```
main.cpp 
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 int main() {
7     int n;
8     int max_height = 0;
9     int result = 0;
10
11     cin >> n;
12
13     int mountains[n];
14     for (int i = 0; i < n; i++) {
15         cin >> mountains[i];
16     }
17
18     for (int i = 0; i < n; i++) {
19         int height = mountains[i];
20         if (height >= max_height) {
21             result++;
}
NORMAL  main.cpp
cpp 
```

# ALEとは？

それぞれの言語ごとのLinterは各自インストール

- C++: cppcheck, clang-tidy, clang++ (コンパイラに含まれている)
- Javascript: JSLint, JSHint, ESLint
- Python: pycodestyle(pep8), flake8, pylint
- Golang: gofmt (Golangについてくる)

ファイル

# ファイル

基本的なファイル操作を  
行うためのツール

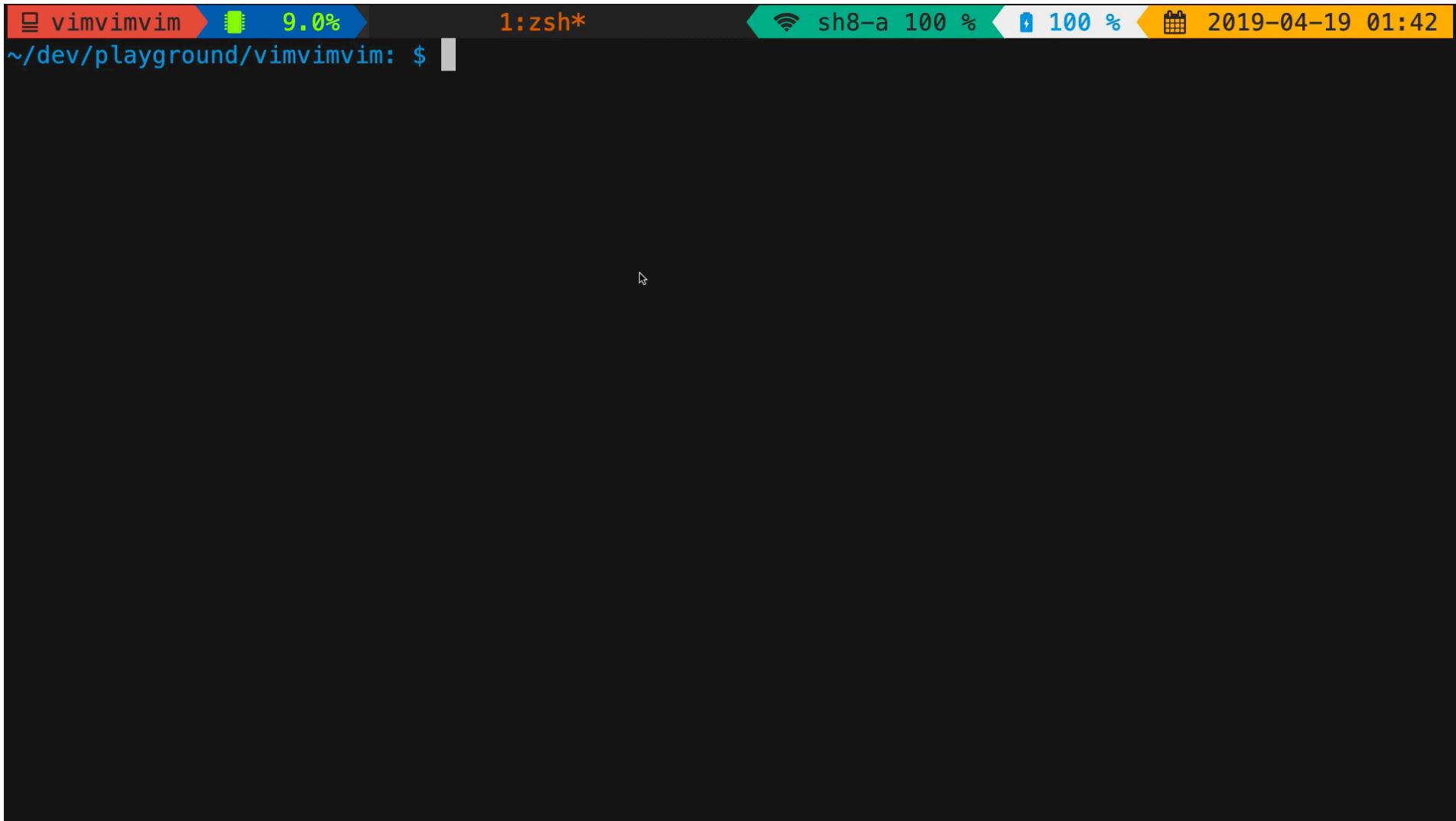
# ファイルでできること

## ファイル・ディレクトリの

- 一覧を表示
- 開く
- 作成
- 削除
- 移動
- コピー
- 名前の変更
- 検索

など

# ファイル



# ファイル

**Netrw**というデフォルトのファイルが  
実はかなり便利...

# ファイル

けれども、サードパーティのファイルは、機能が豊富かつカスタマイズ性が高い！

# ファイル

サードパーティは、  
NERDTreeとvimfilerの一騎打ち状態だった

# ファイル

Vimfilerは開発が終了し,  
**defx.nvim**へ移行中...

# ファイル

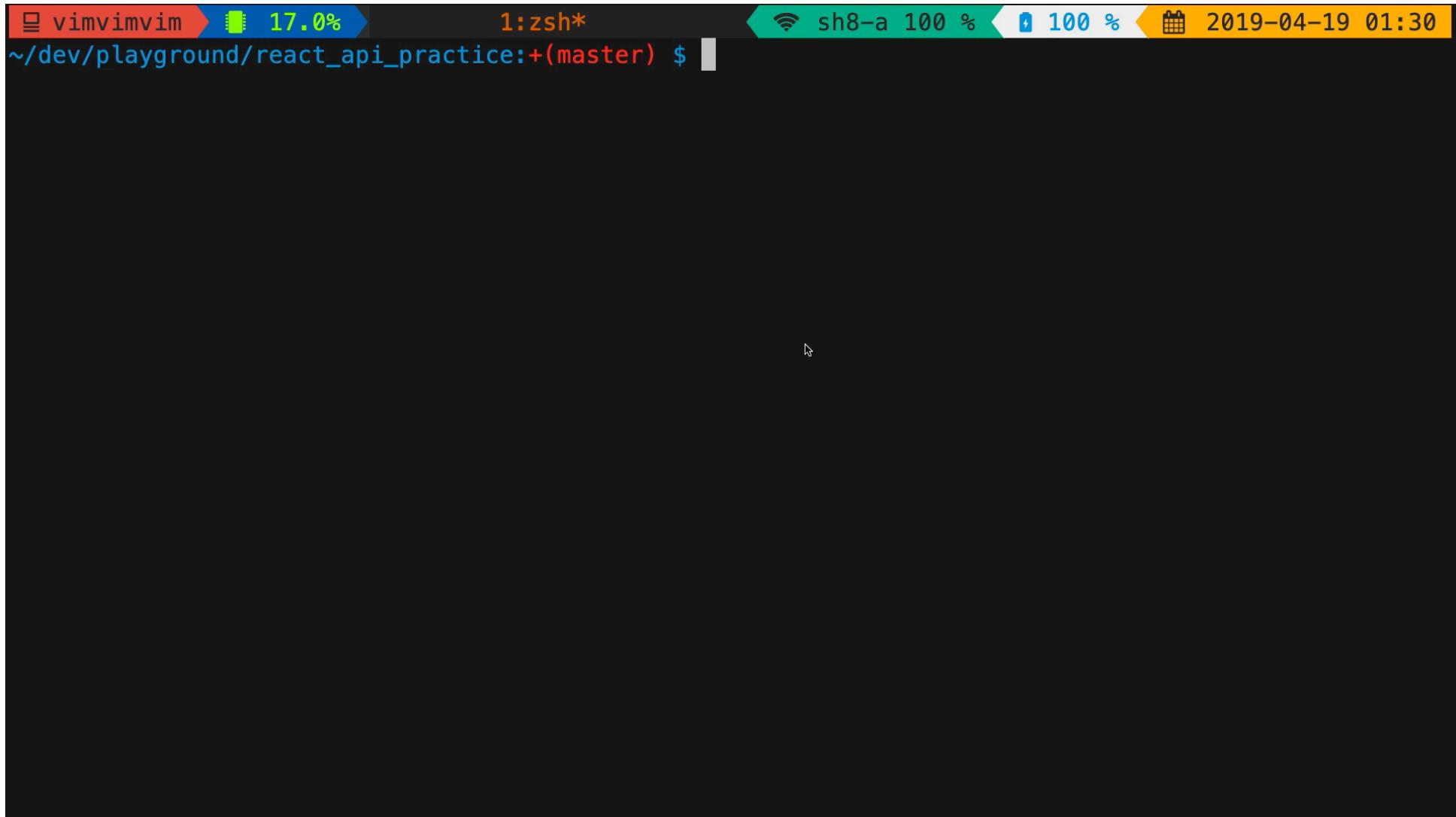
個人的には,  
**defx.nvim**一択

# ファイル

10000ファイルを開くためにかかる時間の比較

|                  |           |
|------------------|-----------|
| netrw            | 15秒       |
| NERDTree         | 15秒       |
| Vimfiler         | 計測不能      |
| <b>defx.nvim</b> | <b>3秒</b> |

# ファイル



# ファイル

defx.nvimはまだ情報があまりないので、  
初心者向けではないかも(2019/04現在)

# ファイル

それでも試したい方は, [github.com/sh8/vimrc](https://github.com/sh8/vimrc)を  
参考にしてみてください！

# 補完機能

# 補完機能

Vimはデフォルトで,  
Ctrl+x, Ctrl+pを使った補完機能がある

# 補完機能

あくまで定義済みの  
変数名や特定の行の補完のみ...

# 補完機能

いろいろな言語で  
関数名やその引数まで補完したい！！

# 補完機能

- **deoplete.nvim** (非同期補完のためのプラグイン)
- **vim-lsp** (VimのLanguage Server Protocol実装)

を使えば簡単に実現できる！！

つまりどんなことができるの？ (Python編)

# 補完機能

The screenshot shows a terminal window with two nvim instances. The left nvim window has the file 'cnn.py+' open, displaying Python code for a convolutional neural network. The right nvim window shows the output of the 'python' command, which is the execution of the script. The status bar at the bottom indicates the file is in 'NORMAL' mode, the buffer is 'cnn.py[+]', the encoding is 'utf-8', and the status is '91% 55/60'. The page number '1' is also visible. The terminal prompt shows the command was run from the directory '~/dev/playground/cnn.py'.

```
nvim  nvim  nvim  buffers
cnn.py+  python  utf-8  91%  55/60  1  w:1(L15)
~"/dev/playground/cnn.py" 60L, 2157C written
```

```
35     n_out = math.floor((n_in - k + 2*p)/s) + 1
36     actualP = (n_out-1)*s - n_in + k
37     pL = math.floor(actualP/2)
38
39     j_out = j_in * s
40     r_out = r_in + (k - 1)*j_in
41     start_out = start_in + ((k-1)/2 - pL)*j_in
42     return n_out, j_out, r_out, start_out
43
44
45 def printLayer(layer, layer_name):
46     print(layer_name + ":")
47     print("\t n features: %s \n \t jump: %s \n \t receptive size: %s \t start: %s " %
48           layer[0], layer[1], layer[2], layer[3]))
49
50
51 layerInfos = []
52 if __name__ == '__main__':
53     print("-----Net summary-----")
54     currentLayer = [imsize, 1, 1, 0.5]
55
56     for i in range(len(convnet)):
57         currentLayer = outFromIn(convnet[i], currentLayer)
58         layerInfos.append(currentLayer)
59         printLayer(currentLayer, layer_names[i])
60     print("-----")
```

つまりどんなことができるの？ (C++編)

# 補完機能

The screenshot shows a terminal window with a dark background. On the left, there is a vertical file tree icon. The main area displays a C++ program named `main.cpp`. The code includes standard library includes, namespace declarations, function definitions for addition and subtraction, and a main function that reads two integers from standard input and prints them to standard output. The word `cin` is highlighted in blue, indicating it is being typed or selected. The status bar at the bottom provides information about the current mode (NORMAL), the file name (`main.cpp`), the programming language (`cpp`), encoding (`utf-8`), and the current line and column (`76% 16/21 3 W:4(L17)`). The top right corner of the window has a "buffers" tab.

```
main.cpp+ 🌈
2 #include <string>
3
4 using namespace std;
5
6 int add(int a, int b) {
7     return a + b;
8 }
9
10 int sub(int a, int b) {
11     return a - b;
12 }
13
14 int main() {
15     int a, b, c, d;
16     cin >> a >> b;
17     cout << c << endl;
18     cout << d << endl;
19
20     return 0;
21 }
```

NORMAL 🌈 main.cpp[+] cpp 🌈 utf-8 🌈 76% 16/21 3 W:4(L17)

# LSP (Language Server Protocol) とは?

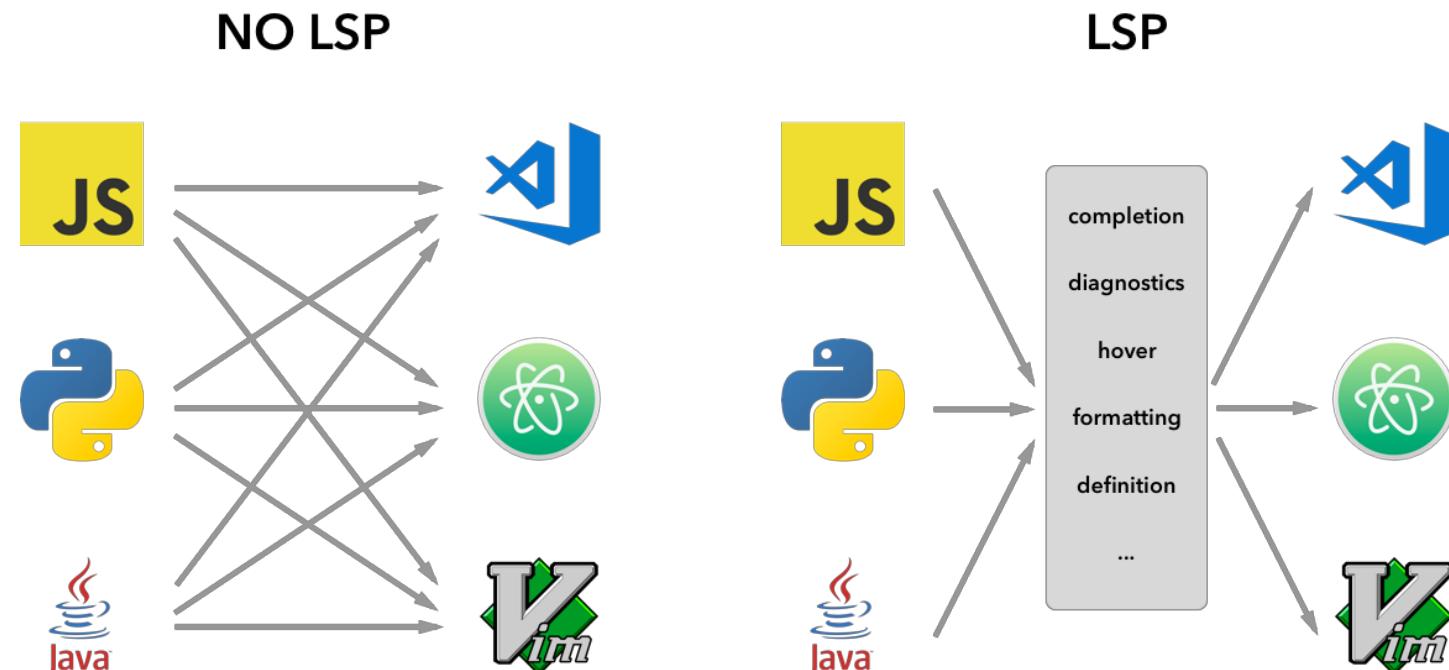
Microsoftが2016年6月に公開  
(結構最近！！)

# LSP (Language Server Protocol) とは?

ソースコードを解析して、型や変数の自動補完や、  
Linter、エラー解析・修正などを行う機能を持つ

# LSP (Language Server Protocol) とは?

様々なエディターで共通して利用できるプロトコルなので、  
言語 × エディターの組み合わせごとに、  
補完エンジンやLinterを開発する必要がなくなった！！



# LSP (Language Server Protocol) とは?

どのエディターを用いても、  
同じ補完エンジン・Linterを使える！

# LSP (Language Server Protocol) とは?

言語毎にLSPの実装があるが,  
どれも機能が充実していて質が高い！

# 補完機能

- **deoplete.nvim** (非同期補完のためのプラグイン)
- **deoplete-lsp** (deopleteとvim-lspをつなげてくれるプラグイン)
- **vim-lsp** (Vimにおける, Language Server Protocol実装)
- **echodoc** (Vimの下部分に関数に関する情報を表示)

を使えば再現できるのでgithub.com/sh8/vimrcを参考に(ry

# その他の便利プラグイン

# その他の便利プラグイン

- [vim-scripts/YankRing.vim](#)
  - ヤンクした履歴を保持してくれる
  - Ctrl + pをすると過去のヤンクしたバッファに遡れる
- [rhysd/accelerated-jk](#)
  - 縦のjk移動がすごく速くなる
- [tomtom/tcomment\\_vim](#)
  - コメントアウトをショートカット

# その他の便利プラグイン

- [vim-airline/vim-airline](#)
  - ステータスバーがかっこよくなる
- [airblade/vim-gitgutter](#)
  - Git管理されてるファイルの差分を表示
- [lervag/vimtex](#)
  - VimでTeX, 書いちゃう？

(番外編) カレンダーの表示

# (番外編) カレンダーの表示

Vimでカレンダー管理したい！  
Vimからもう離れたくない！

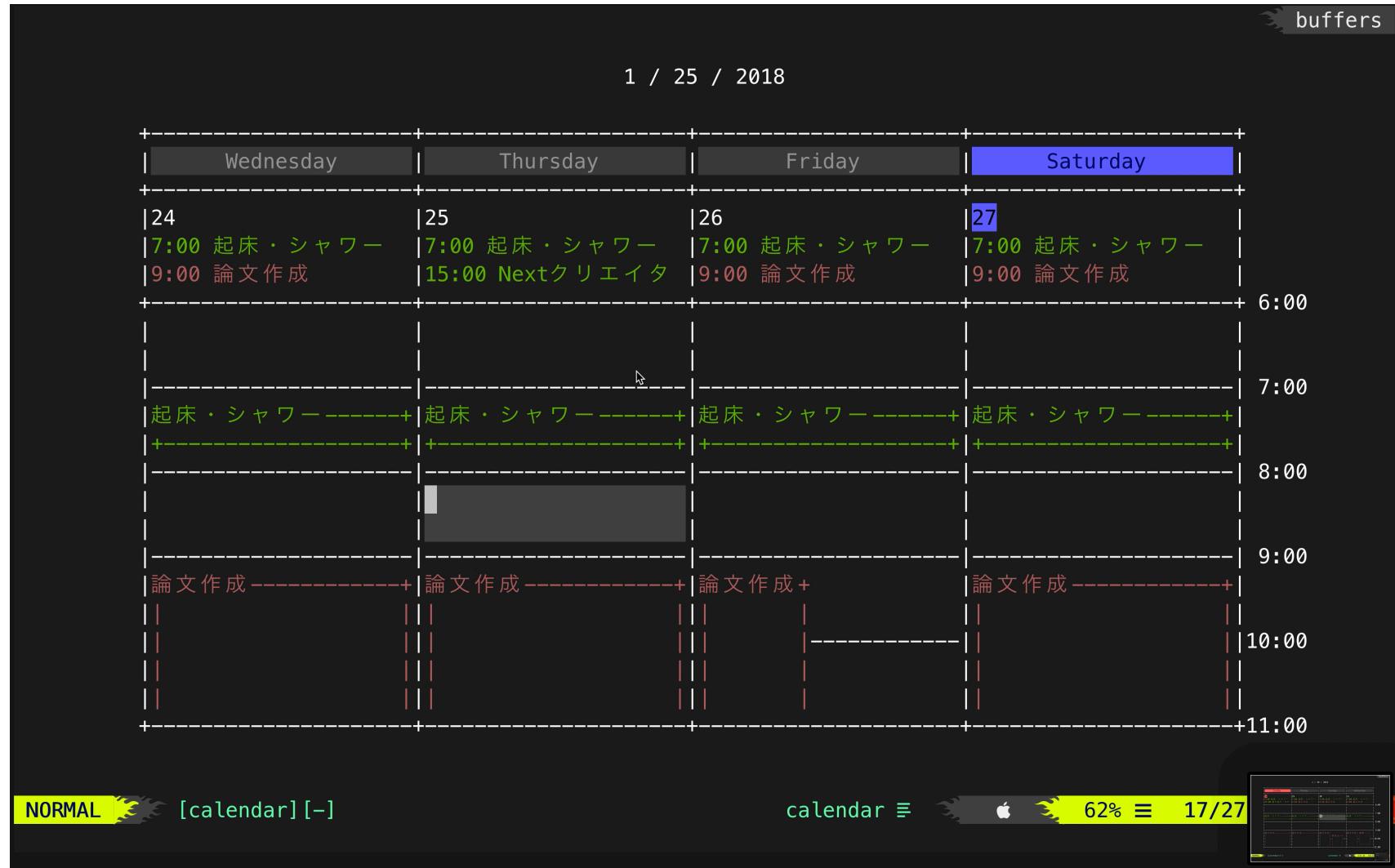
# (番外編) カレンダーの表示

`calendar.vim` を使えば、  
GoogleカレンダーをVimから管理できる！

# (番外編) カレンダーの表示



# (番外編) カレンダーの表示



# 最後に

Vim沼は深いので気をつけましょう

# 最後に

vimrcは公開しています

<https://github.com/sh8/vimrc/>