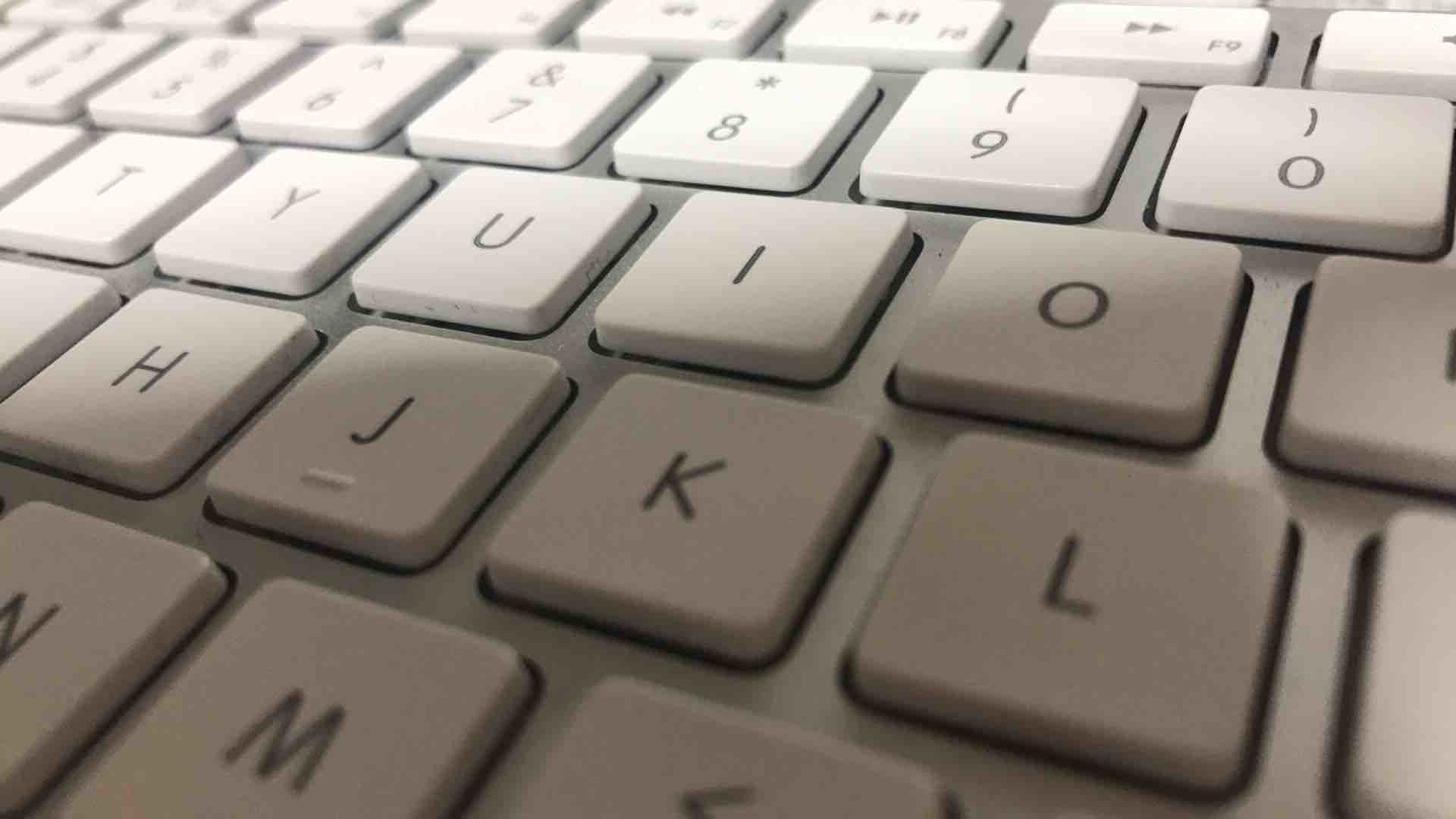
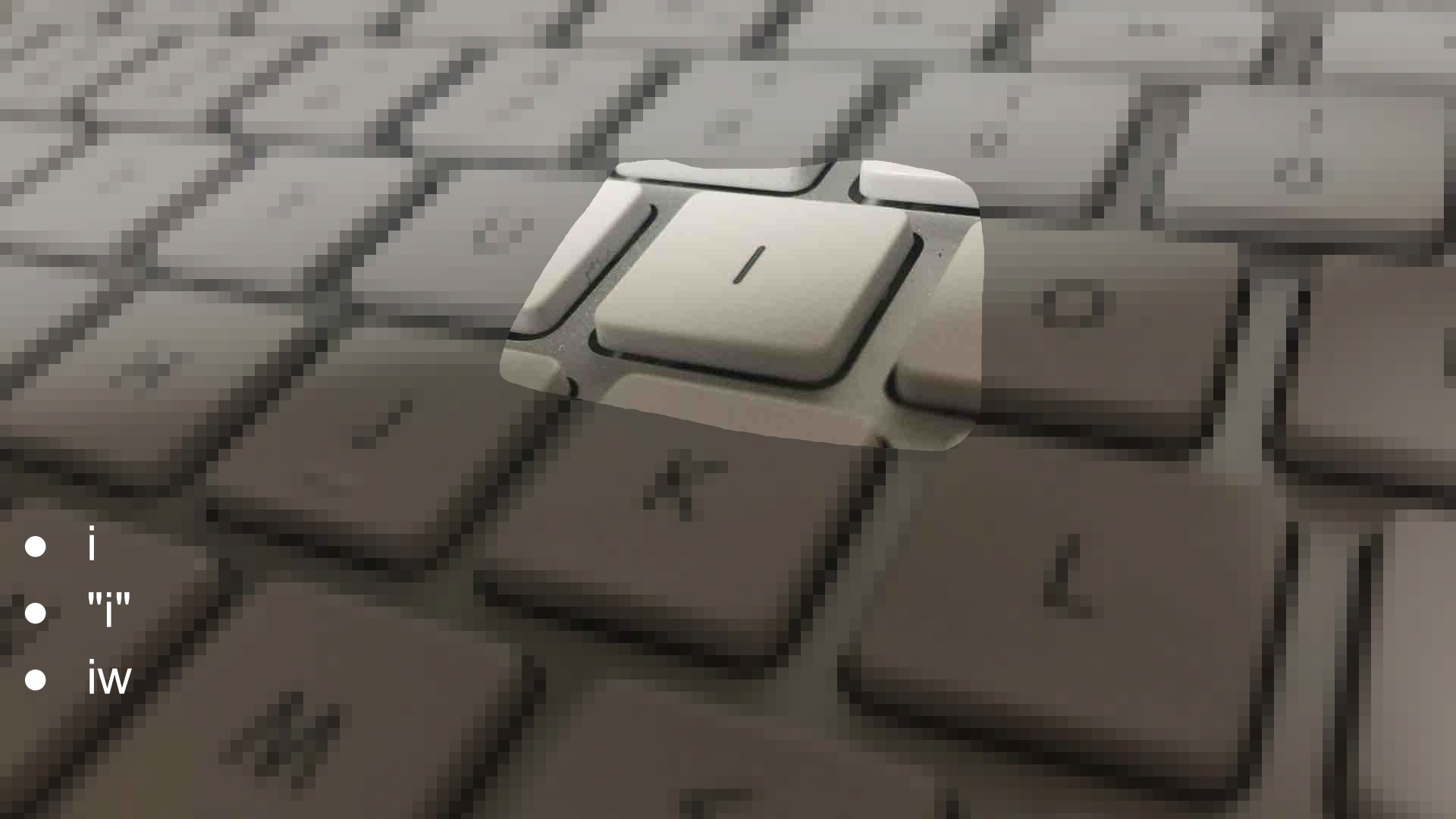
modes

VimConf 2018
2018-11-24
Tatsuhiro Ujihisa


H
J
K
L
M
O
I
U
Y
T
8
9
O



- i
- "i"
- iw

modes

VimConf 2018
2018-11-24
Tatsuhiro Ujihisa

"modes" talk agenda (1/2)

- I talk about Vim modes
- I don't talk about anything other than Vim modes
  - No Vim scripting
  - No vital.vim (obviously)
  - No vim plugins
  - No asynchronous processing
  - No cooking
  - No Cities: Skylines
  - No Civilization 5 or 6
  - No Minecraft or 7 days to die

"modes" talk agenda (2/2)

- Understand what Vim's modes are
  - by the specification and implementation
  - Introduce how they are and how they work
- Tools I use today
  - GDB
  - termdebug.vim (built-in plugin)
- Target audience
  - Beginner and intermediate Vim users
  - Vim plugin authors
  - **Not** active Vim core developers

"modes" talk agenda

- Understand what Vim's modes are
- T
- T
  - Beginner and intermediate Vim users
  - Vim plugin authors
  - **Not** active Vim core developers

Goals:

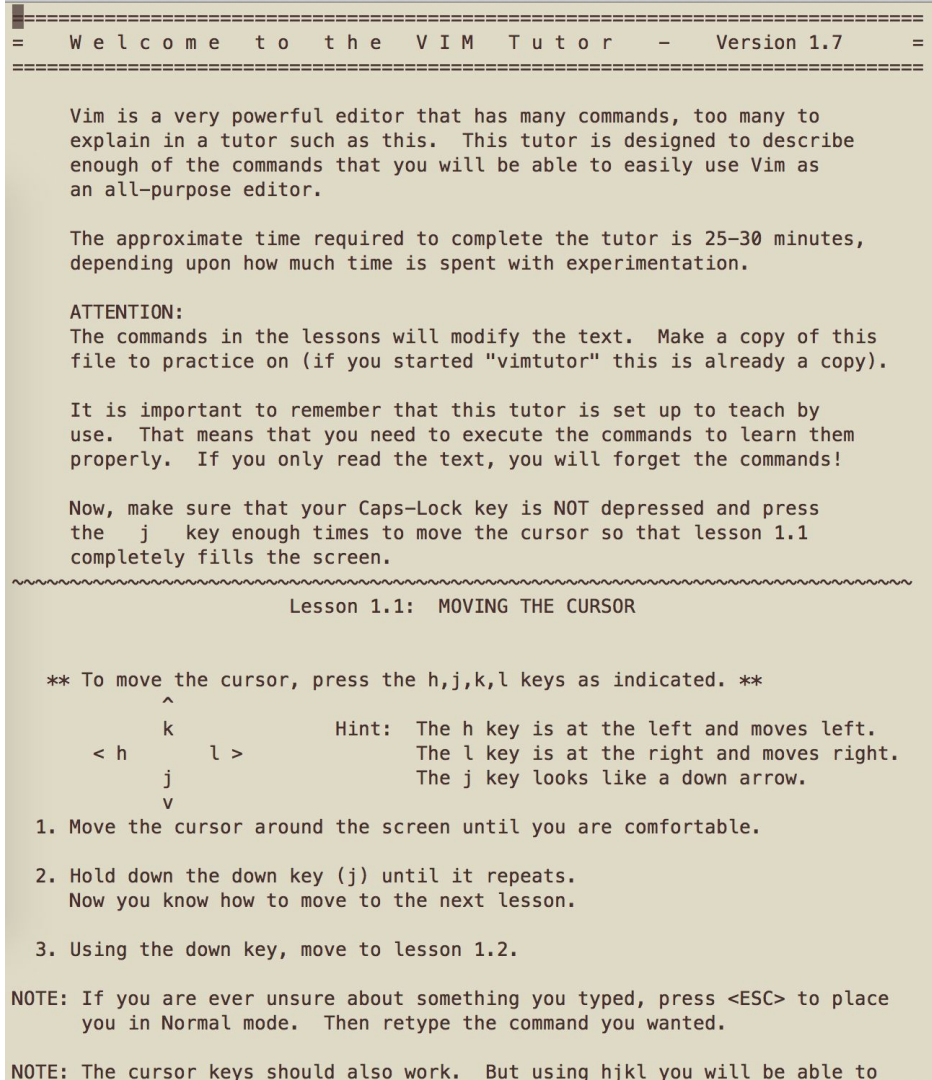
- feel more confident at Vim core
- start working on contributing vim core

Understanding specification

Let's see the doc first

$ vimtutor

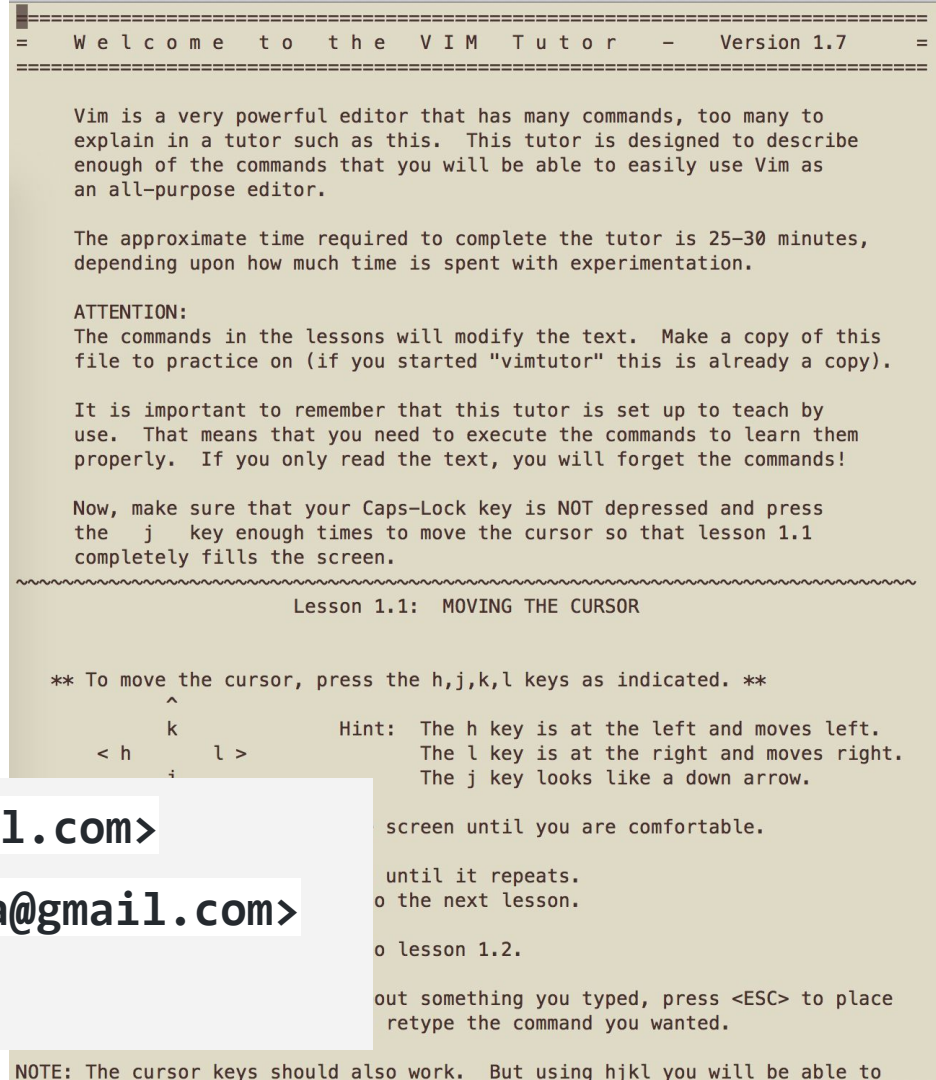
- Have you done?
  - /usr/bin/vimtutor
  - 25-30 minutes
- Not good at English?
  - `vimtutor ja`



```
=   W e l c o m e   t o   t h e   V I M   T u t o r   -   Version 1.7     =
===============================================================================

     Vim is a very powerful editor that has many commands, too many to
     explain in a tutor such as this.  This tutor is designed to describe
     enough of the commands that you will be able to easily use Vim as
     an all-purpose editor.

     The approximate time required to complete the tutor is 25-30 minutes,
     depending upon how much time is spent with experimentation.

     ATTENTION:
     The commands in the lessons will modify the text.  Make a copy of this
     file to practice on (if you started "vimtutor" this is already a copy).

     It is important to remember that this tutor is set up to teach by
     use.  That means that you need to execute the commands to learn them
     properly.  If you only read the text, you will forget the commands!

     Now, make sure that your Caps-Lock key is NOT depressed and press
     the   j   key enough times to move the cursor so that lesson 1.1
     completely fills the screen.
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                      Lesson 1.1:  MOVING THE CURSOR


   ** To move the cursor, press the h,j,k,l keys as indicated. **
             ^
             k               Hint:  The h key is at the left and moves left.
        < h       l >               The l key is at the right and moves right.
             j                      The j key looks like a down arrow.
             v
  1. Move the cursor around the screen until you are comfortable.

  2. Hold down the down key (j) until it repeats.
     Now you know how to move to the next lesson.

  3. Using the down key, move to lesson 1.2.

NOTE: If you are ever unsure about something you typed, press <ESC> to place
      you in Normal mode.  Then retype the command you wanted.

NOTE: The cursor keys should also work.  But using hjkl you will be able to
```

$ vimtutor

- Have you done?
  - /usr/bin/vimtutor
  - 25-30 minutes
- Not good at English?
  - `vimtutor ja`



日本語訳　松本 泰弘　<mattn.jp@gmail.com>

監修　村岡 太郎　<koron.kaoriya@gmail.com>

Lesson 1.4: TEXT EDITING - INSERTION

- *** Press **i** to insert text. ****
  - *press **<ESC>** to return to Normal mode.*
- Next lesson is about **A**

:help

- `:h` (without arguments)
  - it opens help.txt, the index of helps
- help.txt → (intro.txt | howto.txt) → mode-switching
- `:h mode-switching`
  - (or `:h mode-s`)

:h mode-switching

```
6. Switching from mode to mode                                  mode-switching

If for any reason you do not know which mode you are in, you can always get
back to Normal mode by typing <Esc> twice.  This doesn't work for Ex mode
though, use ":visual".
You will know you are back in Normal mode when you see the screen flash or
hear the bell after you type <Esc>.  However, when pressing <Esc> after using
CTRL-O in Insert mode you get a beep but you are still in Insert mode, type
<Esc> again.

                                                                  i_esc
                TO mode
                Normal  Visual  Select  Insert    Replace   Cmd-line  Ex
FROM mode
Normal                  v V ^V    *4      *1        R gR     : / ? !   Q
Visual           *2                 ^G      c C       --       :       --
Select           *5     ^O ^G               *6        --       --      --
Insert           <Esc>    --       --               <Insert>   --      --
```

i_esc

	TO mode						
	Normal	Visual	Select	Insert	Replace	Cmd-line	Ex
FROM mode							
Normal		v V ^V	*4	*1	R gR	: / ? !	Q
Visual	*2		^G	c C	--	:	--
Select	*5	^O ^G		*6	--	--	--
Insert	<Esc>	--	--		<Insert>	--	--
Replace	<Esc>	--	--	<Insert>		--	--
Command-line	*3	--	--	:start	--		--
Ex	:vi	--	--	--	--	--	

i_esc

	TO mode						
	Normal	Visual	Select	Insert	Replace	Cmd-line	Ex
FROM mode							
Normal		v V ^V	*4	*1	R gR	: / ? !	Q
Visual	*2		^G	c C	--	:	--
Select	*5	^O ^G		*6	--	--	--
Insert	<Esc>	--	--		<Insert>	--	--
Replace	<Esc>	--	--	<Insert>		--	--
Command-line	*3	--	--	:start	--		--
Ex	:vi	--	--	--	--	--	

*1 Go from Normal mode to Insert mode by giving the command "i", "I", "a", "A", "o", "O", "c", "C", "s" or S".

i_esc

FROM mode \ TO mode	Normal	Visual	Select	Insert	Replace	Cmd-line	Ex
Normal		v V ^V	*4	*1	R gR	: / ? !	Q
Visual	*2		^G	c C	--	:	--
Select	*5	^O ^G		*6	--	--	--
Insert	<Esc>	--	--		<Insert>	--	--
Replace	<Esc>	--	--	<Insert>		--	--
Command-line	*3	--	--	:start	--		--
Ex	:vi	--	--	--	--	--	

*2 Go from Visual mode to Normal mode by giving a non-movement command, which causes the command to be executed, or by hitting <Esc> "v", "V" or "CTRL-V" (see v_v), which just stops Visual mode without side effects.

i_esc

FROM mode \ TO mode	Normal	Visual	Select	Insert	Replace	Cmd-line	Ex
Normal		v V ^V	*4	*1	R gR	: / ? !	Q
Visual	*2		^G	c C	--	:	--
Select	*5	^O ^G		*6	--	--	--
Insert	<Esc>	--	--		<Insert>	--	--
Replace	<Esc>	--	--	<Insert>		--	--
Command-line	*3	--	--	:start	--		--
Ex	:vi	--	--	--	--	--	

*3 Go from Command-line mode to Normal mode by:
- Hitting <CR> or <NL>, which causes the entered command to be executed.
- Deleting the complete line (e.g., with CTRL-U) and giving a final <BS>.
- Hitting CTRL-C or <Esc>, which quits the command-line without executing the command.

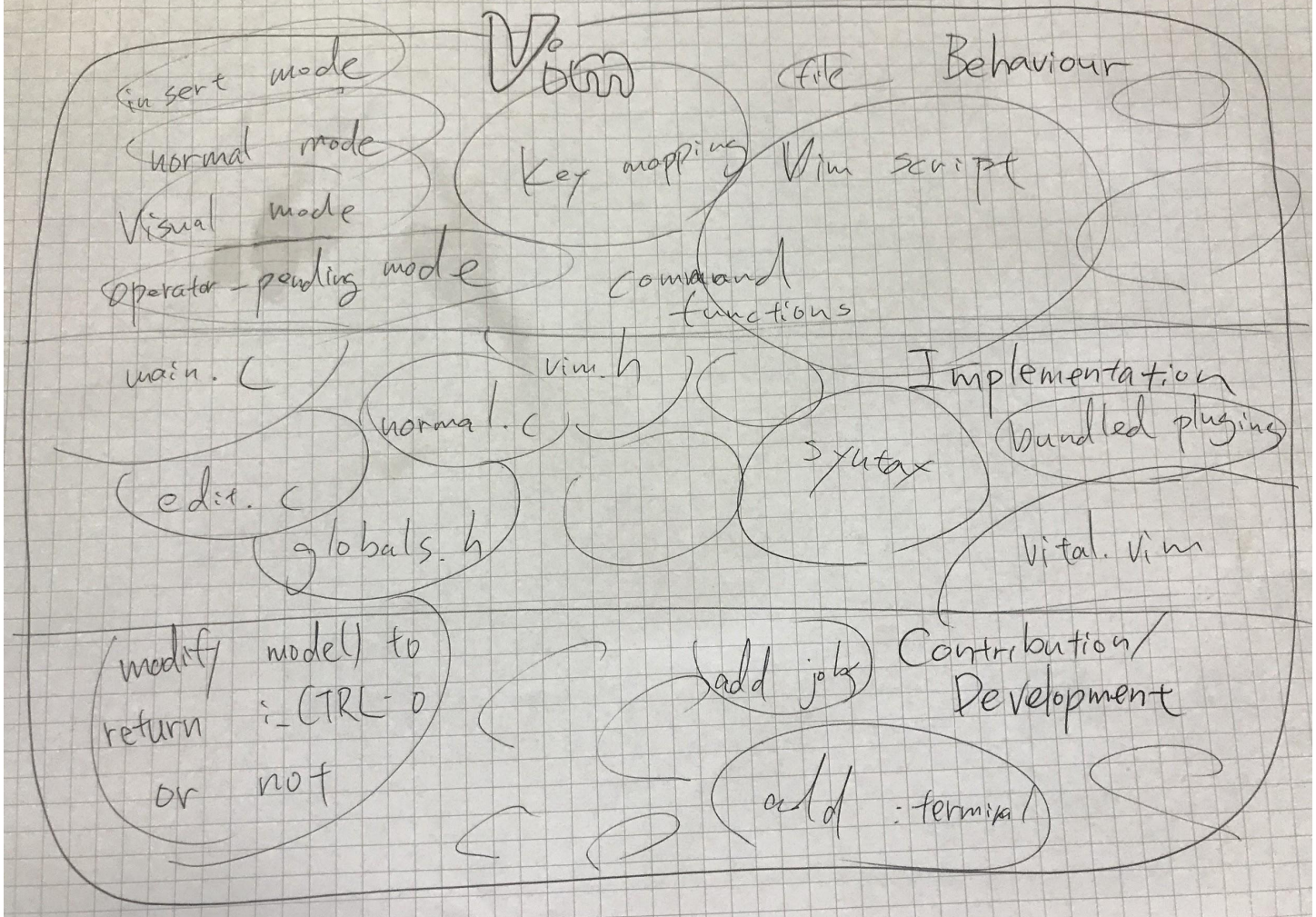
In the last case <Esc> may be the character defined with the 'wildchar' option, in which case it will start command-line completion. You can ignore that and type <Esc> again. {Vi: when hitting <Esc> the command-line is executed. This is unexpected for most people; therefore it was changed in Vim. But when the <Esc> is part of a mapping, the command-line is executed. If you want the Vi behaviour also when typing <Esc>, use ":cmap ^V<Esc> ^V^M"}

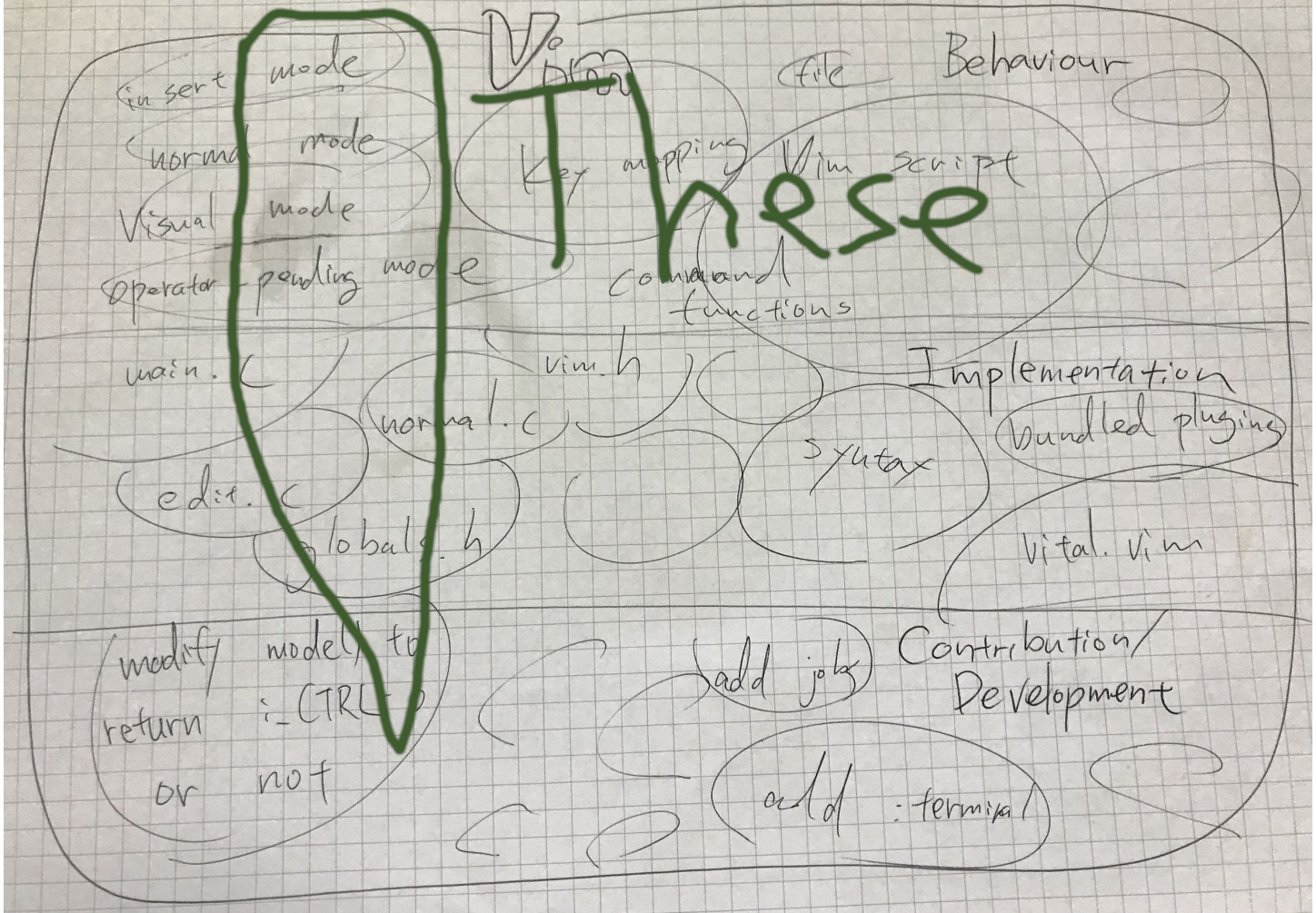
More modes (:h vim-modes)

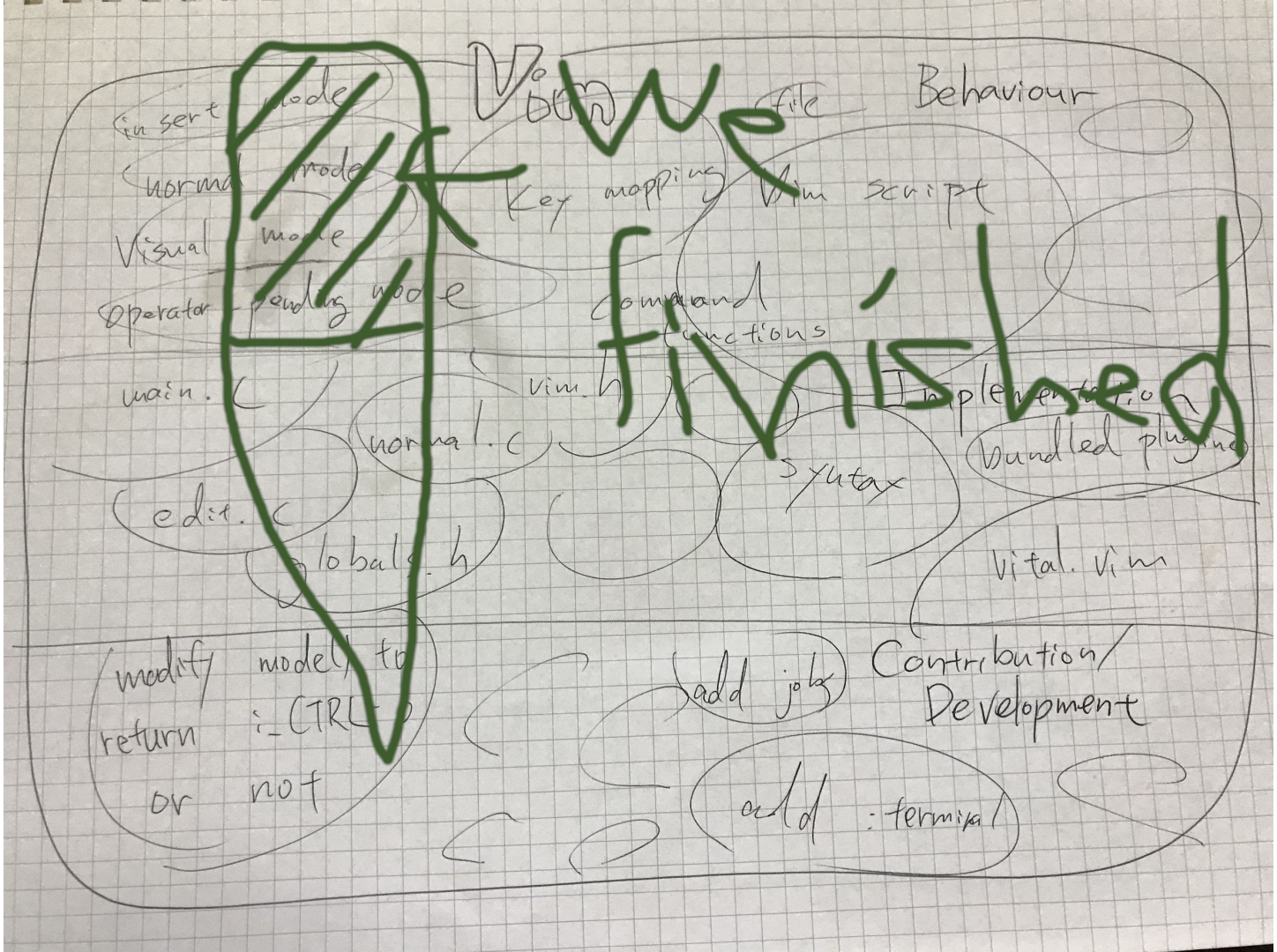
- Normal
- Visual
- Select
- Insert
- Replace
- Command-line (Cmdline)
- Ex
- Terminal-Job
- Operator-pending
- Virtual Replace
- Insert Normal
- Terminal-Normal
- Insert Visual
- Insert Select

Short summary

- vimtutor
- :h
- 7 modes + 7 additional modes
- Many ways switch between modes
- Each key behaviour depend on the current mode


Vim
insert mode
normal mode
Visual mode
Operator-pending mode
key mapping
file
Behaviour
Vim script
command
functions
main.c
vim.h
normal.c
edit.c
globals.h
syntax
Implementation
bundled plugins
vital.vim
modify mode() to
return :_CTRL-O
or not
add job
add :terminal
Contribution/
Development


Vim
These
insert mode
normal mode
Visual mode
Operator-pending mode
file
Behaviour
key mapping
Vim script
command functions
main.c
vim.h
normal.c
edit.c
globals.h
Implementation
syntax
bundled plugins
vital.vim
modify mode return or not
:_CTRL-C
add jobs
add :terminal
Contribution/
Development

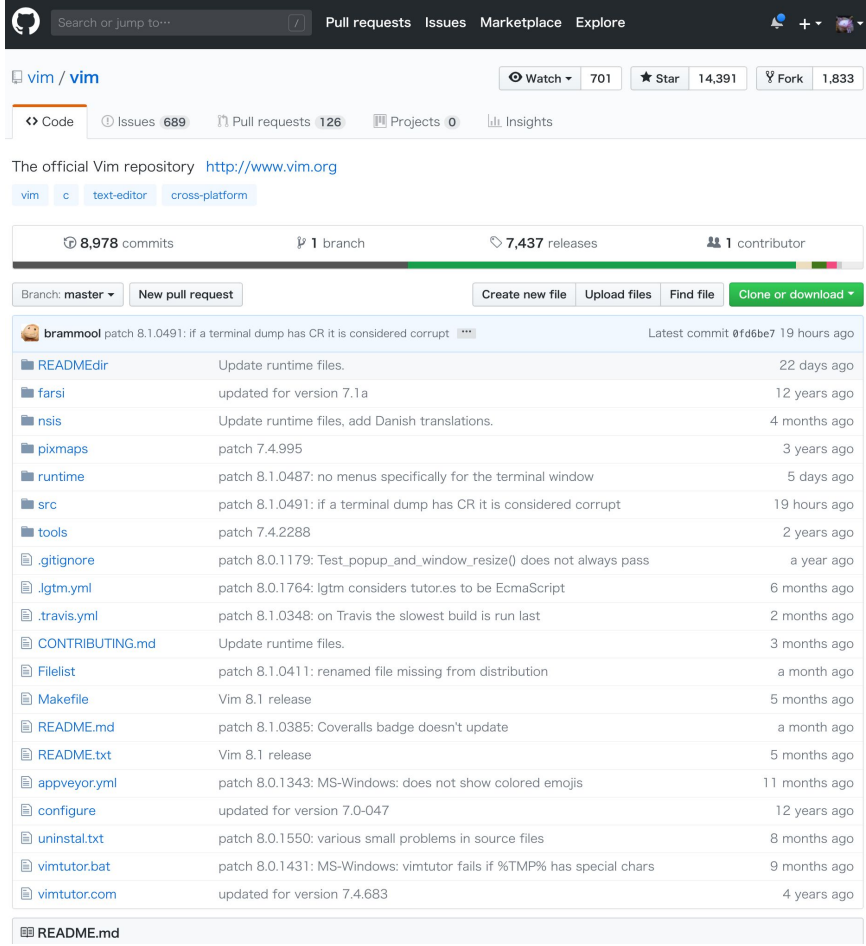

Vim
finished
Behaviour
file
key mapping
Vim script
command functions
Implementation
bundled plugins
syntax
vital.vim
Contribution/
Development
add jobs
add :terminal
insert mode
normal mode
Visual mode
Operator-pending mode
main.c
vim.h
normal.c
edit.c
globals.h
modify mode() to return :_CTRL-G or not

Understanding ~~specification~~ implementation

Let's see the Vim C code now

https://github.com/vim/vim

- src/**/*.c
- everything is there



README.md	Update runtime files.	22 days ago
farsi	updated for version 7.1a	12 years ago
nsis	Update runtime files, add Danish translations.	4 months ago
pixmaps	patch 7.4.995	3 years ago
runtime	patch 8.1.0487: no menus specifically for the terminal window	5 days ago
src	patch 8.1.0491: if a terminal dump has CR it is considered corrupt	19 hours ago
tools	patch 7.4.2288	2 years ago
.gitignore	patch 8.0.1179: Test_popup_and_window_resize() does not always pass	a year ago
.lgtm.yml	patch 8.0.1764: lgtm considers tutor.es to be EcmaScript	6 months ago
.travis.yml	patch 8.1.0348: on Travis the slowest build is run last	2 months ago
CONTRIBUTING.md	Update runtime files.	3 months ago
Filelist	patch 8.1.0411: renamed file missing from distribution	a month ago
Makefile	Vim 8.1 release	5 months ago
README.md	patch 8.1.0385: Coveralls badge doesn't update	a month ago
README.txt	Vim 8.1 release	5 months ago
appveyor.yml	patch 8.0.1343: MS-Windows: does not show colored emojis	11 months ago
configure	updated for version 7.0-047	12 years ago
uninstal.txt	patch 8.0.1550: various small problems in source files	8 months ago
vimtutor.bat	patch 8.0.1431: MS-Windows: vimtutor fails if %TMP% has special chars	9 months ago
vimtutor.com	updated for version 7.4.683	4 years ago

https://github.com/ujihisa

- Vancouver, Canada
  Tokyo, Japan
- Vim for about 20 years
- Ruby on Rails,
  Scala for distributed systems,
  Clojure, Haskell for myself,
  and Vim script
- VimConf founder
  ujihisa.vim

(unorganized) 115 plugins I use

agit.vim
aldmeris
altr
ansible-yaml
asterisk
autodirmake.vim
calendar.vim
caw.vim
coffee-script
colors-pencil
colors-solarized
Colour-Sampler-Pack
concealedyank.vim
context_filetype.vim
cpp
cruby
cursorword
deol.nvim
deoplete.nvim
elixir
filetype-haskell
fontzoom
ft-clojure
ft-cmake
ft-mongo
game-code-break
game_engine.vim
ghcmod
gina.vim
go
groovyindent
haskell.vim
hug-neovim-rpc
iferr/vim
incsearch.vim
J6uil.vim
javaclasspath
jplus
kotlin
lexima.vim
linediff.vim
ltsv
mario.vim
metaffer
monokai
nclipper.vim
neco
neco-ghc
neco-syntax
.neobundle
neobundle.vim
neobundle.vim
neobundle-vim-recipes
neochat.vim
neoclojure.vim
neoinclude.vim
neomru.vim
neopairs.vim
neosnippet
neosnippet-snippets
neverland-vim-theme
nvim-yarp
open-browser.vim
operator-replace
operator-user
papercolor-theme
perfect.vim
poslist
prettyprint
puyo.vim
quicklearn
quickrun
Rainbow-Parenthsis-Bundle
ref
ref-hoogle
ref-ri
rengbang
reversi.vim
ruby
scala
sexp
showtime
smartchr
sudo.vim
surround
tabpagebuffer.vim
tabpagecd
textobj-syntax
textobj-user
textobj-wiw
themis
translua
typescript
unite-build
unite-colorscheme
unite-giti
unite-haskellimport
unite-help
unite-history
unite-include-reversed
unite-javaimport
unite-locate
unite-outline
unite-ruby-require.vim
unite-ssh
unite.vim
vimerl
vimlint
vimlparser
vimport
vimproc
vimshell
vimshell-ssh
vital.vim
zenesque.vim

(unorganized) 115 plugins I use

agit.vim
aldmeris
altr
ansible-yaml
asterisk
autodirmake.vim
groovyindent
haskell.vim
hug-neovim-rpc
iferr/vim
incsearch.vim
J6uil.vim
neverland-vim-theme
nvim-yarp
open-browser.vim
operator-replace
operator-user
papercolor-theme
themis
translua
typescript
unite-build
unite-colorscheme
unite.git

https://github.com/ujihisa/config

vimrc: 2772 lines (not organized at all)

deol.nvim
deoplete.nvim
elixir
filetype-haskell
fontzoom
ft-clojure
ft-cmake
ft-mongo
game-code-break
game_engine.vim
ghcmod
gina.vim
go
neco-ghc
neco-syntax
.neobundle
neobundle.vim
neobundle.vim
neobundle-vim-recipes
neochat.vim
neoclojure.vim
neoinclude.vim
neomru.vim
neopairs.vim
neosnippet
neosnippet-snippets
reversi.vim
ruby
scala
sexp
showtime
smartchr
sudo.vim
surround
tabpagebuffer.vim
tabpagecd
textobj-syntax
textobj-user
textobj-wiw
vimlint
vimlparser
vimport
vimproc
vimshell
vimshell-ssh
vital.vim
zenesque.vim

(unorganized) 115 plugins I use

agit.vim
aldmeris
altr
ansible-yaml
asterisk
groovyindent
haskell.vim
hug-neovim-rpc
iferr/vim
incsearch.vim
neverland-vim-theme
nvim-yarp
open-browser.vim
operator-replace
operator-user
themis
translua
typescript
unite-build
unite-colorscheme

https://github.com/ujihisa/config

vimrc: 2772 lines (not organized at all)

deol.nvim
deoplete.nvim
elixir
filetype-haskell
fontzoom
ft-clojure
ft-cmake
ft-mongo
game-code-break
game_engine.vim
ghcmod
gina.vim
go
neco-ghc
neco-syntax
.neobundle
neobundle.vim
neobundle.vim
neobundle-vim-recipes
neochat.vim
neoclojure.vim
neoinclude.vim
neomru.vim
neopairs.vim
neosnippet
neosnippet-snippets
reversi.vim
ruby
textobj-user
textobj-wiw
vimlint
vimlparser

DEMO
Write something and run it

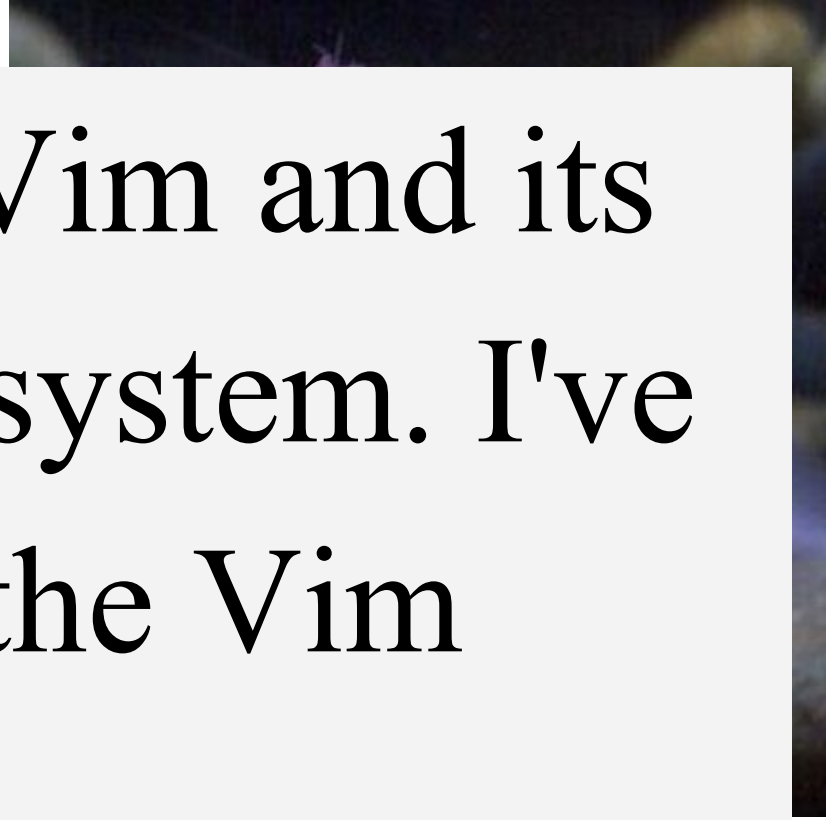
working at quipper (2018-09 ~)

- Ruby on Rails
- React and Redux
- https://www.quipper.com/
- Education service

https://github.com/ujihisa

- Vancouver, Canada
  Tokyo, Japan
- Vim for about 20 years
- **Ruby** on Rails,
  Scala for distributed systems,
  Clojure, **Haskell** for myself,
  and **Vim script**
- VimConf founder / staff
  ujihisa.vim

https://github.com/uiihisa



Thanks a lot for Vim and its development ecosystem. I've been living with the Vim community.

https://github.com/ujihisa

- Vancouver, Canada
  Tokyo, Japan
- Vim for about 20 years
- **Ruby** on Rails,
  Scala for distributed systems,
  Clojure, **Haskell** for myself,
  and **Vim script**
- VimConf founder / staff
  ujihisa.vim

How to start

- Read C code from top to bottom
- Find a specific function, and read it carefully
- Run, and see what debugger shows
  - GDB

Vim

Behaviour

file

key mapping

Vim script

command

functions

Implementation

bundled plugins

syntax

vital.vim

insert mode

normal mode

Visual mode

Operator-pending mode

main.c

vim.h

normal.c

edit.c

globals.h

modify return or not

mode to :_CTRL-C

Contribution/
Development

add jobs

add :terminal

Understanding implementation

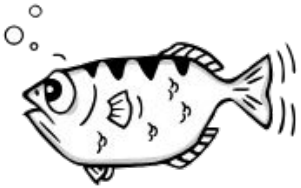
Let's see the Vim C code now

main.c

```
111     {
112     #if defined(STARTUPTIME) || defined(CLEAN_RUNTIMEPATH)
113         int         i;
114     #endif
115
116         /*
117          * Do any system-specific initialisations.  These can NOT use IObuff or
118          * NameBuff.  Thus emsg2() cannot be called!
119          */
120         mch_early_init();
121
122     #if defined(WIN32) && defined(FEAT_MBYTE)
123         /*
124          * MinGW expands command line arguments, which confuses our code to
125          * convert when 'encoding' changes.  Get the unexpanded arguments.
126          */
127         argc = get_cmd_argsW(&argv);
128     #endif
129
130         /* Many variables are in "params" so that we can pass them to invoked
131          * functions without a lot of arguments.  "argc" and "argv" are also
132          * copied, so that they can be changed. */
133         vim_memset(&params, 0, sizeof(params));
134         params.argc = argc;
135         params.argv = argv;
136         params.want_full_screen = TRUE;
137     #ifdef FEAT_EVAL
138         params.use_debug_break_level = -1;
139     #endif
140         params.window_count = -1;
141
142     #ifdef FEAT_RUBY
143         {
144             int ruby_stack_start;
145             vim_ruby_init((void *)&ruby_stack_start);
146         }
147     #endif
148
149     #ifdef FEAT_TCL
```

exec No process In:

```
Reading symbols from /home/ujihisa/git/vim/local/bin/vim...done.
(gdb)
```



GDB

See the behaviour dynamically

GDB

- The GNU Project Debugger
- "go inside a running program"
- Run Vim with GDB
  - The Vim needs to be compiled with some special flags

Build a GDB-Friendly Vim

- git clone from github.com/vim/vim
- edit src/Makefile manually
  - CFLAGS=-ggdb3 to include debug info
    - -O0 to disable optimization
    - -g to include debug info
    - -g2 for more
    - -g3 for even more
    - -ggdb3 for even more just for gdb
    - (See `man gcc` for details)
  - STRIP=/bin/true not to strip (/usr/bin/true for mac)

Build a GDB-Friendly Vim

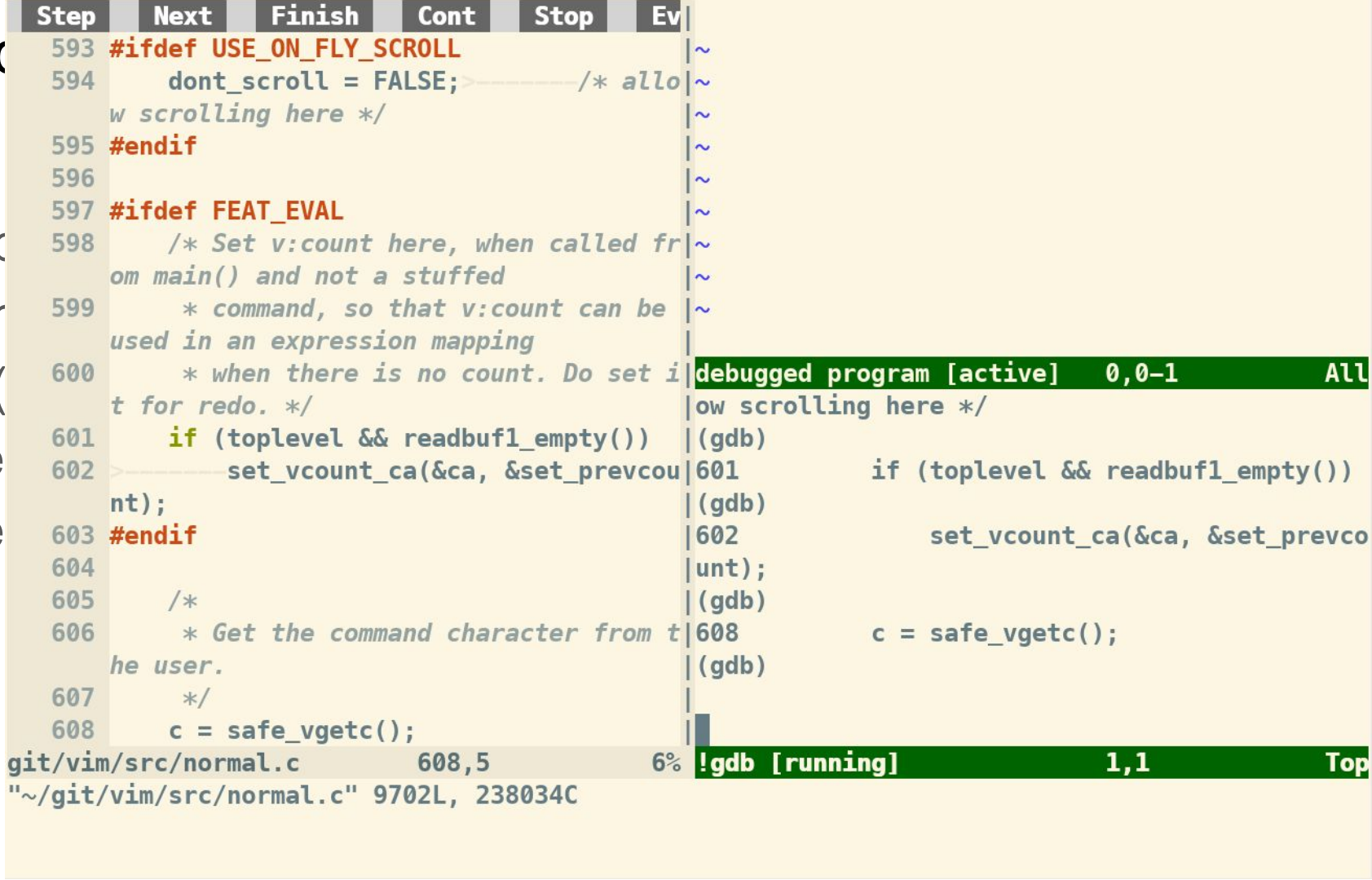
```
diff --git a/src/Makefile b/src/Makefile
index 5b25e033f..0e3051a26 100644
--- a/src/Makefile
+++ b/src/Makefile
@@ -591,7 +591,7 @@ CClink = $(CC)
 # When using -g with some older versions of Linux you might get a
 # statically linked executable.
 # When not defined, configure will try to use -O2 -g for gcc and -O for cc.
-#CFLAGS = -g
+CFLAGS = -ggdb3
 #CFLAGS = -O

 # Optimization limits - depends on the compiler.  Automatic check in configure
@@ -1005,7 +1005,7 @@ TOOLS = xxd/xxd$(EXEEXT)
 ### prefix the top directory for the data (default "/usr/local")
 #
 # Uncomment the next line to install Vim in your home directory.
-#prefix = $(HOME)
+prefix = $(HOME)/git/vim/local

 ### exec_prefix is the top directory for the executable (default $(prefix))
 #
@@ -1156,7 +1156,7 @@ INSTALL_DATA_R = cp -r

 ### Program to run on installed binary.  Use the second one to disable strip.
 #STRIP = strip
-#STRIP = /bin/true
+STRIP = /bin/true

 ### Permissions for binaries  {{{1
 BINMOD = 755
```

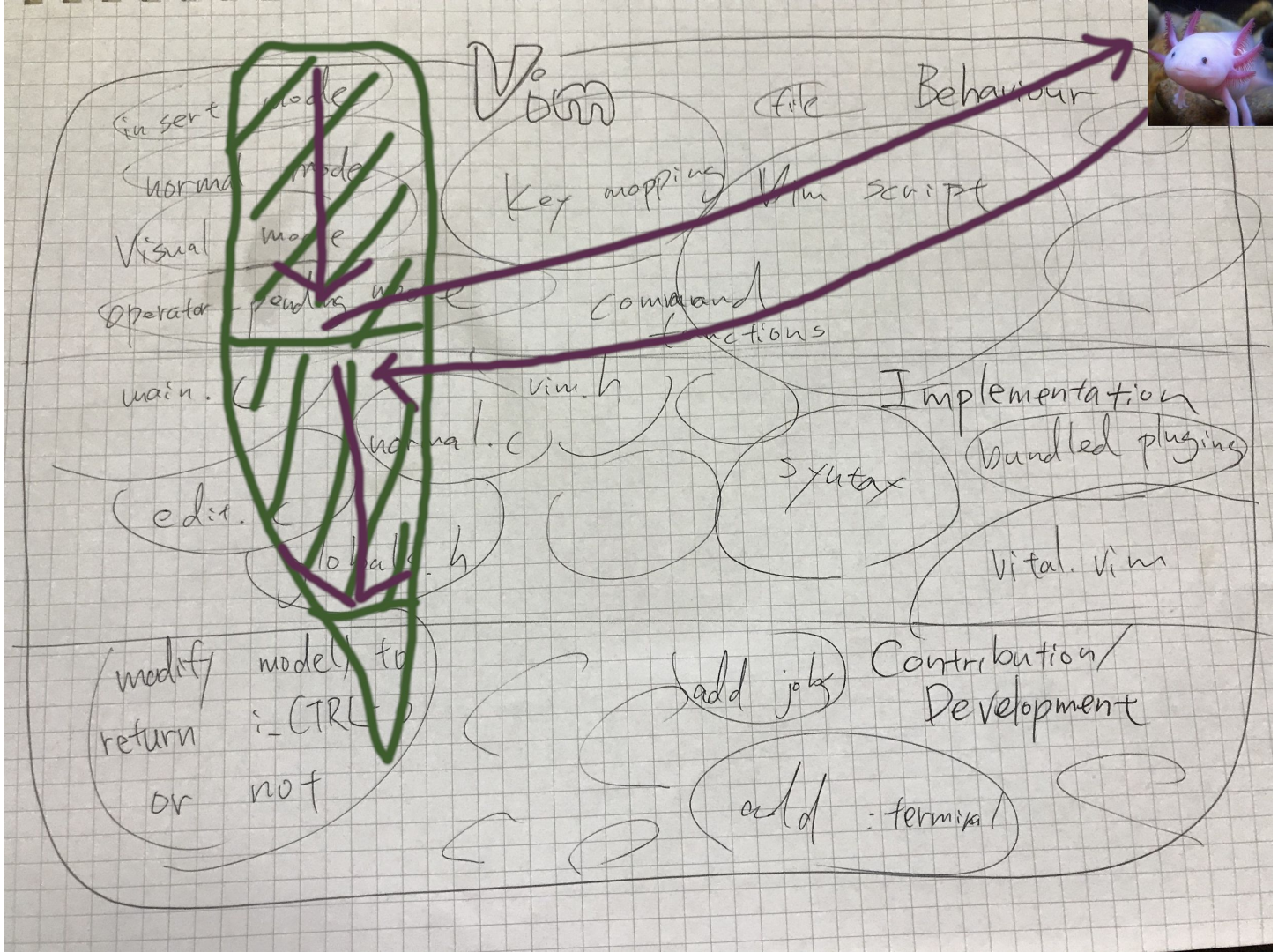
:Termdebug

- Vim wrapper for GDB
  - powered by :terminal
  - made by Bram Moolenaar
  - (He introduced at his talk)
- Use vim to see the source code
- Use :terminal to run a debuggee program (i.e. Vim)

:Termc

- Vim
  - p
  - r
  - (
- Use
- Use




Vim
insert mode
normal mode
Visual mode
Operator-pending mode
file
Behaviour
key mapping
Vim script
command functions
main.c
vim.h
normal.c
edit.c
globals.h
Implementation
bundled plugins
syntax
vital.vim
modify mode to return or not
:_CTRL-C
add jobs
Contribution/ Development
add :terminal

Short summary

- GDB: step execution / view code
- Termdebug: Vim in different window / code in Vim
- main() ->vim_main2()->main_loop()
- normal_cmd()
  - safe_vgetc()
- find_command() / nv_cmds[idx]
- nv_edit()
- edit()
- restart_edit

Contribution / Development

Let me show my example quickly

- Make pull requests to vim/vim
  - see existing issues first
  - vim-jp also has some
- ujihisa's contributions (vim-core)
  - https://github.com/vim/vim/search?q=author%3Aujihisa&unscoped_q=author%3Aujihisa&type=Issues

Provide a way to tell if a command is executed from i_CTRL-O or not #3000

Closed ujihisa wants to merge 6 commits into `vim:master` from `ujihisa:mode`

ujihisa commented on Jun 12

Problem

A user-defined command cannot tell if it's executed as an Ex command, or as `i_CTRL-O` from insert mode. This makes it hard to implement some Vim plugins, such as one that inserts text into the buffer with adjusting contents depending on its context.

Example use case: https://github.com/koron/iferr `:IfErr` command inserts a Go code snippet at the

Pipeline

Closed

Review

No revie

Assigne

No one

10 src/evalfunc.c

```
@@ -8313,10 +8313,9 @@ f_mkdir(typval_T *argvars, typval_T *rettv)
8313 8313       static void
8314 8314   f_mode(typval_T *argvars, typval_T *rettv)
8315 8315   {
8316      -     char_u    buf[3];
     8316 +     char_u    buf[4];
8317 8317
8318      -     buf[1] = NUL;
8319      -     buf[2] = NUL;
     8318 +     vim_memset(buf, 0, sizeof(buf));
8320 8319
8321 8320       if (time_for_testing == 93784)
8322 8321       {
@@ -8382,6 +8381,11 @@ f_mode(typval_T *argvars, typval_T *rettv)
8382 8381         buf[0] = 'n';
8383 8382         if (finish_op)
8384 8383             buf[1] = 'o';
     8384 +       else if (restart_edit == 'I' || restart_edit == 'R' || restart_edit == 'V')
     8385 +       {
     8386 +           buf[1] = 'i';
     8387 +           buf[2] = restart_edit;
     8388 +       }
8385 8389       }
8386 8390
8387 8391       /* Clear out the minor mode when the argument is not a non-zero number or
```

f_mode()
 void
 put values into rettv

char_u buf[4]
 used for return value
 rettv->vval.v_string =
  vim_strsave(buf);

buf[0]
 'n', 'i', ...
 mode() uses just this part

buf[1]
 NULL, 'i', ...
buf[2]
 NULL, 'I', 'R', or 'V'
 currently only for C-o

https://github.com/ujihisa

- https://twitter.com/ujm

  ↑ I'll share my talk slides today

- Tatsuhiro Ujihisa
- Tokyo, Japan

  and

  Vancouver, Canada