

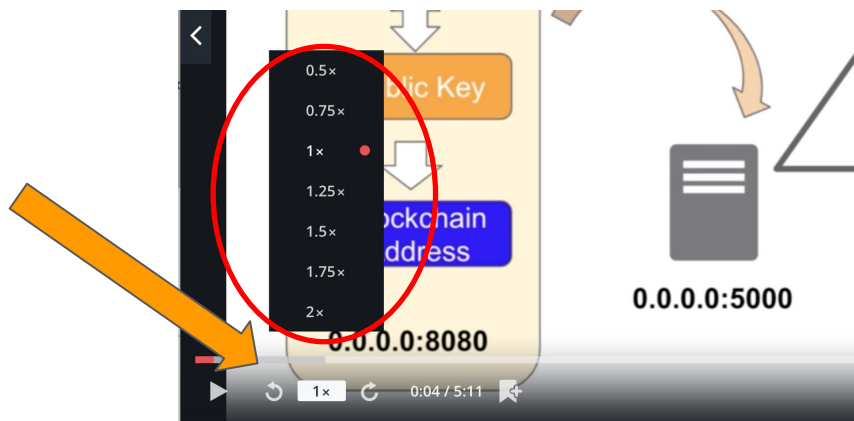
# ZSH + NEOVIM + TMUX

# 講義を受講する前の注意事項

- 3rdパーティーのアプリのインストールによる不具合や問題が起きた場合には、責任が持てませんのでご了承ください。
- 講師は、Windowsと同様な内容のレクチャーをMacを用いて講義を進めていきますが、WindowsとMacでの動作が異なる可能性がございます。
- Windowsをお使いの方は、WSL2をインストールできるOSとPCである必要があります。そのため、OSはWindows 10以上でハードウェアがHyper-Vに対応している必要があります。WSLでは講義内容は確認しておりませんのでご了承ください。
- 解説は日本語で行いますが、使用するOSなどは英語表記ですので、日本語OSとは表記が異なります。
- Zshのバージョンは5.8、Vimのバージョン8.2、NeoVimのバージョンはv0.4.3、Tmuxのバージョンは3.1bを使用します。新しいバージョンでも同様なコマンドとなる可能性が高いですが、バージョンの違いにより動作が異なることがあります。

# 講義に関する注意事項

- 講義と関係のない質問はお答えできないことがあります。
- 質問する前にGoogleなどで検索してくださいと幸いです。
- 講義のスピードを調整してください。



# コードに関する注意事項

- オンラインコース、Youtube、ブログ、電子書籍等でコードを外部に公開して、教授するようなレクチャーによる商用利用はご遠慮ください。ブログ等で、数行のコードの紹介と本コースへの宣伝リンクがわかりやすい形で行ってくださるアフィリエイト等の使用は可能です。
- Github等でのソースの公開、再配布はお控えください。
- コードは商用利用は可能ですが、上記の条件付きとなります。もしご不明な点がございましたらご連絡ください。

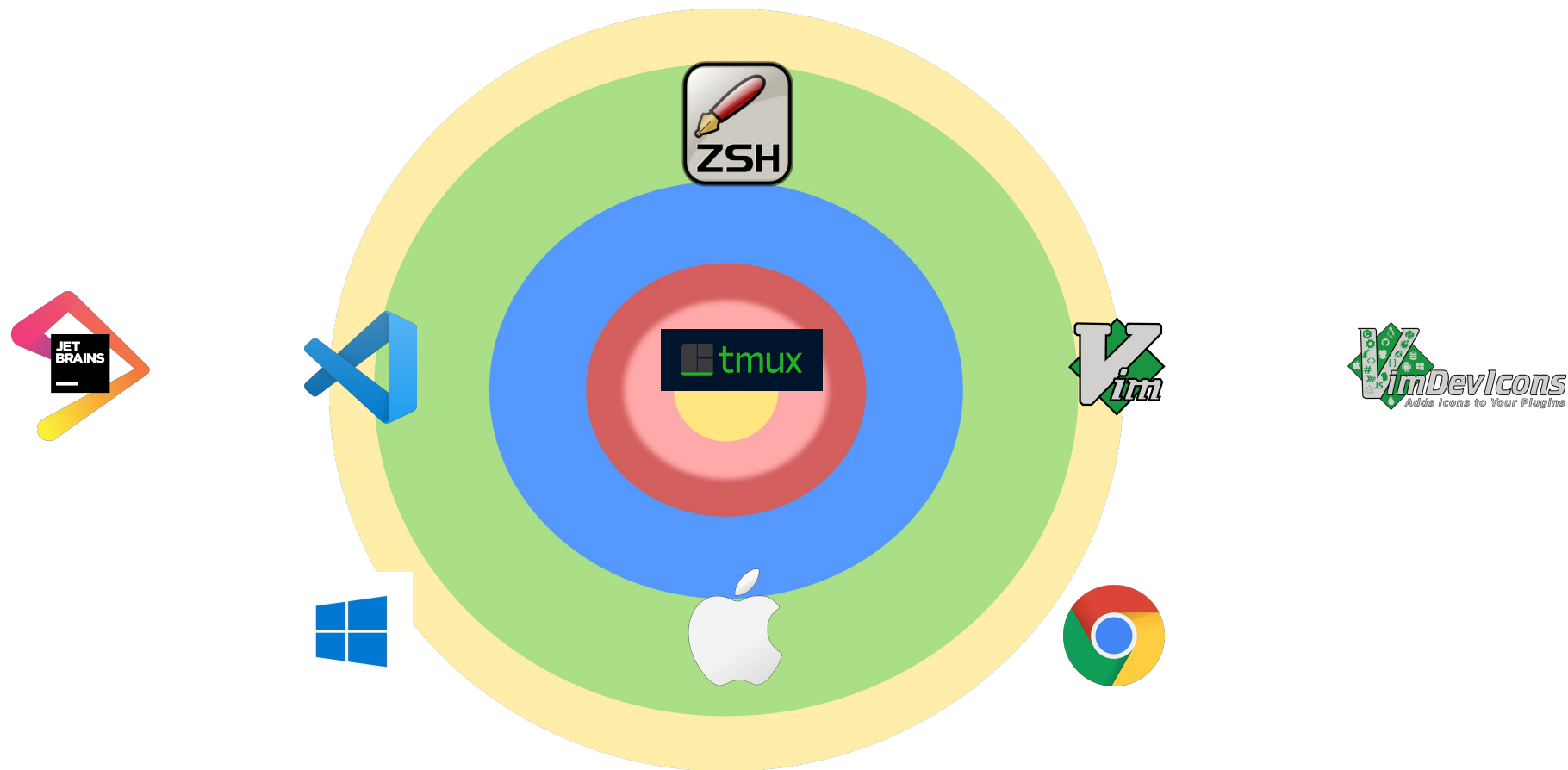
# 事前知識

10段階のうち

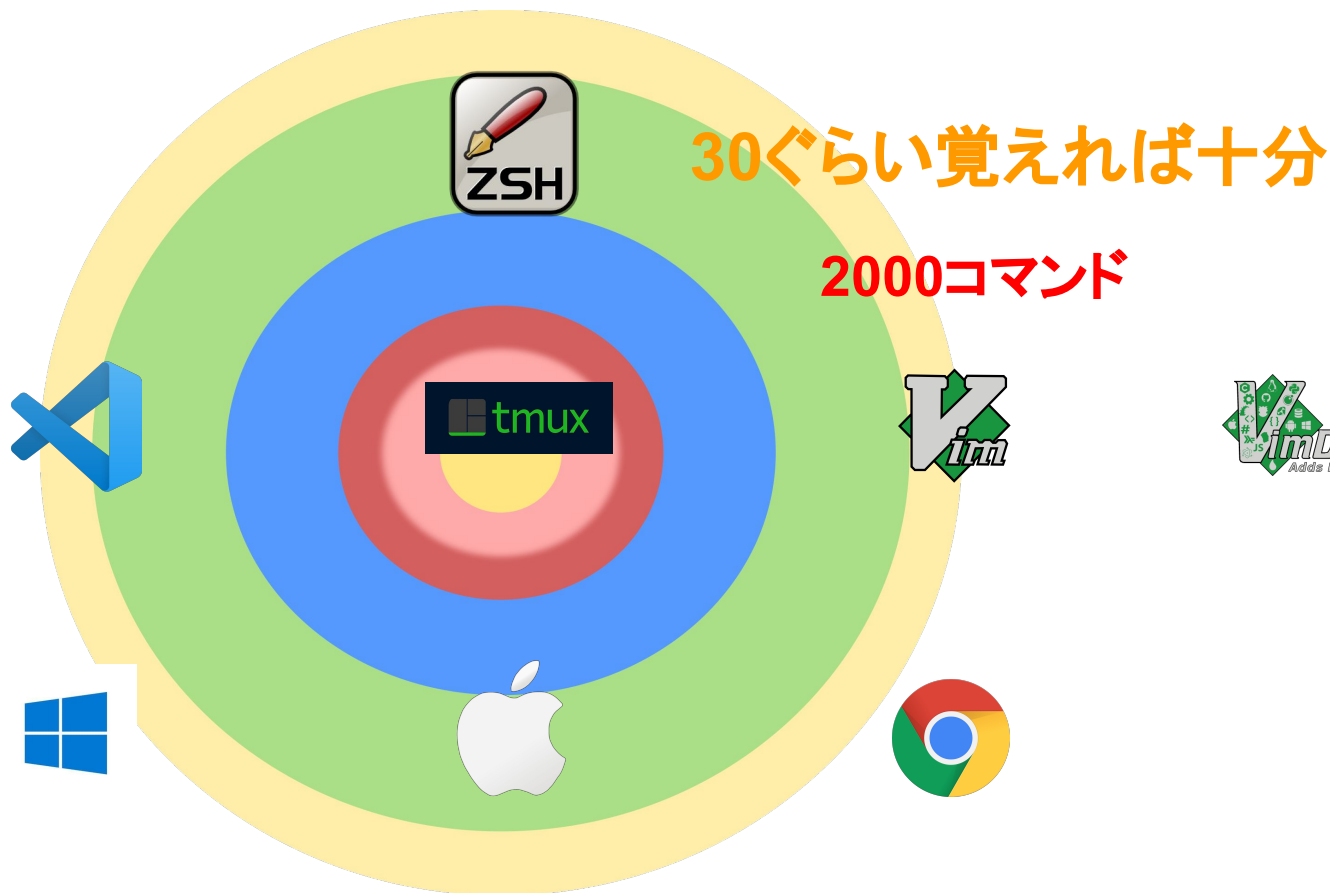
Shell (Bash/Zsh)	1
Vim	0
tmux	0
Python/Go	1
Mac/Windows/Linux Command (cd, pwd, ls, ps, echo, grep, find etc)	3
Environment Settings (install package, IDE, WSL2 etc)	8

\*コース内でもクイックスタート的に説明しておりますので必須ではありません。

全ての便利コマンドは覚えきれないので、各アプリの良い部分だけ使い覚えていく



# できる限りマウスを使わない



 **VimDevIcons**  
Adds Icons to Your Plugins

# キーボード設定

- USキーボードが便利(日本語キーボードでも問題ありません)
- Caps Lock と Control を入れ替える (HHKBなどはデフォルトで交換されている )
  - Windowsの方は、Windows PowerToys toolsをインストールして入れ替え可能です。
- 薄型か厚めか
- 静音かどうか
- テンキーは必要か
- BluetoothかUSBワイヤレスか



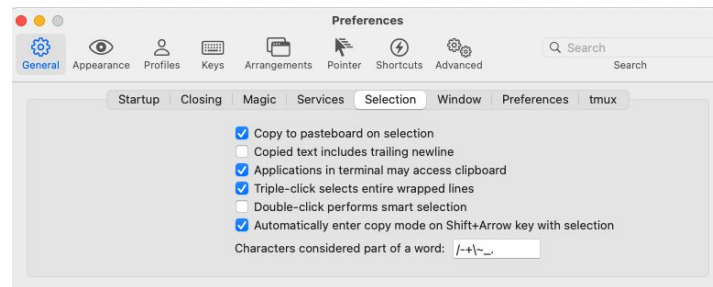
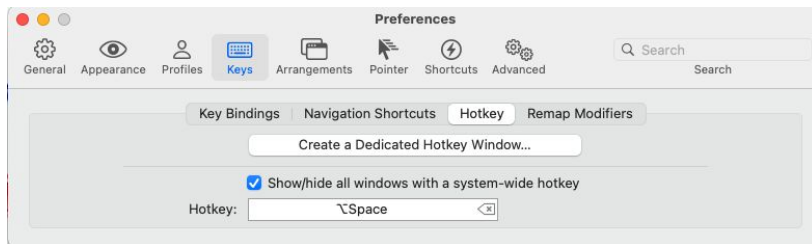
## 4つ割り当てで混乱

A diagram of a standard QWERTY keyboard layout. A vertical line, colored blue on the left and red on the right, divides the keyboard into two halves. The keys are labeled with their standard symbols and characters. The left half includes keys like ESC, F1-F5, 1-6, TAB, Q-T, CAPS LOCK, A-V, SHIFT, FN, CONTROL, OPTION, and COMMAND. The right half includes keys like F6-F12, 7-0, DELETE, Y-P, H-N, M, COMMA, SEMI, SLASH, SHIFT, COMMAND, OPTION, and arrow keys.

よく使うControlではない

# iTerm (Mac)

- カラー設定 (IDE白, Terminal黒 etc)
- Show/Hide設定
- 画面の分割はTmuxでも
- Applications in terminal my access clipbaordの設定

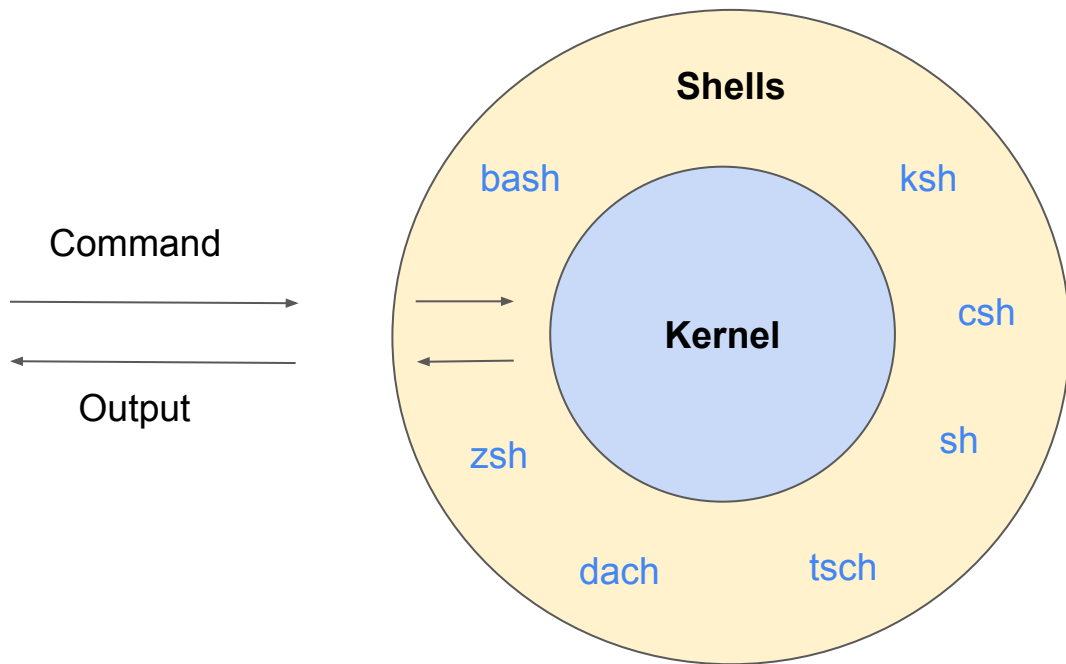
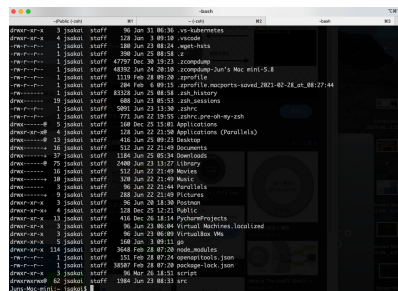


# iTerm (Mac)

command + t	新しいタブの作成
command + w (control + d)	タブの削除
command + →← or 数字	タブの移動
command + shift + h	クリップボードの履歴の呼び出し
command + f	検索
command + + / -	拡大縮小
command + enter	フルスクリーンと解除

# シェルとは

OSの中核を構成するソフトウェアKernerllに提供する機能へのアクセスを提供する



# bash vs zsh

- 2019年 MacOS X Catalinaより標準のシェルをzshに変更すると発表した
- bashとzshは一部互換性がないが、bashでできることはほぼzshでできる

bash	zsh
Auto completionはZshほど早くない	Auto completionが早い
プロンプトの設定が単調	プロンプトの設定が豊富
コンフィグしにくい	コンフィグしやすい
プラグインがzshほどない	多くのプラグインがある

# Macユーザーの準備

## brewのインストール

<code>zsh --version</code>	zshのバージョン確認
<code>echo \$SHELL</code>	現在のシェルの確認
<code>cat /etc/shells</code>	設定できるシェルの確認
<code>chsh -s /bin/zsh</code>	シェルの変更
<code>brew install zsh</code>	zshのインストール

※ zshの詳細い設定はコースの後半に行います

# Macユーザーの準備

<code>screen -v</code>	スクリーンのバージョン確認
<code>brew install screen</code>	スクリーンのインストール
<code>tmux -V</code>	tmuxのバージョン確認
<code>brew install tmux</code>	tmuxのインストール
<code>vim --version</code>	Vimのバージョン確認
<code>brew install vim</code>	Vimのインストール
<code>nvim --version</code>	NeoVimのバージョン確認
<code>brew install neovim</code>	NeoVimのインストール

# Cmder (Windows)

- カラー設定 (IDE 白, Terminal 黒)
- Show/Hide 設定
- 画面の分割はTmux (コピーがやりやすい)

control + t	新しいタブの作成
control + d	タブの削除
command + 番号	タブの移動
control + mouseのスクロール	拡大縮小



Search (Ctrl+F)



Storage:

C:\cmdr\vendor\conemu-maximus5\ConEmu.xml

Export...

## General

- Fonts
- Size & Pos
- Appearance
- Quake style
- Background
- Tab bar
- Confirm
- Task bar
- Update

## Startup

- Tasks
- Environment

## Features

- Text cursor
- Colors
- Transparency
- Status bar
- App distinct

## Integration

- Default term
- ComSpec
- Children GUI
- ANSI execution

## Keys &amp; Macro

- Keyboard

## General settings

Choose your startup task:

{WSL::Ubuntu}

&gt;&gt;&gt;

Interface language:

en: English

Choose color scheme:

Monokai



Minimize/Restore hotkey:

Ctrl+`

☐ Single instance mode (use existing window instead of running new instance)☐ Quake-style slide down from the top of the screen☐ Support special hotkeys (Win+Number, Win+Tab, PrntScr and so on)☒ Inject ConEmuHk.dll into all processes started in ConEmu tabs☒ Enable automatic updates☒ 'Stable'☐ 'Preview'☐ 'Alpha' builds<https://conemu.github.io/en/SettingsFast.html>

Donate

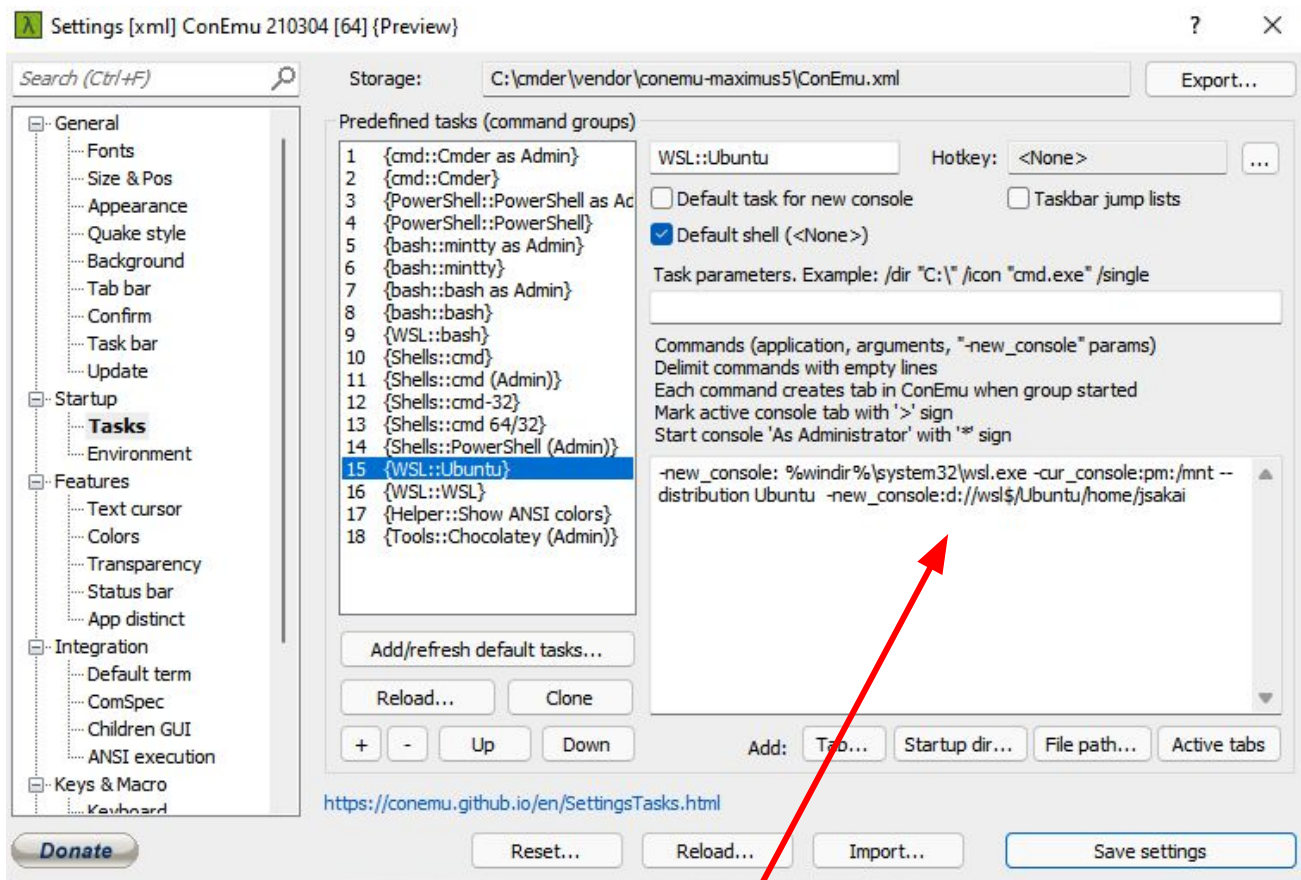
Reset...

Reload...

Import...

Save settings

Save settings  
Don't close



-new\_console:d://wsl\$/Ubuntu/home/{Username}を追加  
ex) -new\_console:d://wsl\$/Ubuntu/home/jsakai

# WSL(Linux環境) + Zshのインストール

- Control Panel -> Programs -> Windows Features
  - Hyper-Vを有効
  - Linux用Windowsサブシステムを有効
  - 仮想マシンプラットフォームを有効

<code>wsl --install</code>	wsl2 (ubuntu)がインストールされる
<code>wsl -l -v</code>	wslのバージョンの確認
<code>zsh --version</code>	zshのバージョン確認
<code>echo \$SHELL</code>	現在のシェルの確認
<code>cat /etc/shells</code>	設定できるシェルの確認
<code>chsh -s /bin/zsh</code>	シェルの変更

※ zshの詳しい設定はコースの後半に行います

# Zshのインストール

<code>zsh --version</code>	zshのバージョン確認
<code>echo \$SHELL</code>	現在のシェルの確認
<code>cat /etc/shells</code>	設定できるシェルの確認
<code>apt-get install zsh</code>	シェルのインストール
<code>chsh -s /bin/zsh</code>	シェルの変更

※ zshの詳しい設定はコースの後半に行います

# 各パッケージのインストール

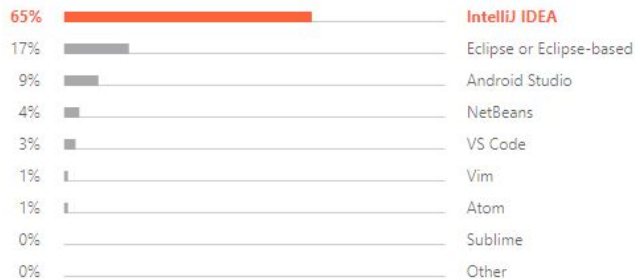
screen -v	スクリーンのバージョン確認
apt-get install screen	スクリーンのインストール
tmux -V	tmuxのバージョン確認
apt-get install tmux	tmuxのインストール
vim --version	Vimのバージョン確認
apt-get install vim	Vimのインストール
nvim --version	NeoVimのバージョン確認
apt-get install neovim	NeoVimのインストール

Vim

# IDE(VS code or JetBrains) VS Vim (NeoVim)

- VimをIDE化するのではなく補助的に
- がっつりと開発するにはIDEの機能を使うのが良い
  - IDEにはVimで操作できるPluginをインストールし開発
  - 開発以外はNeoVimを使うなど使い分ける
- VS code VS JetBrains

Which IDE / editor do you use the most for Java development?



What IDE / editors do you primarily use when writing Rust code?



# VIM vs NeoVim

- Vimができることは、ほぼ NeoVimでできる。
- 使い方はほぼ一緒なので、Vimを習得してもNeoVimを習得してもスキルレベルはほぼ一緒
- 今後の世間の動向で切り替えれば良い

Vim	NeoVim
Less extensible	More extensible
LSP Support depends on other plugins	LSP Support inbuilt
Plugins	More powerful pulugins



# VIM vs NeoVim

Vim	NeoVim
コンフィグの場所 ~/.vimrc	コンフィグの場所 ~/.config/nvim/init.vim
起動コマンド vi or vim	起動コマンド nvim
Viewモードの起動 view	Viewモードの起動 nvim -R

```
$ open -a TextEdit ~/.zshrc (Mac)
```

```
$ noepad.exe ~/.zshrc etc (Windows)
```

```
# 以下を追加
```

```
alias vi="nvim"
```

```
alias vim="nvim"
```

```
alias view="nvim -R"
```

```
$ exec zsh
```

# Vim 入門 1

vim	起動
i	Insert Mode
esc	Normal mode
k (Normal Mode)	↑に移動
j (Normal Mode)	↓に移動
h (Normal Mode)	←に移動
l (Normal Mode)	→に移動
:q!	保存しないで編集終了
view	Viewモードで起動

## Vim 入門 2

<code>:w</code> ファイル名	名前をつけて保存
<code>:w</code>	上書き保存
<code>:q</code>	編集終了
<code>:q!</code>	保存しないで編集終了
<code>:wq</code>	保存して終了
<code>i</code>	Insert mode
<code>a</code>	次の文字から Insert mode
<code>view</code>	Viewモードで起動

# Vim 入門 3

vim /tmp/test/test.txt	ファイル指定で開く
i	インサートモード
a	次の文字からInsert mode
x (Normal mode)	1文字削除
dd (Normal mode)	1行削除 (コピー可能)
dw (Normal mode)	単語の削除
2 dd (Normal mode)	2行削除
u (Normal mode)	Undo
Control + r (Normal mode)	Undo redo

# Vim 入門 4

yy (Normal mode)	一行コピー
p (Normal mode)	下にペースト
P (Normal mode)	現在行にペースト
2 yy (Normal mode)	2行コピー
.	Repeat

# Vim 入門 5

: or /	Command mode
!python /tmp/a.py (Command mode)	コマンド実行
!!	前のコマンドを実行

# Vim 入門 6

<code>:set number</code>	行番号を表示する
<code>:10</code>	10行目に移動
<code>\$ (shift + 4)</code>	行末に移動
<code>0</code>	先頭に移動
<code>^</code>	インテントの先頭に移動
<code>{</code>	段落ごとに上に移動
<code>}</code>	段落ごとに下に移動
<code>[[</code>	セクションごとに上に移動
<code>]]</code>	セクションごとに下に移動

# Vim 入門 6

:1 or gg	ファイルの先頭に移動
G	ファイルの最後に移動
CTRL + o	移動前に戻る



# Vim 入門 7

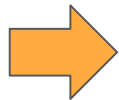
<code>/ {pattern} ex) /test</code>	検索
<code>n</code>	次の検索結果に移動
<code>N (shift + n )</code>	前の検索結果に移動
<code>R</code>	置換
<code>dw</code>	Rよりも単語削除をよく使うケースが多い
<code>:%s/search/replace/g</code>	一括置換
<code>:%s/search/replace/gc</code>	確認しながら置換

# Vim 入門 8

o	選択行の下に空白を入れ、インサートモード
O (shift + o)	選択行に空白を入れ、インサートモード
J (shift + j)	行の連結
>	右にインテントを移動
<	左にインテントを移動
v (visual mode) 移動してから y (yank)	カーソル指定コピー

# Vim 入門 9

1 visual block mode	Control + v
2 行を選択	j or k
3 先頭に挿入する場合	:norm I # or //
4 行末に追加の場合	:norm A # or //



vim commentary を使うのが楽

# Vimのコンフィグ設定

```
$ vim ~/.config/nvim/init.vim    # Neovimではなくvimの方はvim ~/.vimrc
```

以下を追加

```
set shell=/bin/zsh # コマンドの際にはzshを使う
```

```
set shiftwidth=4 # Indentの幅
```

```
set tabstop=4 # タブに変換されるサイズ
```

```
set expandtab # タブの入力の際にスペース
```

```
set textwidth=0 # ワードラッピングなし
```

```
set autoindent # 自動インデント :set pasteで解除可能
```

```
set hlsearch # Searchのハイライト
```






```
set clipboard=unnamed # クリップボードへの登録
```

```
syntax on # SyntaxをEnable
```

# Vim Plugin

# Vim Plugin managers

- Vim Plug
- Vundle
- Pathogen
- Dein
- Janus

THE BEST 1 OF 8 OPTIONS <span>WHY?</span>			
BEST PLUGIN MANAGERS FOR VIM		PRICE	LAST UPDATED
96	 vim-plug	-	Jun 11, 2021
67	 Vundle	-	May 18, 2021
--	 Pathogen	-	May 18, 2021
--	 Dein.vim	-	Jun 8, 2021
--	 Janus	-	May 5, 2021
<a href="#">SEE FULL LIST</a>			

<https://www.slant.co/topics/1224/~best-plugin-managers-for-vim>

# vim plug <https://github.com/junegunn/vim-plug>

Install:

```
$ sh -c 'curl -fLo "${XDG_DATA_HOME:-$HOME/.local/share}"/nvim/site/autoload/plug.vim \ --create-dirs  
https://raw.githubusercontent.com/junegunn/vim-plug/master/plug.vim'
```

Usage:

Vimのコマンドモード

:PlugStatus

:PlugInstall

:PlugUpdate

:PlugUpgrade

# Vim horizon <https://github.com/ntk148v/vim-horizon>

Install:

```
$ vim ~/.config/nvim/init.vim
```

```
call plug#begin()
```

```
Plug 'ntk148v/vim-horizon'
```

```
call plug#end()
```

```
: Pluginstall
```

```
$ vim ~/.config/nvim/init.vim
```

```
set termguicolors
```

```
set t_Co=256i
```

```
colorscheme horizon
```



# NERDTree <https://github.com/preservim/nerdtree>

Install:

```
$ vim ~/.config/nvim/init.vim
```

```
call plug#begin()
```

```
Plug 'preservim/nerdtree'
```

```
call plug#end()
```

```
:NERDTreeFocus
```

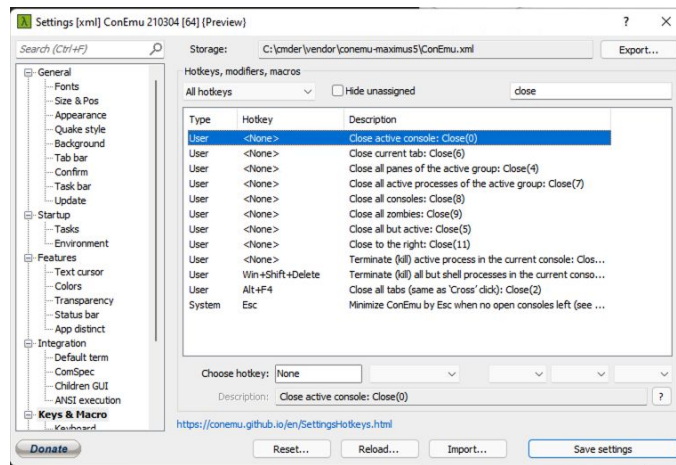
```
$ vim ~/.config/nvim/init.vim
```

# ファイル引数なしの時だけ起動がおすすめ

```
autocmd VimEnter * if argc() == 0 && !exists('s:std_in') && v:this_session == '' | NERDTree | endif
```

ctrl + ww (double w)

ctrl + w (h/j/k/l)



# fzf <https://github.com/junegunn/fzf>

brew install fzf or apt-get install fzf

```
$ git clone --depth 1 https://github.com/junegunn/fzf.git ~/.fzf && ~/.fzf/install
```

or

```
git clone --depth 1 https://github.com/junegunn/fzf.git ~/.fzf && ~/.fzf/install
```

```
$ vim ~/.config/nvim/init.vim
```

```
Plug 'junegunn/fzf', { 'do': { -> fzf#install() } }
```

```
Plug 'junegunn/fzf.vim'
```

If there is installation error:

```
Plug 'junegunn/fzf', { 'dir': '~/.fzf', 'do': './install --all' }
```

Usage:

control + j / k で移動

```
:FZF ../src
```

```
:Files
```

# Vim-fugitive <https://github.com/tpope/vim-fugitive>

Install:

```
$ mkdir -p ~/.config/nvim/pack/tpope/start
```

```
$ cd ~/.config/nvim/pack/tpope/start
```

```
$ git clone https://tpope.io/vim/fugitive.git
```

```
$ vim -u NONE -c "helptags fugitive/doc" -c q
```

Usage:

```
:Gblame (git blame)
```

# Vim-gitgutter <https://github.com/airblade/vim-gitgutter>

Install:

```
$ mkdir -p ~/.config/nvim/pack/airblade/start
```

```
$ cd ~/.config/nvim/pack/airblade/start
```

```
$ git clone https://github.com/airblade/vim-gitgutter.git
```

```
$ nvim -u NONE -c "helptags vim-gitgutter/doc" -c q
```

```
$ vim ~/.config/nvim/init.vim
```

```
let g:gitgutter_highlight_lines = 1
```

Usage:

```
:GitGutterToggle
```

# Vim-commentary <https://github.com/tpope/vim-commentary>

Install:

```
$ mkdir -p ~/.config/nvim/pack/tpope/start
```

```
$ cd ~/.config/nvim/pack/tpope/start
```

```
$ git clone https://tpope.io/vim/commentary.git
```

```
$ vim -u NONE -c "helptags commentary/doc" -c q
```

行を選択	v + ↓
コメント/アンコメント	gc

# vim-polyglot <https://github.com/sheerun/vim-polyglot>

Install:

```
$ pip3 install pylint (apt-get install python3 python3-pip)
```

```
export PATH=$HOME/.local/bin:$PATH
```

```
$ vim ~/.config/nvim/init.vim
```

```
set nocompatible
```

```
call plug#begin()
```

```
Plug 'sheerun/vim-polyglot'
```

```
call plug#end()
```

Usage:

```
$ vim /test.csv
```

# coc.nvim <https://github.com/neoclide/coc.nvim>

python: brew install python node curl (Mac)

nvm install node

source .nvm/nvm.sh

\$ apt-get install python3 python3-pip (Windows)

\$pip3 install pylint jedi

\$export PATH=\$HOME/.local/bin:\$PATH (Windows)

Install NVM and Node

sudo apt install curl

curl https://raw.githubusercontent.com/creationix/nvm/master/install.sh | bash

\$ sudo curl -sL install-node.now.sh/its | bash

\$ vim ~/.config/nvim/init.vim

Plug 'neoclide/coc.nvim', {'branch': 'release'}

:CocInstall coc-python

Usage:

vim /tmp/a.py

# Vim-go <https://github.com/fatih/vim-go>

Install go: <https://golang.org/doc/install>

Download and install (Mac)

sudo apt-get install golang-go (Windows)

```
$ vim ~/.config/nvim/init.vim
```

```
call plug#begin()
```

```
Plug 'fatih/vim-go', { 'do': ':GoUpdateBinaries' }
```

```
call plug#end()
```

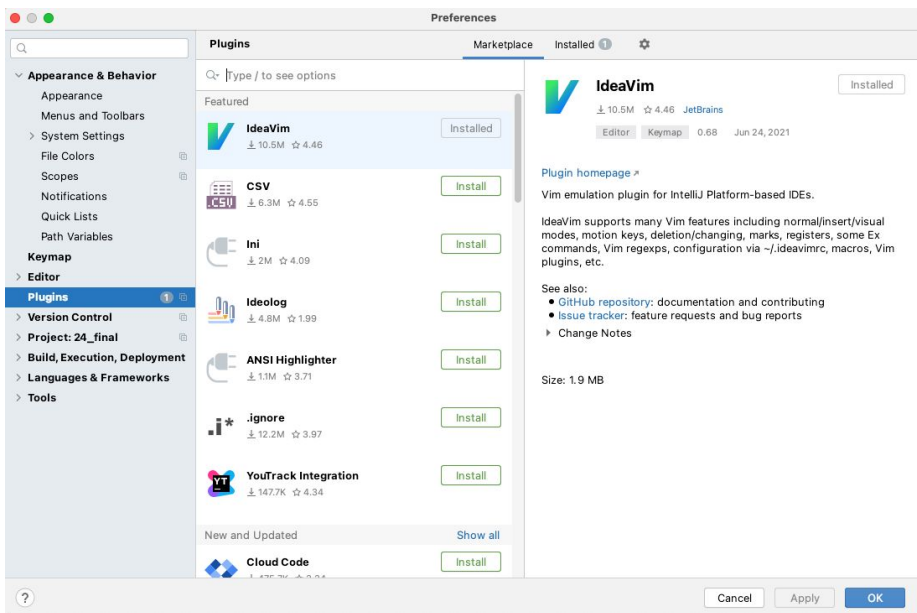
Usage:

```
$ vim /tm/a.go
```

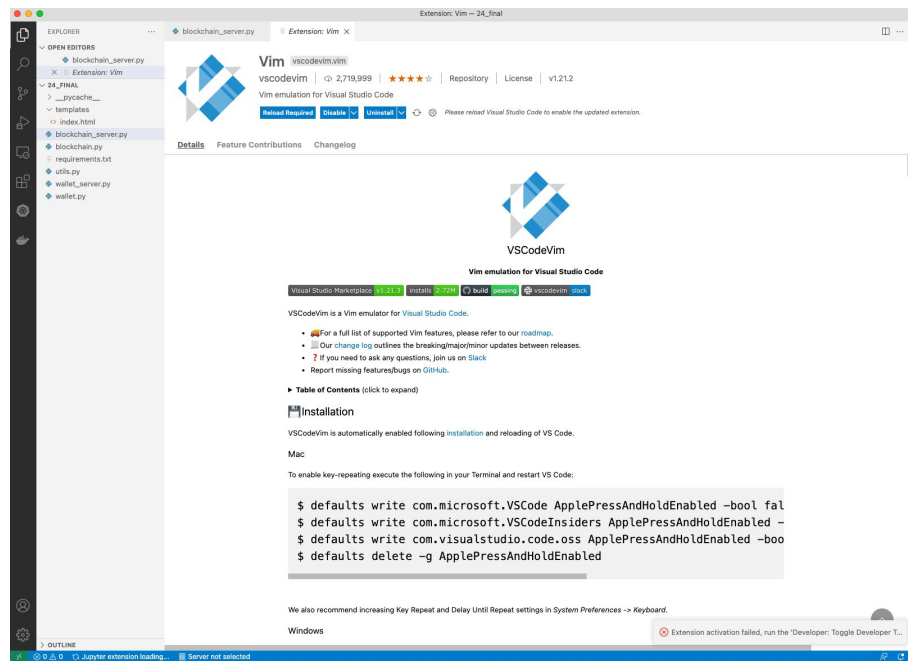
```
:GoRun :GoLint
```



# IDE Plugin for vim



Pycharm



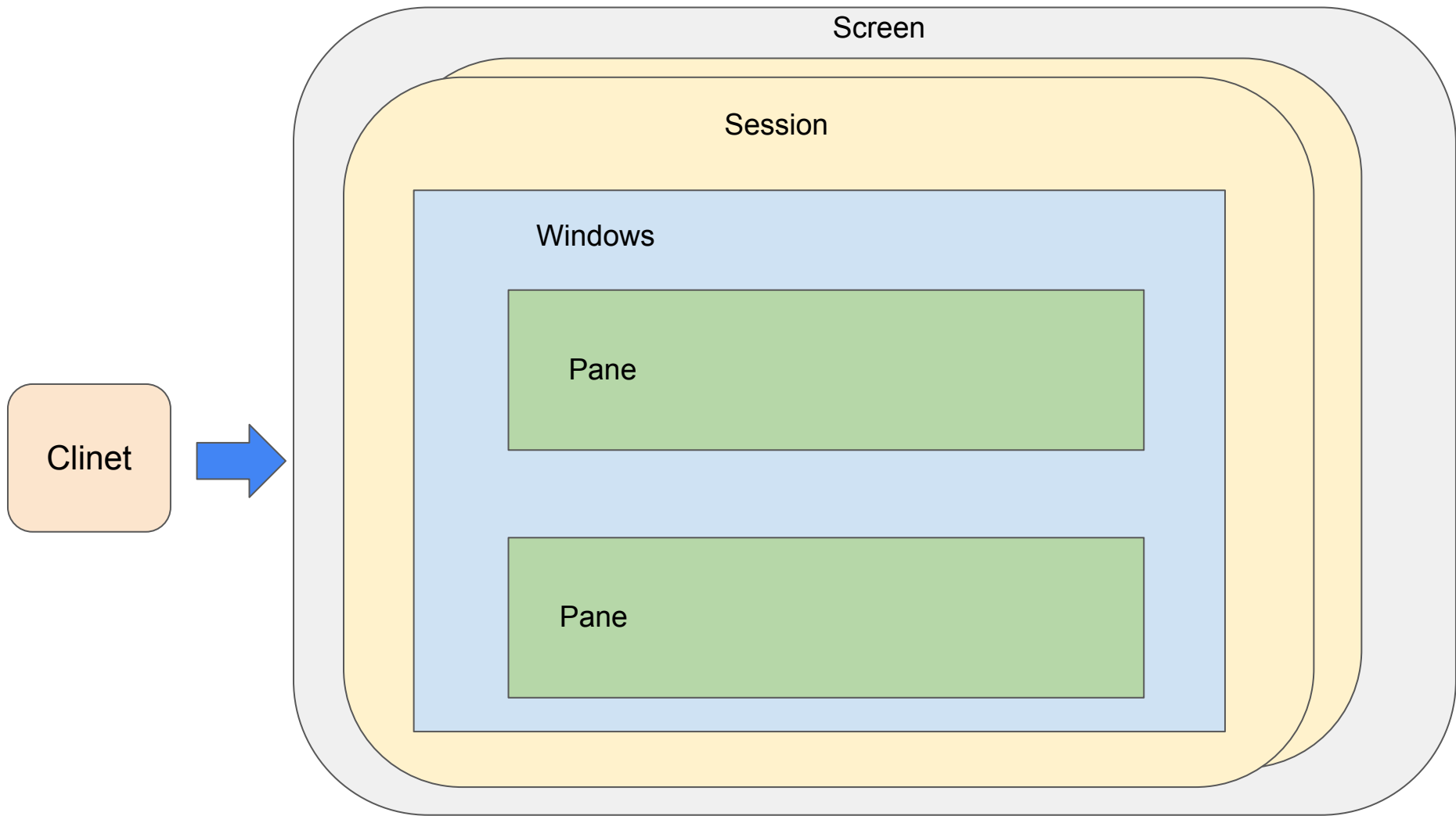
VS code

# Tmux

## (Terminal Multiplexer)

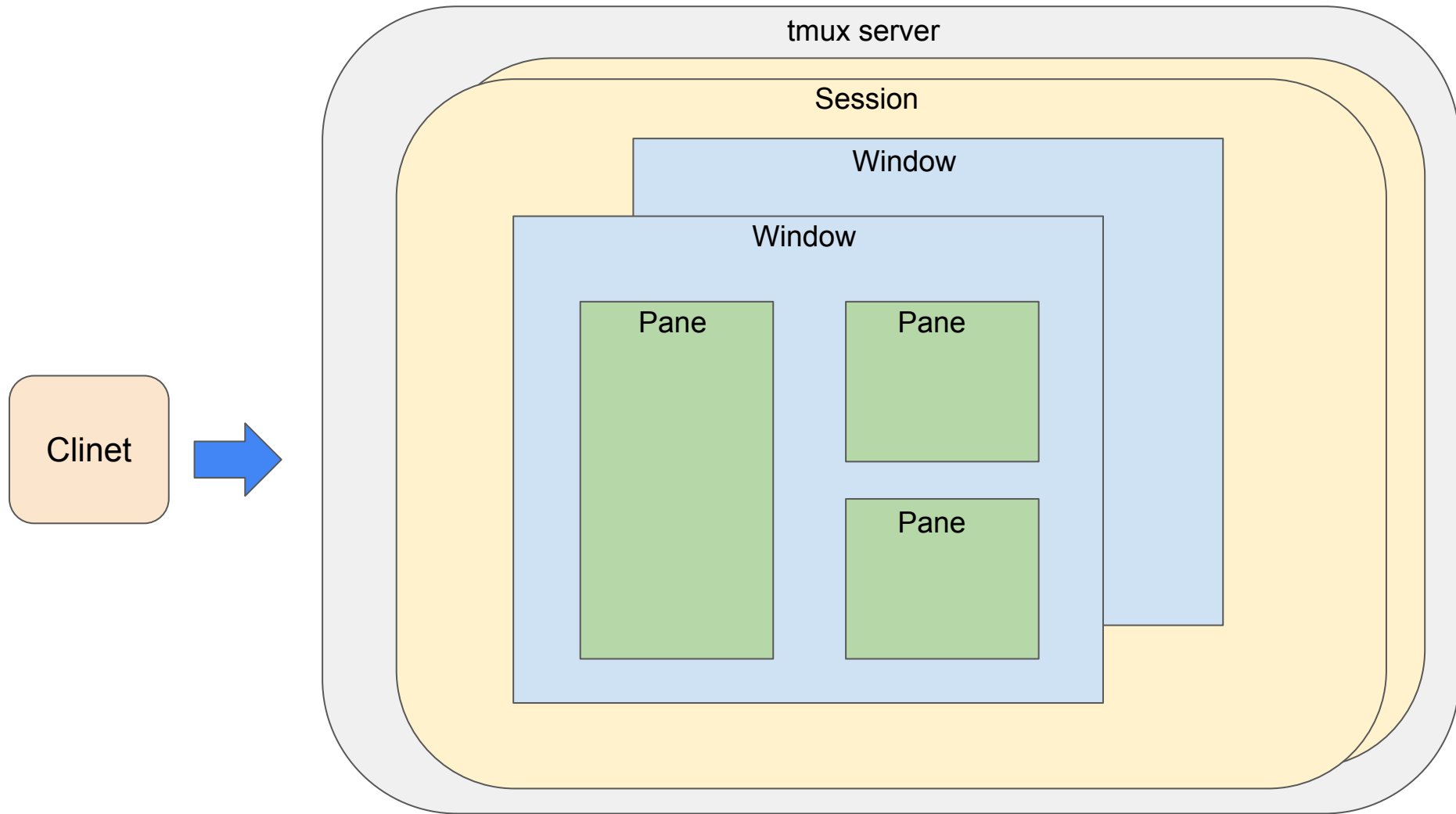
# Screen VS Tmux

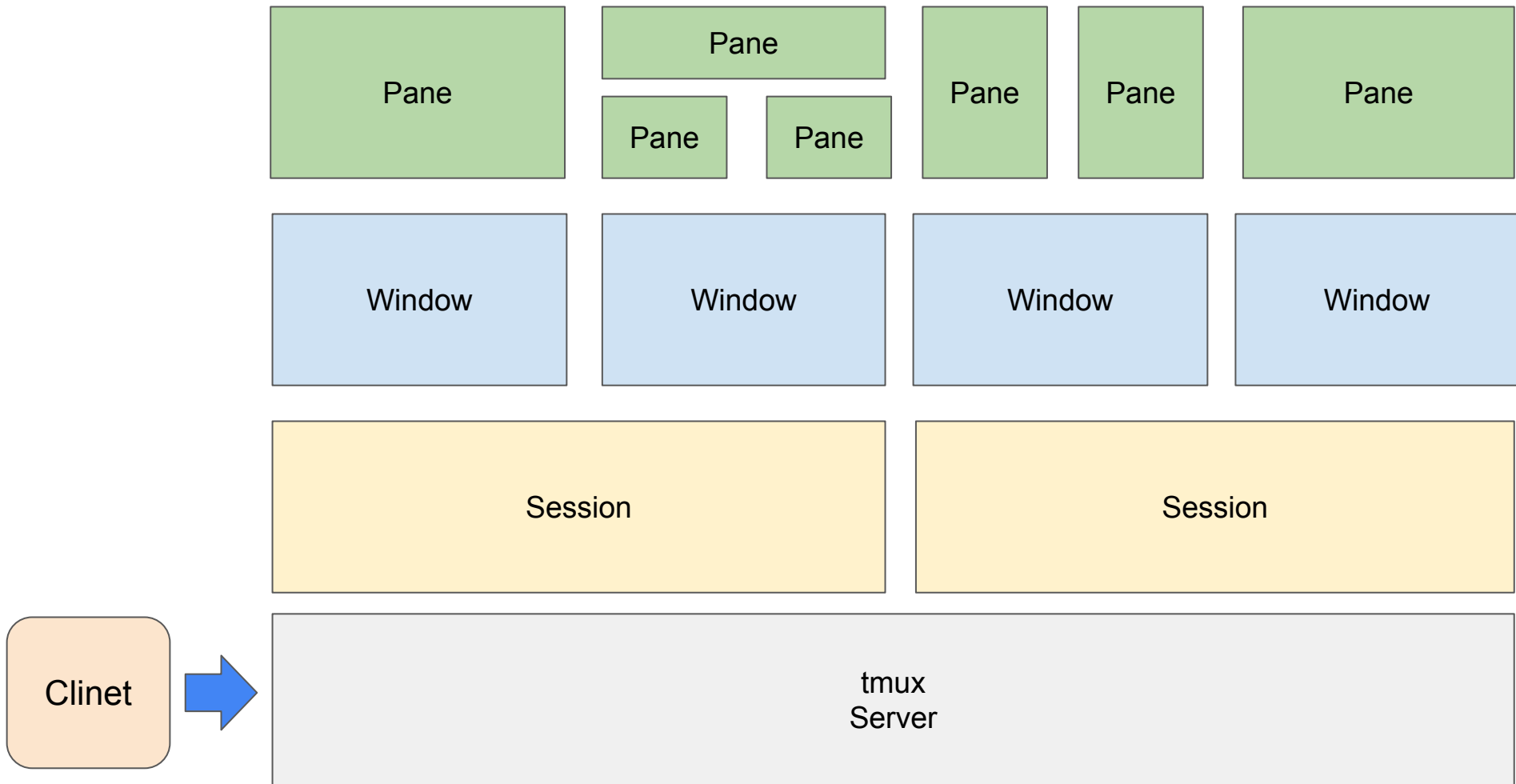
Screen	Tmux
GNU GPL	BSD License
簡単	フレンドリー
セッションハンドリングは普通	セッションハンドリングが優れている
普通程度に更新	頻繁に更新



# Screen

screen	起動
Ctrl + a S (shift + s)	横画面分割
Ctrl + a Tab	画面移動
Ctrl + a c	プロセス起動
Ctrl + a   (mac does not support)	縦画面分割
Ctrl + a Q (shift + q)	ペインを閉じる
Ctrl + a d	デタッチ
screen -ls	screen画面チェック
screen -r	アタッチ





# tmux session

tmux	起動
Control + b d	デタッチ
tmux ls	セッションの一覧
tmux a -t セッション名	アタッチ
Control + b s => 0 or 1	セッションの一覧から選択
tmux kill-server	tmuxのシャットダウン
Control + b t	時計を表示

※ Prefix のデフォルトは Control + bですがtmux.confで Control + gに変更します。



# Tmux config

デフォルトのPrefix Control + bを Control +g に変更

```
$ cat ~/.tmux.conf  
  
unbind C-b  
  
set -g prefix C-g  
  
bind C-g send-prefix
```

# tmux session

tmux	起動
Control + g d	デタッチ
tmux ls	セッションの一覧
tmux a -t セッション名	アタッチ
Control + g s => 0 or 1	セッションの一覧から選択
tmux kill-server	tmuxのシャットダウン
Control + g t	時計を表示

※ Prefix のデフォルトは Control + bですがtmux.confで Control + gに変更しています。

# tmux pane

Control + g %	横(左右)に分ける
Control + g “	縦(上下)に分ける
Control + g x or Control d (通常のterminalのexit)	ペイン破棄
Control + g z	ペイン拡大/縮小
Control + g o	次のペインに移動
Control + g ←→↑↓	ペインを移動
Control + g {	ペインの順序を前方向に入れ替え
Control + g }	ペインの順序を後方向に入れ替え
Control + g space	レイアウトの変更
Control g + q => 数字	ペインの番号の表示から移動
set -g display-panes-time 2000	表示時間の設定

※ Prefix のデフォルトは Control + bですがtmux.confで Control + gに変更しています。

# tmux pane

Control + g \ ( があるキー)	横(左右)に分ける
Control + g -	縦(上下)に分ける
Control + g x or Control d (通常のterminalのexit)	ペイン破棄
Control + g z	ペイン拡大/縮小
Control + g o	次のペインに移動
Control + g ←→↑↓	ペインを移動
Control + g {	ペインの順序を前方向に入れ替え
Control + g }	ペインの順序を後方向に入れ替え
Control + g space	レイアウトの変更
Control g + q => 数字	ペインの番号の表示から移動
set -g display-panes-time 2000	表示時間の設定

※ Prefix のデフォルトは Control + bですがtmux.confで Control + gに変更しています。

# Tmux config

ペインの表示時間を2秒にする

```
$ cat ~/.tmux.conf
```

```
# 以下を追加
```

```
set -g display-panes-time 2000
```

# Tmux config

デフォルトのペインの分割を変更する。

```
$ cat ~/.tmux.conf
```

```
# 以下を追加
```

```
bind \ split-window -h
```

```
bind - split-window -v
```

# tmux windows

Control + g c	新規ウィンドウ
Control + g n	次のウィンドウ
Control + g p	前のウィンドウ
Control + g 数字	指定番号のウィンドウに移動
Control + g &	ウィンドウを破棄する
Control + g ,	ウィンドウの名前を変える

※ Prefix のデフォルトは Control + bですがtmux.confで Control + gに変更しています。

# tmux copy

Control + g [ -> 移動(jklh) -> v -> 移動(jklh) -> y -> enter -> Command + v (Mac)	コピー&ペースト
Control + g [ -> 移動(jklh) -> v -> 移動(jklh) -> y -> enter -> Control v (Windows)	コピー&ペースト

※ Prefix のデフォルトは Control + bですがtmux.confで Control + gに変更しています。



# Tmux config

クリップボードへのコピーを Vim の機能で使えるようにする。

```
$ cat ~/.tmux.conf
```

```
# 以下を追加
```

```
setw -g mode-keys vi
```

```
bind-key -T copy-mode-vi 'v' send -X begin-selection
```

```
bind-key -T copy-mode-vi 'C-v' send -X rectangle-toggle
```

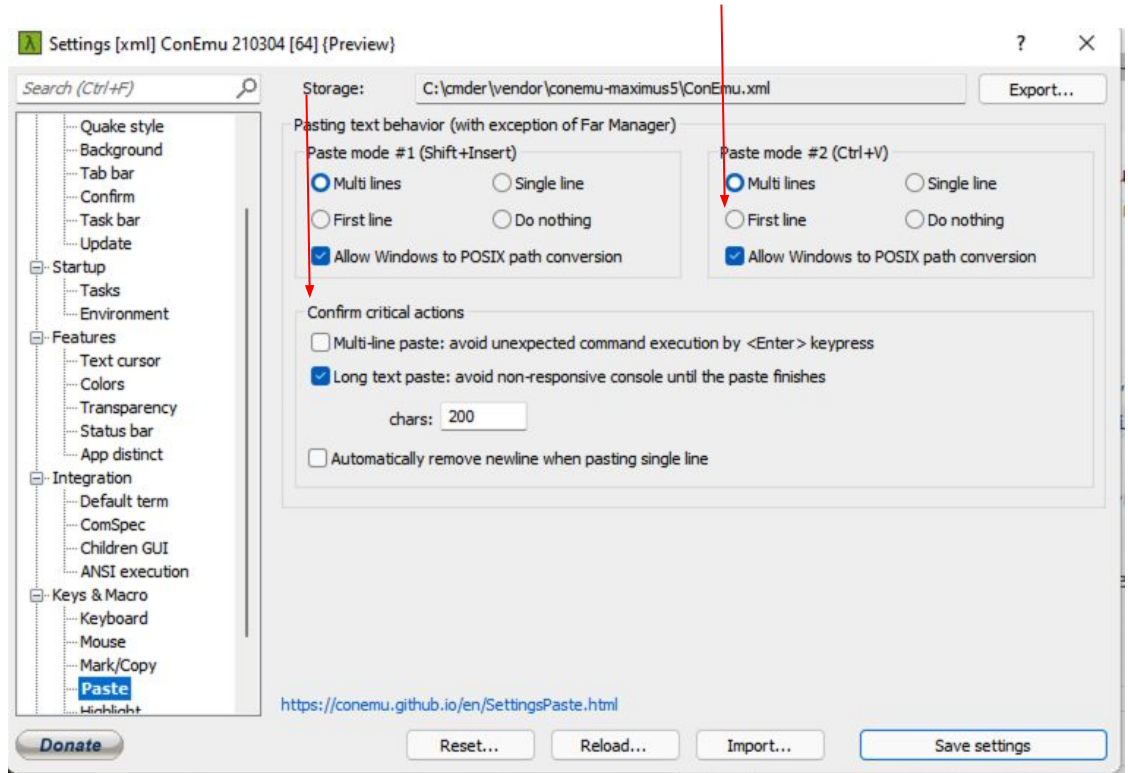
```
bind-key -T copy-mode-vi 'y' send -X copy-selection
```

```
# WSL の場合は
```

```
bind-key -T copy-mode-vi 'y' send-keys -X copy-pipe-and-cancel clip.exe
```

```
vim --version | grep clipboard
```

```
sudo apt-get install vim-gtk
```



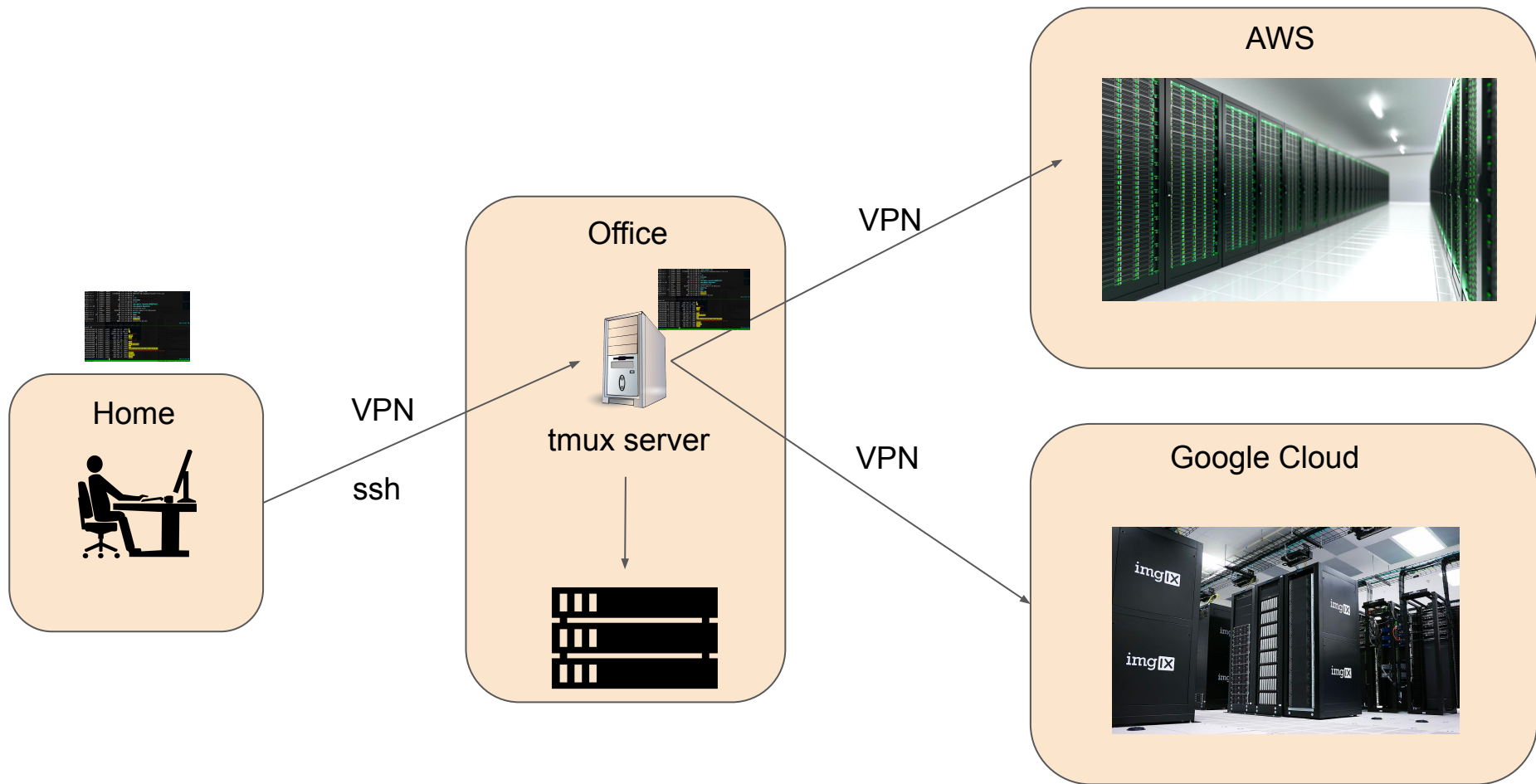
# Tmux config

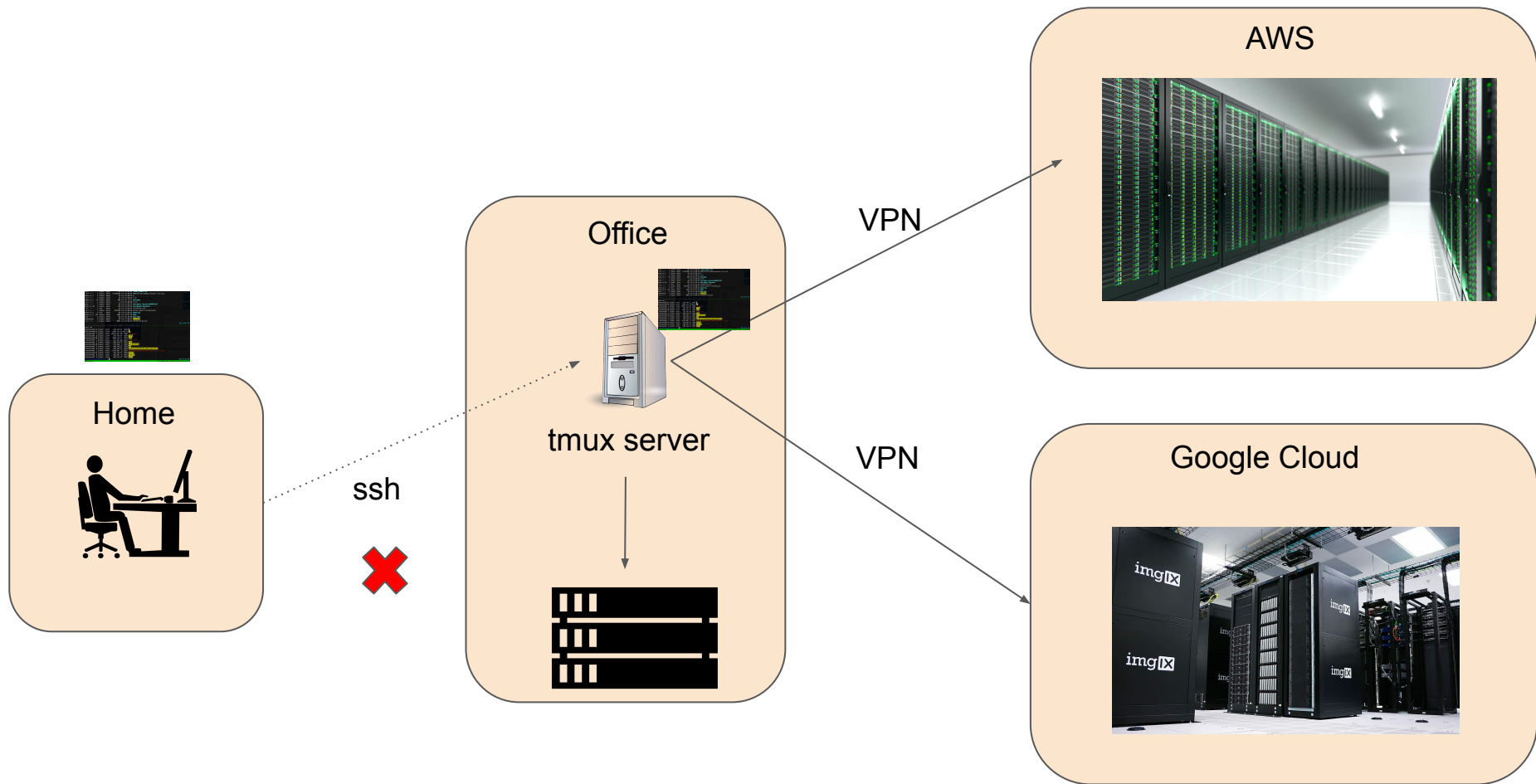
TmuxがEscape街をするので、NeovimのEscapeが遅いためescape timeは0に設定する

```
$ cat ~/.tmux.conf
```

以下を追加

```
set -s escape-time 0
```





Zsh

# zshのコマンド操作

! <code>&lt;command&gt;</code> ex) !source	直近のコマンド実行
!! (vimの再編集などによく使う)	直前のコマンドを実行
control + r	Historyのコマンド検索
control + a	カーソルを先頭
control + k	カーソルから行末まで削除
control + e	カーソルを行末へ
control + b	カーソルを1文字戻る
control + f	カーソルを1文字進める
control + d	カーソル位置の文字を削除

# zshの設定 <https://zsh.sourceforge.io/Doc/Release/Options.html>

```
$vim ~/.zshrc
```

setopt no_beep	音を鳴らさない
setopt auto_pushd	ディレクトリの移動に pushd
setopt pushd_ignore_dups	重複は削除
setopt auto_cd	一致するディレクトリに cdなしで移動できる
setopt hist_ignore_dups	直前と同じコマンドは履歴に追加しない
setopt share_history	他のzshで履歴を共有する
setopt inc_append_history	即座に履歴を保存する



# History 設定

```
vim ~/.zshrc  
  
export HISTFILE=~/.zsh_history  
  
export HISTSIZE=100000  
  
export SAVEHIST=100000
```

Oh My Zsh

# Oh my Zsh <https://ohmyz.sh>

Install:

```
sh -c "$(curl -fsSL https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"
```

Plugins: <https://github.com/ohmyzsh/ohmyzsh/wiki/Plugins>

```
alias zshconfig="vim ~/.zshrc"
```

```
alias ohmyzsh="vim ~/.oh-my-zsh"
```

```
cat ~/.zshrc.pre-oh-my-zsh >> ~/.zshrc
```

GitがデフォルトでEnableになっておりBranch情報が消えるので以下の設定をしておくといい

```
git branch
```

```
$vim ~/.gitconfig
```

```
[pager]
```

```
branch = false
```

# Z

Install:

```
$ vim ~/.zshrc
```

```
plugins=(... z)
```

Usage:

```
$ z plug
```

# autojump <https://github.com/wting/autojump>

Install:

```
$sudo apt-get install python-is-python3 (Windows)
```

```
$ git clone git://github.com/wting/autojump.git
```

```
$ cd autojump
```

```
$ ./install.py or ./uninstall.py
```

```
$ vim ~/.zshrc
```

```
plugins=(... autojump)
```

Usage:

```
cd XXX
```

```
j XXX
```

```
j -s
```

# web-search

Install:

```
$ vim ~/.zshrc  
plugins=(... web-search)
```

Usage:

```
google <search word>
```

# python

Install:

```
$ vim ~/.zshrc
```

```
plugins=(... python)
```

Usage:

```
$ pyclean
```

# fzf

Install:

```
$ brew install fzf
```

```
$ vim ~/.zshrc  
plugins=(... fzf)
```

Usage:

Control + T or fzf (cmdrはショートカットキーを変更する必要)

Control + R => j or k



# zsh-syntax-highlighting <https://github.com/zsh-users/zsh-syntax-highlighting>

Install:

```
$ git clone https://github.com/zsh-users/zsh-syntax-highlighting.git  
${ZSH_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-syntax-highlighting
```

```
$ vim ~/.zshrc  
plugins=( ... zsh-syntax-highlighting)
```

Usage:

```
$ echo 'hello'
```

# zsh-autosuggestions <https://github.com/zsh-users/zsh-autosuggestions>

Install:

```
$ git clone https://github.com/zsh-users/zsh-autosuggestions
```

```
${ZSH_CUSTOM:-~/.oh-my-zsh/custom}/plugins/zsh-autosuggestions
```

```
$ vim ~/.zshrc
```

```
plugins=( ... zsh-autosuggestions)
```

# Shpotify <https://github.com/hnarayanan/shpotify>

Install app : <https://www.spotify.com/us/download/mac/> (Mac)

brew install shpotify

<https://www.spotify.com/us/download/linux/> (Windows)

```
$ vim ~/.zshrc  
plugins=(... shpotify)
```

Usage:

<https://github.com/hnarayanan/shpotify>

# powerlevel10k <https://github.com/romkatv/powerlevel10k>

Install:

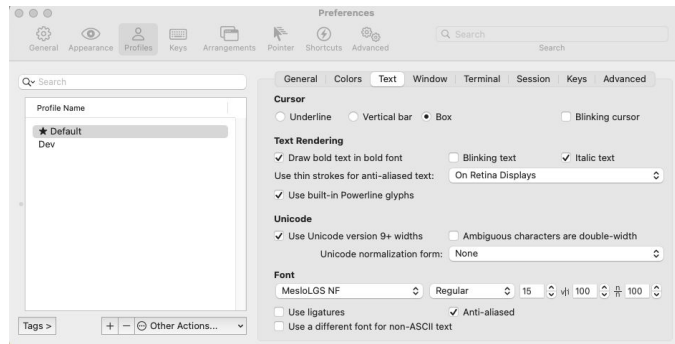
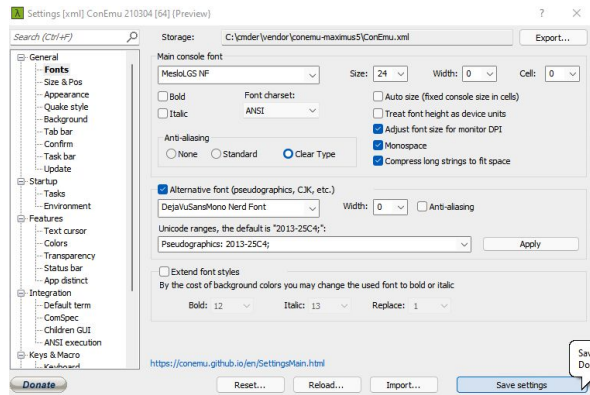
```
git clone --depth=1 https://github.com/romkatv/powerlevel10k.git
${ZSH_CUSTOM:-$HOME/.oh-my-zsh/custom}/themes/powerlevel10k
```

Install MesloLGS NF font and setup in iterm2

```
$ vim ~/.zshrc
Set ZSH_THEME="powerlevel10k/powerlevel10k" in ~/.zshrc.
```

```
$ p10k configure
```

```
$ source ~/.p10k.zsh
```



## その他

- <https://github.com/andreafrancia/trash-cli>
- <https://github.com/ohmyzsh/ohmyzsh/tree/master/plugins/history>
- <https://github.com/ohmyzsh/ohmyzsh/tree/master/plugins/extract>
- <https://github.com/ohmyzsh/ohmyzsh/tree/master/plugins/osx>
- <https://github.com/ohmyzsh/ohmyzsh/tree/master/plugins/kubectl>

# Neofetch <https://github.com/dylanaraps/neofetch>

Install:

brew install neofetch (Mac)

sudo apt-get install neofetch (WSL)

Usage:

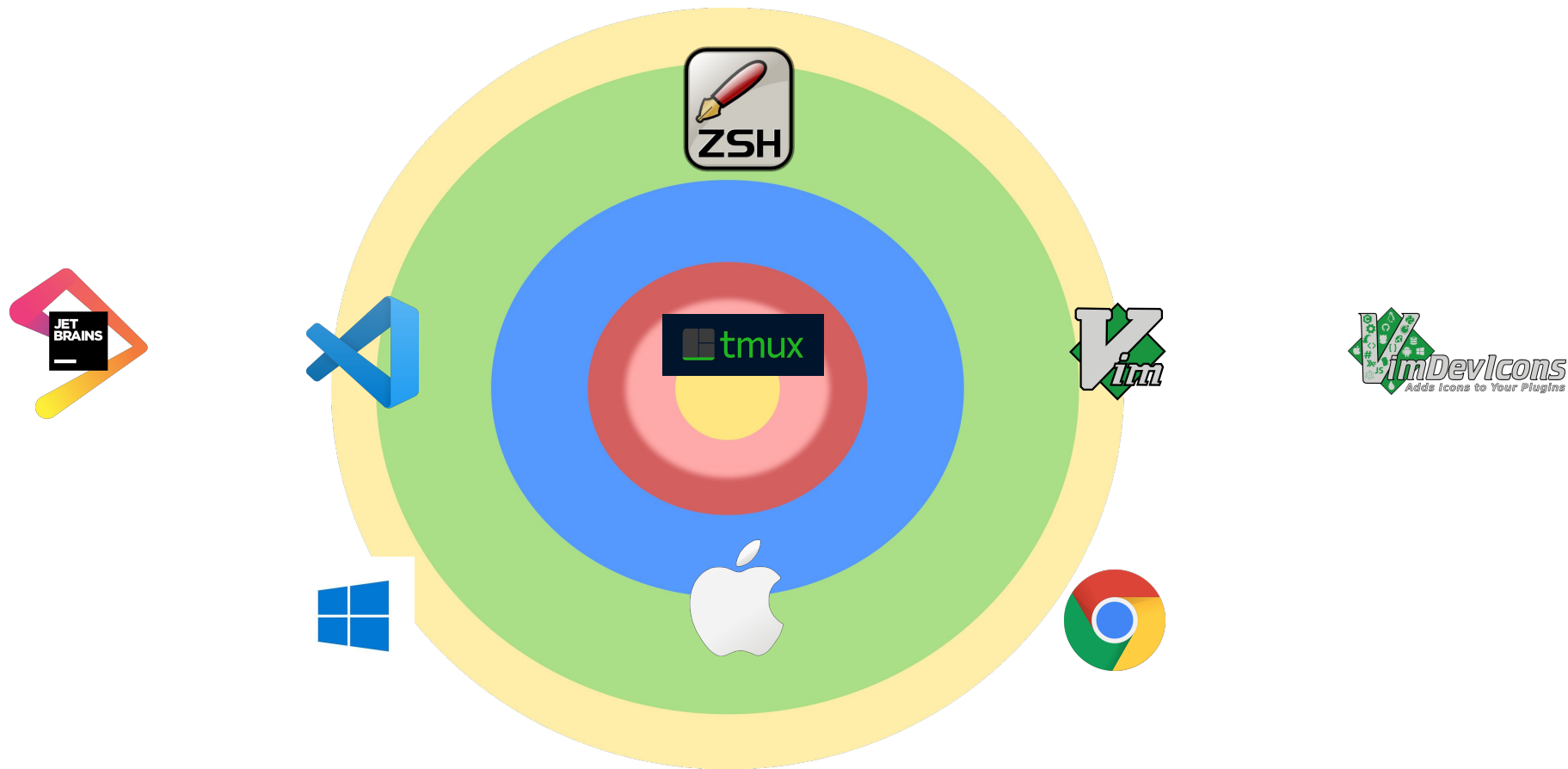
\$neofetch

Config:

vim ~/.config/neofetch/config.conf

便利なツール

全ての便利コマンドは覚えきれないので、各アプリの良い部分だけ使い覚えていく





# よく使うコマンド

<code>pushd popd =&gt; auto_pushd</code>	ファイルのスタック移動
<code>find . -name models.py =&gt; fzf</code>	ファイル検索
<code>find . -name '*.pyc'   xargs echo</code> <code>find . -name '*.pyc'   xargs rm =&gt; pyclean</code>	ファイル削除
<code>find . -name '*.py'   cut -d '/' -f5   cut -d '.' -f1</code>	ファイル名の抽出
<code>grep "test" -nr .</code>	ファイルの中身の検索
<code>ps auwx   grep a.py   grep -v grep   awk '{print \$2}'   xargs kill -9</code>	プロセスのkills
<code>du   sort -nr   head -n 5</code>	ソートからヘッド
<code>ls -al   wc -l</code>	行のカウント

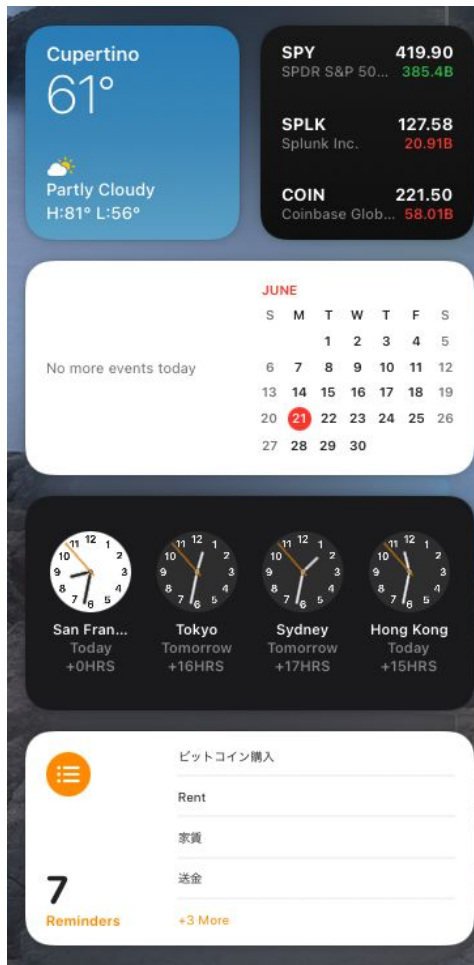
# Macショートカット

command + space	Spotlight Search
command + tab	起動しているアプリの切り替え
control + ↓ / control + ↑	Windowsの切り替え
Fn + F11	デスクトップの表示 / 非表示
option + command + space	Finderと検索を表示

# その他便利なMACアプリ

- Rectangle
- Authy
- Disk Inventory X
- MacVim
- Lastpass or 1Password
- Stickers
- Chrome Remote Desktop, Google Translate, Google Drive
- As a timer or Horo
- Skitch
- ToDo
- Aerial (目標のものか自然なもの、遊び系はあまり良くない )
- 壁紙(自然系か IDEの邪魔にならない色)他の PCと区別がつきやすいもの。

# Widget (Mac)



# Windowsショートカット

Win + l	パソコンをロックする
Control + Alt + Del	シャットダウンする
Win + e	エクスプローラーを起動させる
Win + x	クイックメニュー
Win + d	デスクトップを表示させる
Alt + Tab	画面を切り替え
Win + Tab	タスクビューを開く
Win + v	クリップボードの履歴を表示する
Win + i	設定を開く
Win + r	ファイル名を指定して実行させる

# その他便利なWindowsアプリ

- DIVVY (Windows Management)
  - <https://mizage.com/windivvy/>
- gVim
- Microsoft Sticky Notes
- CrystalDiskInfo
- CCleaner
- 8GadgetPack (Windows 11からはガジェット機能あり)
- Chrome Remote Desktop, Google Translate, Google Drive
- Authy
- 1Password or Lastpass
- Aerial Screen Saver(目標のものか自然なもの、遊び系はあまり良くない)
- 壁紙(自然系かIDEの邪魔にならない色)他のPCと区別がつきやすいもの。

# Chrome Browser ショートカット

## Mac

command + T	新しいタブの作成
command + w	タブの削除
command + R	リロード
command + 数字	タブの移動
command + / -	拡大縮小

## Windows

Control + T	新しいタブの作成
Control + w	タブの削除
Control + R	リロード
Control + 数字	タブの移動
Control + / -	拡大縮小

# Chrome Browser Extensions

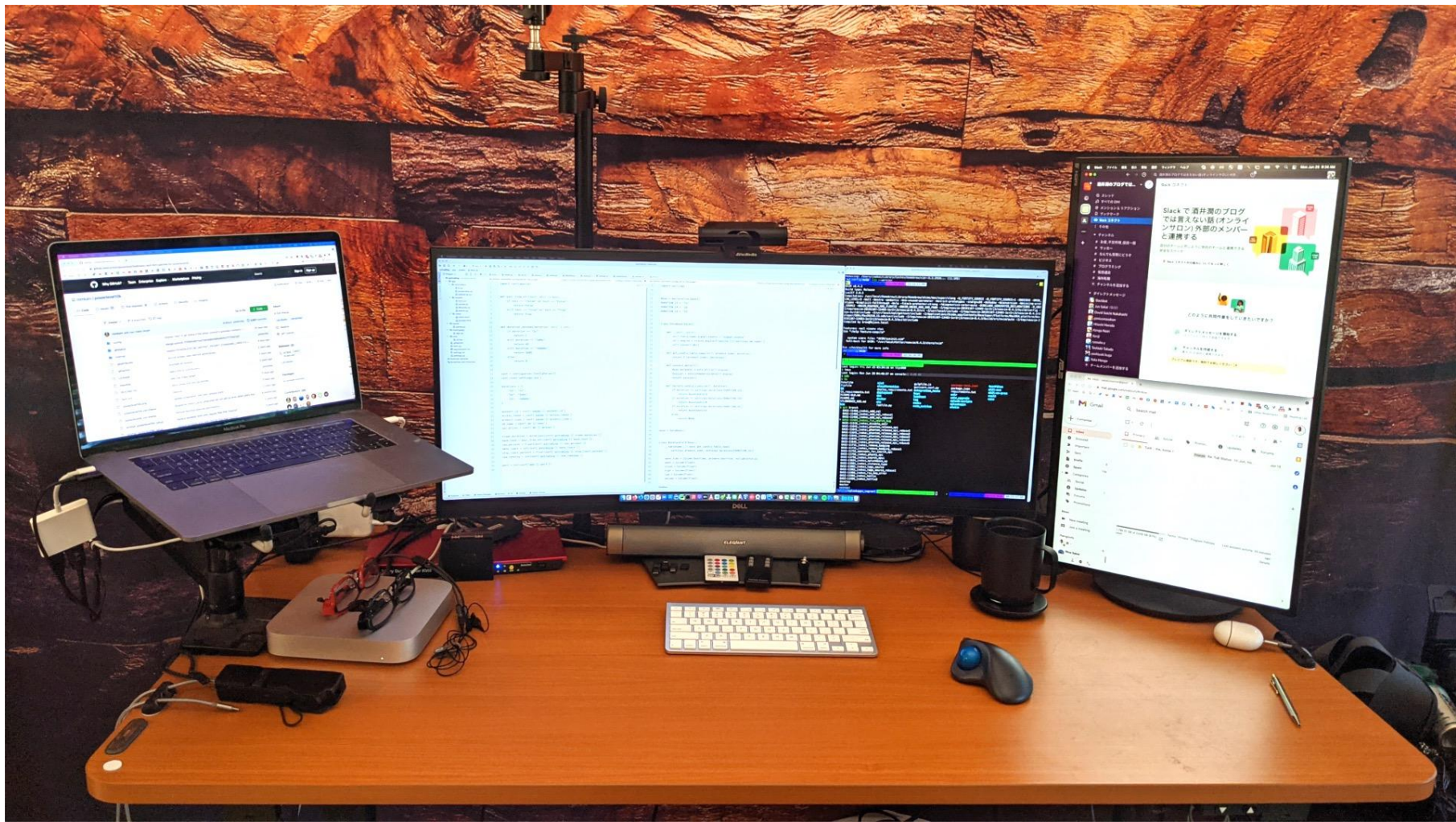
- Chrome Remote desktop
- JSON Formatter
- Google Translate
- Autopagerize for Chrome
- One Tab
- Tab Resize
- Advanced Rest API -> Postman



# Slack

- 自分のスレッドにメモ
- Plugin
  - zoom
  - Google Calendar
  - Forecast
- 作業の邪魔をするようであればDisableにする
  - Gmail
  - Twitter
  - RSS (News)
- 画像共有

仕事環境



# Desktop Environment

