

# Boost your vimrc with some template techniques!

---

- 2023-11-18
- aiya000 ([@public\\_ai000ya](https://public.ai000ya))



← This session's slide

# **What is this session?**

---

# What is this session?

---

- Learn how to **refine** your vimrc
- using some **techniques**

# Me

---

# Me

---

- Name
  - aiya000
- Twitter
  - [@public\\_ai000ya](https://twitter.com/public_ai000ya)
- High tone Peter Pan 😅



# Me

---

- Name
  - aiya000
- Like
  - Strong static typed languages
    - **TypeScript, Haskell, Idris, Scala3**
  - Math
    - Category theory, Algebraic structure  
↑ a little bit :)



# Me

---

- I also love in **Vim** ;)

# Me

---

## My works!



# Me

---

## Latest works!



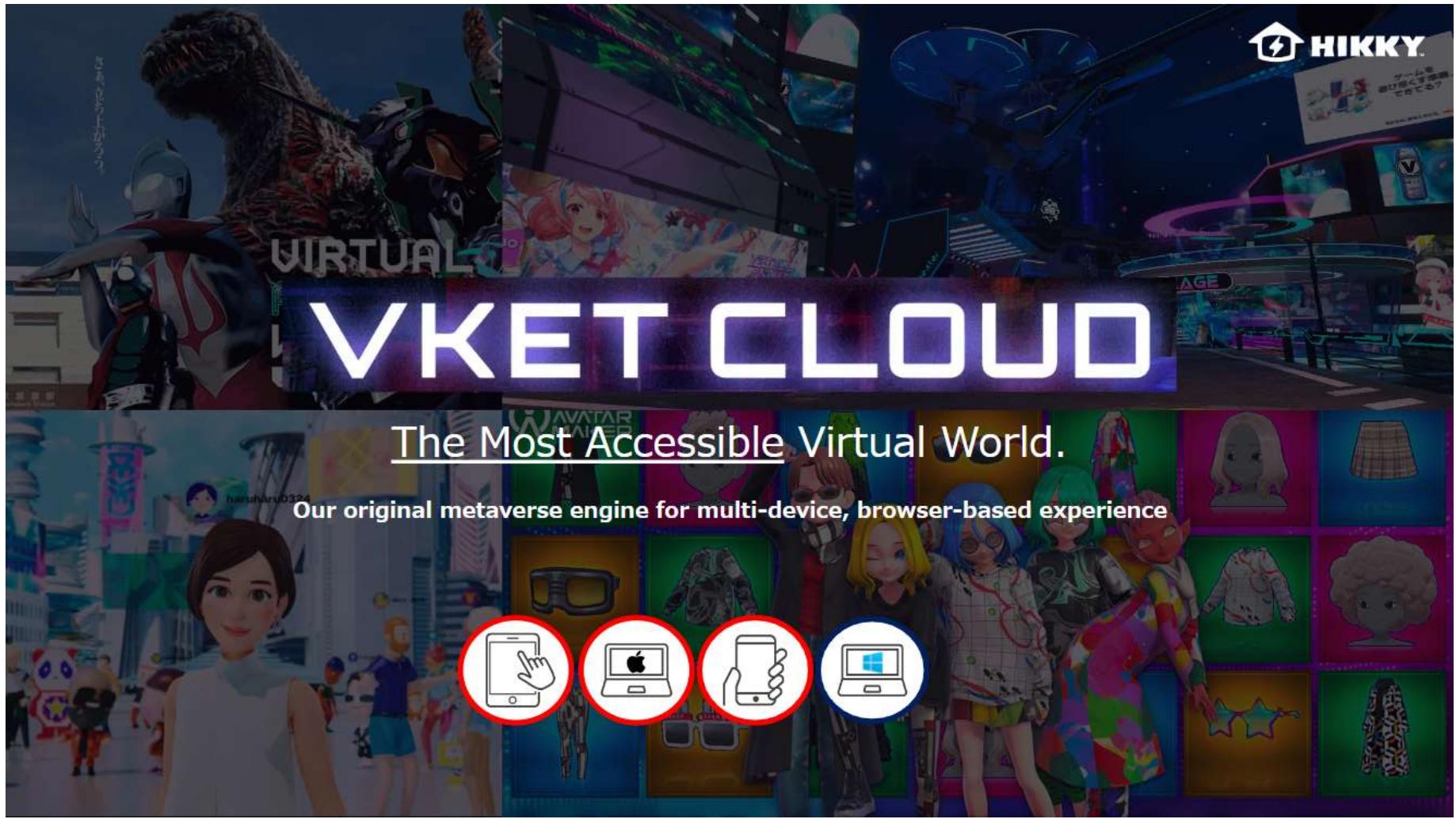
nice



---

# HIKKY, Inc.

---



# HIKKY, Inc.

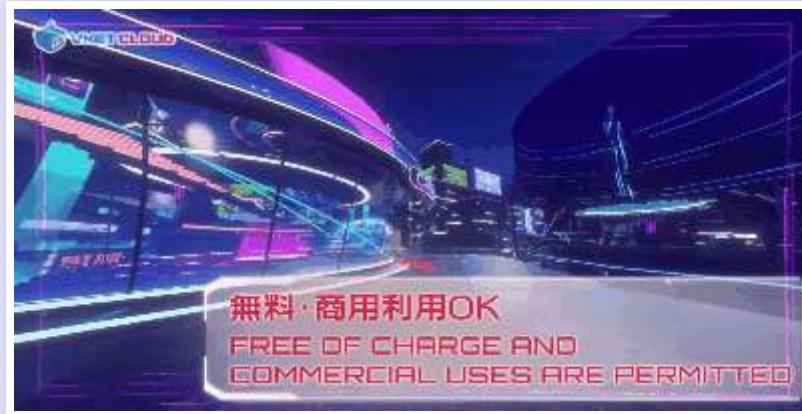
---

A company which is holding **Vket** on VRChat.

# HIKKY, Inc.

---

## Vket Cloud



- A **metaverse** development platform
- For PC, smartphones, tablet devices
- **Free** (for persons)

Try now: [My Vket](#)

**Boost your vimrc with  
some template techniques!**

---

# **autoload, plugin, vimrc**

---

# autoload, plugin, vimrc

---

In vimrc,  
function and command declarations  
is placing num of lines.

```
function s:read_git_root() abort
    " ...
endfunction
function s:job_start_simply(cmd) abort
    " ...
endfunction
" ... and a lot of functions and sub functions.

command! -bar GitPushAsync call s:job_start_simply(['git', 'pu
command! -bar GitAddAllAsync call s:job_start_simply(['git', 'a
" ... and a lot of commands.

let s:root = call s:read_git_root()
" ... others
```

## autoload, plugin, vimrc

---

You can use **~/.vim/autoload** and **~/.vim/plugin** directory.

```
$HOME  
| - .vim  
|   | - autoload  
|   | - plugin
```

# autoload, plugin, vimrc

---

## autoload

---

### .vim/autoload/vimrc.vim

```
function vimrc#read_git_root() abort
    " ...
endfunction

function s:foo() abort
    " a sub function (not be exposed)
endfunction

" ...
```

# (Sub namespaces `foo#bar#baz()`)

---

## `.vim/autoload/vimrc/job.vim`

```
function vimrc#job#start_simply(cmd) abort
    " ...
endfunction

function s:bar() abort
    " a sub function (not be exposed)
endfunction

" ...
```

# plugins

# .vim/plugin/vimrc.vim

```
command! -bar GitPushAsync call s:job_start_simply(['git', 'pu
command! -bar GitAddAllAsync
    \ call s:job_start_simply(['git', 'add', '-A'])
command! -bar -nargs=1 GitCommitMAsync
    \ call s:job_start_simply(['git', 'commit', '-m', <q-args>
command! -bar -nargs=1 GitCheckoutAsync
    \ call s:job_start_simply(['git', 'checkout', <q-args>])
"
    ...
```

🙌 Easy to use 🙌

---

## .vimrc

```
let s:root = vimrc#read_git_root()  
" ...
```

## On your Vim

```
:GitCommitMAsync awesome  
:GitPushAsync
```

## autoload, plugin, vimrc

- autoload: **functions**
- plugin: **commands**
- vimrc: settings and others

nice



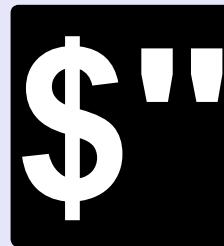
---

## **autoload, plugin, vimrc**

---

- doc, indent, syntax, ftdetect, ftplugin, ...

# String interpolation



---

# String interpolation \$" \$"

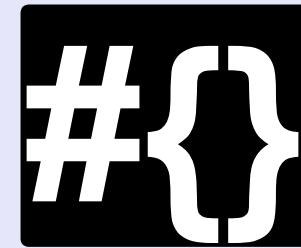
---

```
" No more '...' !!!!!!!  
  
" Not easy to read  
call system('chown -R ' .. $USER .. ':' .. $GROUP .. '"{foo_di  
  
" ↓ Easy to read ↓  
call system($'chown -R "{$USER}:{$GROUP}" "{foo_directory}"')
```

```
" No more expand('~') !  
  
if filereadable('${$HOME}/dein_env.toml')  
    call dein#load_toml('~/dein_env.toml', {'lazy': 0})  
endif
```

```
" Better than printf()  
let name = 'Vim'  
  
" Not easy to read  
echo printf('Hi %s', name)  
  
" ↓ Easy to read ↓  
echo $'Hi {name}'
```

# Literal Dict



---

# Literal Dict #{}

---

## Dict {}

```
call ddc#custom#patch_global({  
    \ 'ui': 'native',  
    \ 'sources': ['vim-lsp', 'around', 'neosnippet', 'file', '  
    \ 'sourceOptions': {  
        \ '_': {  
            \ 'matchers': ['matcher_fuzzy'],  
            \ 'sorters': ['sorter_fuzzy'],  
            \ 'converters': ['converter_fuzzy'],  
            \ 'ignoreCase': v:true,  
        \ },  
        \ 'vim-lsp': #{  
    " ...
```

# highlighter to be **Karoshi**

```
call ddc#custom#patch_global({  
    \ 'ui': 'native',  
    \ 'sources': ['vim-lsp', 'around', 'neosnippet', 'file', 'buffer'],  
    \ 'sourceOptions': {  
        \ '_': {  
            \ 'matchers': ['matcher_fuzzy'],  
            \ 'sorters': ['sorter_fuzzy'],  
            \ 'converters': ['converter_fuzzy'],  
            \ 'ignoreCase': v:true,  
        },  
        \ 'vim-lsp': #{}  
    },  
}
```

hard to write

# Literal Dict #{}

---

```
call ddc#custom#patch_global(#{
    \ ui: 'native',
    \ sources: ['vim-lsp', 'around', 'neosnippet', 'file', 'bu
    \ sourceOptions: #{
        \ _: #{(
            \ matchers: ['matcher_fuzzy'],
            \ sorters: ['sorter_fuzzy'],
            \ converters: ['converter_fuzzy'],
            \ ignoreCase: v:true,
        \ },
        \ vim-lsp: #{(
    " ...
```

# Good highlighting!

```
call ddc#custom#patch_global({<
    \ ui: 'native', <
    \ sources: ['vim-lsp', 'around', 'neosnippet', 'file', 'buffer'], <
    \ sourceOptions: #{<
        \ _: #{<
            \ matchers: ['matcher_fuzzy'], <
            \ sorters: ['sorter_fuzzy'], <
            \ converters: ['converter_fuzzy'], <
            \ ignoreCase: v:true, <
        }, <
        \ vim-lsp: #{<
```

nice



---

method **x->Foo(y, z)**

---

## method **x->Foo(y, z)**

---

```
function Sum(num, x, y) abort
    return a:num + a:x + a:y
endfunction
```

```
echo Sum(10, 20, 30)
```

```
" ↑ Same ↓
```

```
echo 10->Sum(20, 30)
```

## method **x->Foo(y, z)**

---

```
" `x` is 1st argument
" `y` and `z` is rest arguments
x->Foo(y, z)

" ↑ Same ↓
```

```
FOO(x, Y, z)
```

**method x->Foo(y, z)**

---

Where is the method notation useful?

## method **x->Foo(y, z)**

---

Easy to read.

Can read from above to below.

```
" Fisrt foo(), next bar(), and then baz()
echo self->foo()
    \ ->bar(x)
    \ ->baz(y)

" If don't use the method notation
echo baz(
    \ bar(
        \ foo(self)
    \ )
\ )
```

# Vim script libraries

---

# vital.vim

---

# vital.vim

---

Vim script's **semi standard** library,  
from vim-jp.

<https://github.com/vim-jp/vital.vim>

```
let s>List = vital#vimrc#import('Data.List')
let s(Msg = vital#vimrc#import('Vim.Message')
let s:Promise = vital#vimrc#import('Async.Promise')
```

# vital.vim

---

## What vital.vim provides

Module	Description
<a href="#">Assertion</a>	assertion library
<a href="#">Async.Promise</a>	An asynchronous operation like ES6 Promise
<a href="#">Bitwise</a>	bitwise operators
<a href="#">Color</a>	color conversion library between RGB/HSL/terminal code
<a href="#">ConcurrentProcess</a>	manages processes concurrently with vimproc
<a href="#">Data.Base64</a>	base64 utilities library
<a href="#">Data.Base64.RFC4648</a>	base64 RFC4648 utilities library
<a href="#">Data.Base64.URLSafe</a>	base64 URLSafe utilities library
<a href="#">Data.Base32</a>	base32 utilities library
<a href="#">Data.Base32.Crockford</a>	base32 Crockford utilities library
<a href="#">Data.Base32.Hex</a>	base32 Hex utilities library
<a href="#">Data.Base32.RFC4648</a>	base32 RFC4648 utilities library
<a href="#">Data.Base16</a>	base16 utilities library
<a href="#">Data.BigNum</a>	multi precision integer library
<a href="#">Data.Closure</a>	Provide Closure object
<a href="#">Data.Counter</a>	Counter library to support convenient tallies
<a href="#">Data.Dict</a>	dictionary utilities library
<a href="#">Data.Either</a>	either value library

# vital.vim

---

<a href="#">Data.Either</a>	either value library
<a href="#">Data.LazyList</a>	lazy list including file io
<a href="#">Data.List</a>	list utilities library
<a href="#">Data.List.Closure</a>	Data.List provider for Data.Closure
<a href="#">Data.List.Byte</a>	Data.List provider for Bytes-List and other bytes-list like data converter.
<a href="#">Data.Optional</a>	optional value library
<a href="#">Data.OrderedSet</a>	ordered collection library
<a href="#">Data.Set</a>	set and frozenset data structure ported from python
<a href="#">Data.String</a>	string utilities library
<a href="#">Data.String.Interpolation</a>	build string with \${}
<a href="#">Data.Tree</a>	tree utilities library
<a href="#">Database.SQLite</a>	sqlite utilities library
<a href="#">DateTime</a>	date and time library
<a href="#">Experimental.Functor</a>	Utilities for functor
<a href="#">Hash.HMAC</a>	Hash-based Message Authentication Code
<a href="#">Hash.MD5</a>	MD5 encoding
<a href="#">Hash.SHA1</a>	SHA1 encoding
<a href="#">Interpreter.Brainf_k</a>	Brain**k interpreter
<a href="#">Locale.Message</a>	very simple message localization library
<a href="#">Mapping</a>	Utilities for mapping
<a href="#">Math</a>	Mathematical functions

# vital.vim

---

Mapping	Utilities for mapping
Math	Mathematical functions
OptionParser	Option parser library for Vim
Prelude	crucial functions
Process	Utilities for process
Random.Mt19937ar	random number generator using mt19937ar
Random.Xor128	random number generator using xor128
Random	Random utility frontend library
Stream	A streaming library

...

And a lot of modules!!

# vital.vim

---

vital.vim is for writing Vim script.  
Meaning also vital.vim for writing your **vimrc**.

```
" Writing expression oriented error messages
let g:vimrc.open_on_gui =
    \ g:vimrc.is_macos ? 'open' :
    \ g:vimrc.is_windows ? 'start' :
    \ g:vimrc.is_unix ? 'xdg-open' : s:Msg.warn('no method

" Do keymapping for the range of @a ~ @z
for x in s>List.char_range('a', 'z')
    execute 'nnoremap' '<silent>' $"@{x}'
        \ (":@<C-u>" .. $"call vimrc#foo("{x}") \<CR>')
endfor
```

nice



---

# vital.vim

---

## My favorite modules.

### First: **Data.List**

```
let s>List = vital#vimrc#import('Data.List')

echo s>List.has([1, 2, 3], 2)
" 1

echo s>List.char_range('a', 'f')
" ['a', 'b', 'c', 'd', 'e', 'f']

echo s>List.count({ x -> x % 2 == 0 }, [1, 2, 3, 4, 5])
" 2
```

# These are VERY basic functions.

```
echo s>List.foldl({ memo, val -> memo + val }, 0, range(1, 10)
" 55
" (= 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10)

echo s>List.intersect(['a', 'b', 'c'], ['b', 'c'])
" ['b', 'c']
```

## vital.vim

---

- `pop`, `shift`, `unshift`, `cons`, `uncons`
- `uniq`, `uniq_by`, `sort`, `sort_by`
- `all`, `any`

vital.vim

---

**Data.Optional**

# vital.vim

---

## Data.Optional

```
let s:Optional = vital#vimrc#import('Data.Optional')

let _1 = s:Optional.none()
" none
let _2 = s:Optional.some(42)
" some(42)
let _3 = s:Optional.new(v:null) " Returns none if v:null, or r
" none
let _4 = s:Optional.new(42)
" some(42)
```

# vital.vim

---

```
let _1 = v:null  
let _2 = 42
```



---

# vital.vim

---

## Expression Oriented Programming

```
call s:Optional.new(s:read_file_if_exist())
    \ ->s:Optional.flat_map({ foo -> s:parse_foo(foo) })
    \ ->s:Optional.optional(
        \ { parsed -> s:make_parsed_file(parsed) },
        \ { -> execute('echo "Nothing to do"') }
    \ )
```

# vital.vim

---

```
call s:Optional.optional(
    \ s:Optional.flat_map(
        \ s:Optional.new(s:read_file_if_exist()),
        \ { foo -> s:parse_foo(foo) })
    \ ) ,
    \ { parsed -> s:make_parsed_file(parsed) } ,
    \ { -> execute('echo "Nothing to do"') }
\ )
```

# vital.vim

---

```
" Folds an optional value to a non optional value
s:Optional.optional(
    \ maybeOptional,
    \ { innerValue -> nonOptionalValue1 },
    \ { -> nonOptionalValue2 }
\ )
```

# vital.vim

---

```
" Extracts innerValue, or throws error
get(optional)

" Extracts innerValue, or to be undefined behavior
get_unsafe(optional)

" Extracts innerValue, or returns alternative value (altValue)
get_or(optional, { -> altValue })
```

# vital.vim

---

```
let optinoalValue =  
  \ s:Optional.flat_map(maybeOptional, { innerValue ->  
    \ optinalValue1  
  \ } )
```

# vital.vim

---

```
" You understood this
call s:Optional.optional(
    \ s:Optional.flat_map(
        \ s:Optional.new(s:read_file_if_exist()) ,
        \ { foo -> s:parse_foo(foo) })
    \ ) ,
    \ { parsed -> s:make_parsed_file(parsed) } ,
    \ { -> execute('echo "Nothing to do"') }
\ )
```

vital.vim

---

**Data.Either**

# vital.vim

---

```
" Where is error info...?
call s:Optional.new(s:read_file_if_exist())
  \ ->s:Optional.flat_map({ foo -> s:parse_foo(foo) })
  \ ->s:Optional.optional(
    \ { parsed -> s:make_parsed_file(parsed) },
    \ { -> execute('echo "Nothing to do"') }
  \ )
```

**vital.vim**

---

>>> Data.Either <<<

# vital.vim

---

```
let s:Either = vital#vimrc#import('Data.Either')

let _1 = s:Either.left('file not found')
" left('file not found')
let _2 = s:Either.right(42)
" right(42)
let _3 = s:Either.null_to_left(v:null, 'it is null')
" left('it is null')
let _4 = s:Either.null_to_left(42, 'it is null')
" right(42)
```

# vital.vim

---

Either is Optional with info.

```
" Meaning right (correct) foo value, or below error message
let result = s:Either.null_to_left(
    \ s:read_foo_file_if_exist(),
    \ 'file foo is not exist.'
\ )
```

# vital.vim

---

```
let eitherValue =  
  \ s:Either.flat_map(maybeRight, { innerValue ->  
    \ eitherValue1  
  \ } )
```

# vital.vim

---

```
" Extracts from a left value, or returns the default value
from_left(defaultValue, either)

" Extracts from a right value, or returns the default value
from_right(defaultValue, either)

" Extracts from a left value, or throws error
unsafe_from_left(either)

" Extracts from a right value, or throws error
unsafe_from_right(either)
```

vital.vim

---

Vim.Message

# vital.vim

---

## Vim.Message

```
let s:Msg = vital#vimrc#import('Vim.Message')

call s:Msg.echo('WarningMsg', 'some warning')
" > some warning

call s:Msg.echomsg('ErrorMsg', 'some error')
" > some error

call s:Msg.warn('some warning')
" > some warning

call s:Msg.error('some error')
" > some error
```

Usually, `:echo` is a syntax (a command).  
But Vim.Message allows to use as an **expression**.

```
let g:vimrc.open_on_gui =
  \ g:vimrc.is_macos    ? 'open' :
  \ g:vimrc.is_windows ? 'start' :
  \ g:vimrc.is_unix   ? 'xdg-open' :
    \ s:Msg.warn('no method for GUI-open')
```

(Also this is useful than `execute('echo "foo"')`)

nice



---

nice



---

**Boost your vimrc with  
some template techniques!**

---

# **END**

---

