Migrating plugins to standard features

VimConf 2018
daisuzu

About me

- daisuzu(Daisuke Suzuki)
  - https://twitter.com/dice_zu
  - https://github.com/daisuzu
  - https://daisuzu.hatenablog.com
- Vim experience
  - 10 years
- Jobs
  - Testing engineer ➡ Server side software engineer
- Languages
  - Perl
  - Python
  - Go

Introduction

- Vim has many useful features built-in
  - It is not poor even without plugins
  - Most plugins are made with a combination of standard features
- My Vim life depended on many plugins, but migrated to the standard features
  - Not completely
- Although plugins are important for efficient use of Vim

- **Understand Vim's standard features deeply**
- **Be able to use plugins more effectively**
- **Make it a opportunity to create plugin**

Agenda

1. How I used Vim
   - Testing engineer
   - Server side software engineer
2. How to migrate the following plugins
   - neocomplete
     - Auto-completion
   - neobundle
     - Plugin manager
   - unite
     - File finder
   - vimfiler
     - File manager

Note: Shougo ware only? Because I was very grateful to him.

A testing Engineer meet Vim

- I started using Vim to check the log of embedded devices
  - ***KaoriYa Vim*** on Windows XP
  - Other options are ***Maruo***, ***SAKURA*** or ***Emacs(Meadow)***
- However, there were a lot of things I could not understand
  - mswin.vim + arrow keys
- In addition, I was using completely different from now
  - Normal mode centric
    - Vertical movement
    - Marks
  - Function keys

```
map <F11> :vimgrep /MANY MANY MANY PATTERNS/ %
map <F12> :SearchReinit<CR>:SearchReset<CR>:Search KEYWORD#1<CR>:Search KEYWORD#2<CR>
:Search KEYWORD#3<CR>:Search KEYWORD#4<CR>:Search KEYWORD#5<CR>:Search KEYWORD#6<CR>
:Search KEYWORD#7<CR>:Search KEYWORD#8<CR>:Search KEYWORD#9<CR>
```

Useful plugins for checking logs

- Grep
  - sf.vim : Fold everything except search results
  - ttoc : A regexp-based table of contents of the current buffer
  - grep.vim : Search tools (grep, ripgrep, ack, ag, findstr, git grep) integration with Vim
  - occur.vim : Show all lines in the buffer containing word (grep buffer)
  - QFixGrep : A grep plugin with preview & refine search (and search)
- Mark
  - wokmarks.vim : Local marks usage more similar to other editors
  - marksbrowser.vim : A graphical marks browser
- Highlight
  - MultipleSearch : Highlight multiple searches at the same time, each with a different color
  - quickhl.vim : Quickly highlight multiple word
  - rainbowcyclone.vim : A vim plugin to highlight different color for each search

Using insert mode to improve operation

- Create if_pyth plugins and python scripts
  - Several utilities
  - Alternative to grep
- Non-programmers can not write code without assistance of plugins
  - neocomplcache
  - neocomplcache-snippets-complete
  - python-mode
  - etc.
- I felt my skill has been enhanced by plugins!?
  - Plugin is power

My Vim plugins have up to one hundred and eight

1. neobundle.vim
2. vim-pathogen
3. vim-ipi
4. vimdoc-ja
5. vim-ref
6. neocomplcache
7. neocomplcache-snippets-complete
8. neocomplcache-clang
9. neco-ghc
10. jscomplete-vim
11. taglist.vim
12. TagHighlight
13. vim-fugitive
14. gitv
15. vim-extradite
16. unite.vim
17. unite-build
18. unite-colorscheme
19. quicklearn
20. unite-qf
21. unite-outline
22. vim-alignta
23. unite-help
24. unite-tag
25. unite-mark
26. unite-everything
27. unite-scriptnames
28. unite-webcolorname
29. unite-grep_launcher
30. unite-gtags
31. vim-textobj-user
32. vim-textobj-indent
33. vim-textobj-syntax
34. vim-textobj-line
35. vim-textobj-fold
36. vim-textobj-entire
37. vim-textobj-between
38. vim-textobj-comment
39. textobj-wiw
40. vim-textobj-sigil
41. vim-operator-user
42. vim-operator-replace
43. operator-camelize.vim
44. operator-reverse.vim
45. vim-operator-sort
46. vim-qfreplace
47. quickfixstatus
48. vim-hier
49. qfixhowm
50. vim-fontzoom
51. vim-indent-guides
52. MultipleSearch
53. vim-easymotion
54. matchparenpp
55. matchit.zip
56. vim-surround
57. vim-textmanip
58. tcomment_vim
59. DrawIt
60. RST-Tables
61. sequence
62. vim-visualstar
63. occur.vim
64. ideone-vim
65. project.tar.gz
66. vimproc
67. vinarise
68. vinarise-plugin-peanalysis
69. vimfiler
70. vimshell
71. vim-logcat
72. vim-quickrun
73. vim-prettyprint
74. vim-editvar
75. open-browser.vim
76. splice.vim
77. gundo.vim
78. copypath.vim
79. DirDiff.vim
80. ShowMultiBase
81. ttoc
82. wokmarks.vim
83. vim-ambicmd
84. vim-altercmd
85. tcommand_vim
86. headlights
87. a.vim
88. c.vim
89. CCTree
90. Source-Explorer-srcexpl.vim
91. trinity.vim
92. cscope-menu
93. gtags.vim
94. DoxygenToolkit.vim
95. pytest.vim
96. python-mode
97. perl-support.vim
98. vim-javascript
99. vim-filetype-haskell
100. haskellmode-vim
101. vim-syntax-haskell-cabal
102. ghcmod-vim
103. vimclojure
104. csv.vim
105. Color-Sampler-Pack
106. webapi-vim
107. cecutil
108. tlib

Job change to programmer

- I thought that I would like to use Vim even more, such as writing code
    - However, as there was no programming experience, it was often fail the screening process
    - In a company where I was employed, I talked about Vim at the interview
        - It may have been a positive?
- Coding environment changed from Windows to Linux
    - I did not have any trouble as Vim and plugins could be used
- I got more and more crazy about Vim
    - Tried to use my customized vim on every host
    - Make recommended vimrc for colleague

A few years later, decided to migrate

- Improve that depending too much on plugins
  - There is also an influence by Spartan Vim
- Need to change my main plugins
  - Because *neo* series stopped active development
    - Use dark powered plugins, or
    - Use other plugins, or
    - Do not use plugins
      - Replacing with built-in command or a few lines of Vim script

De-neocomplete(Auto-completion)

Requirements:

- Some kind of completion
  - Don't care about it manually
  - Don't care much about speed
- Almost the same behavior as neocomplete

Insert mode completion + `completeopt=menuone,longest,preview`

Note: The default is `menu,preview`

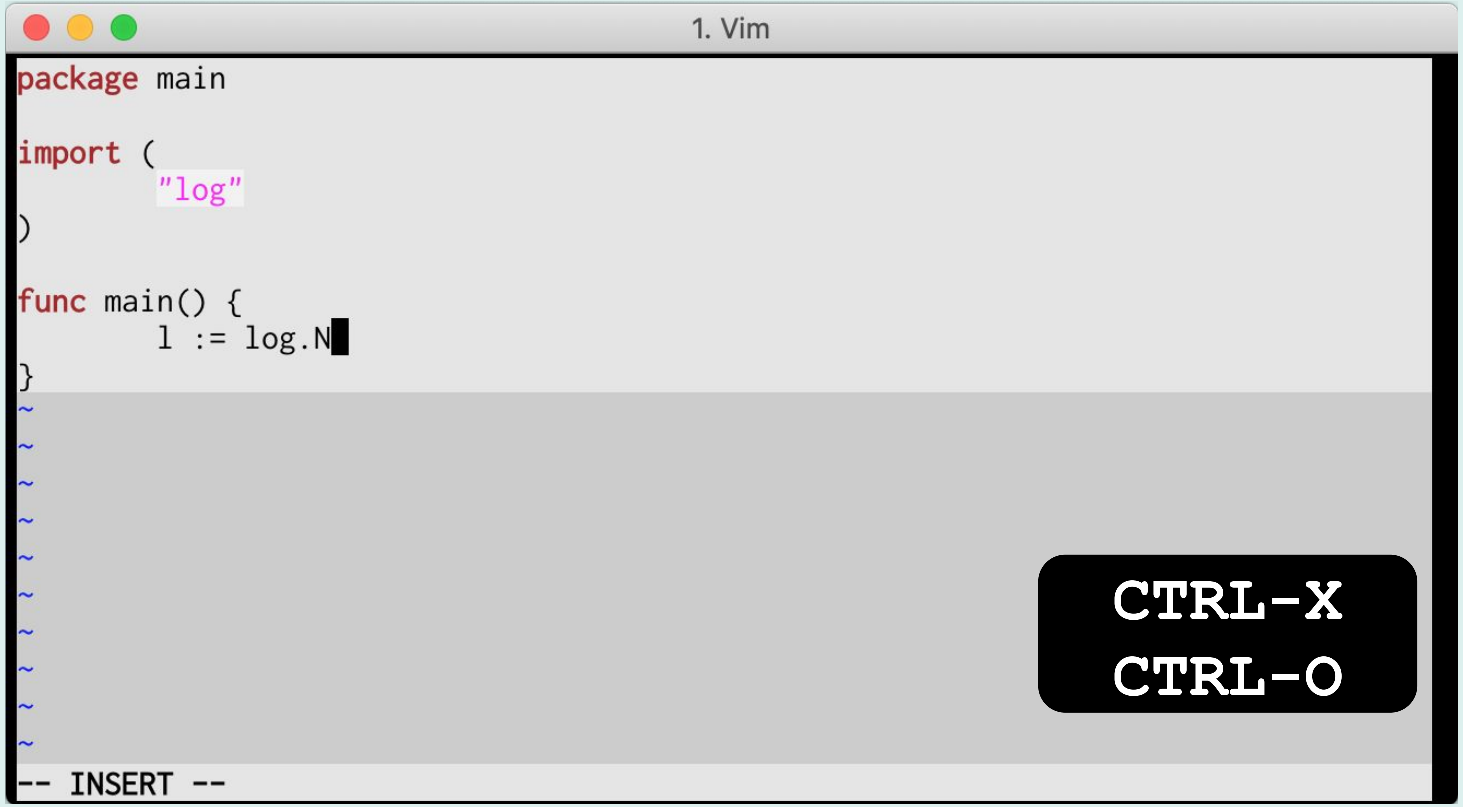
List of completions

Key	Completion
CTRL-X CTRL-L	whole lines
CTRL-X CTRL-N or CTRL-X CTRL-P	keywords in the current file
CTRL-X CTRL-K	keywords in 'dictionary'
CTRL-X CTRL-T	keywords in 'thesaurus'
CTRL-X CTRL-I	keywords in the current and included files
CTRL-X CTRL-]	tags

Key	Completion
CTRL-X CTRL-F	file names
CTRL-X CTRL-D	definitions or macros
CTRL-X CTRL-V	Vim command-line
CTRL-X CTRL-U	User defined completion
CTRL-X CTRL-O	omni completion
CTRL-X s	Spelling suggestions
CTRL-N or CTRL-P	keywords in 'complete'

See `:help ins-completions` for details

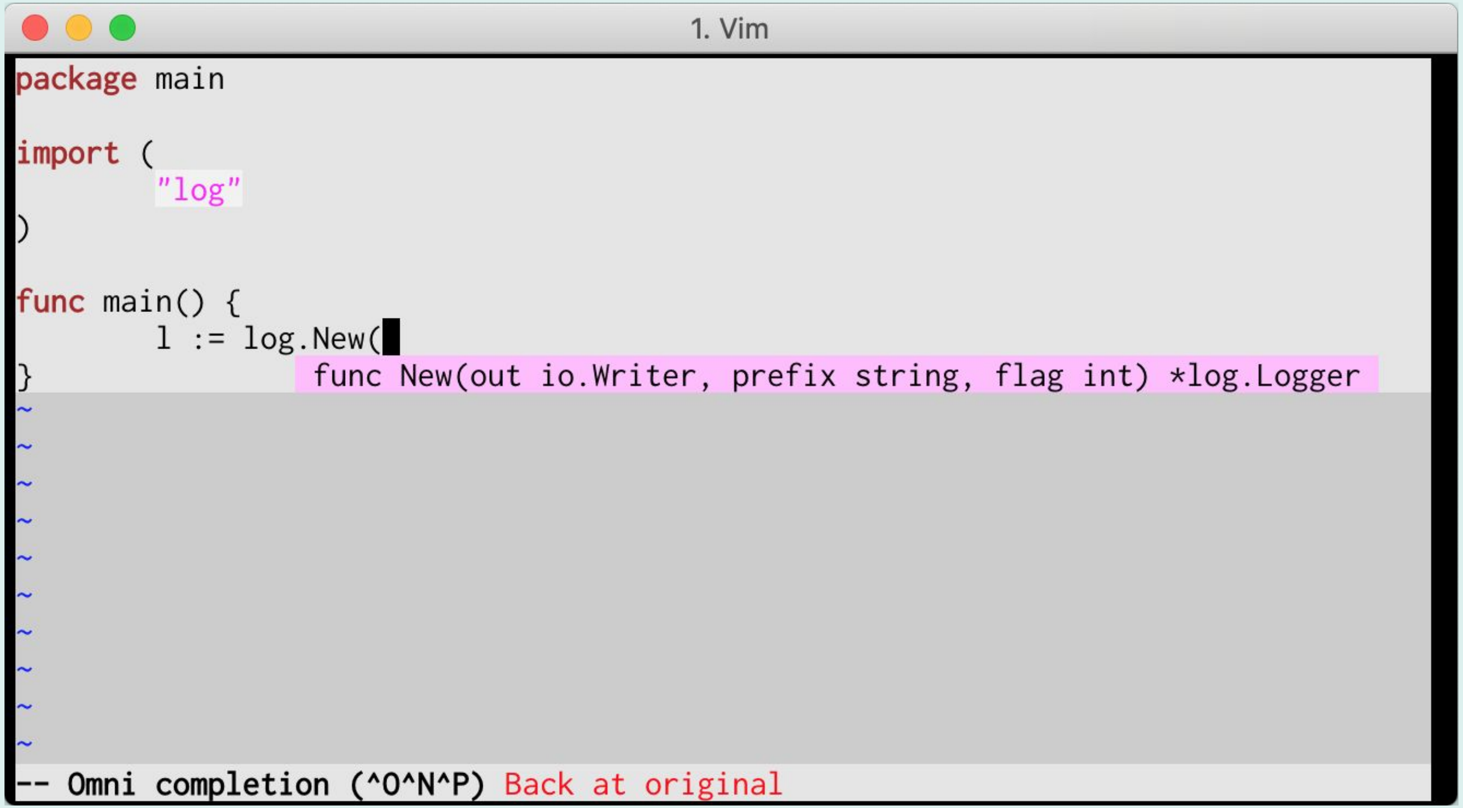
Omni completion



```
package main

import (
        "log"
)

func main() {
        l := log.N
}
~
~
~
~
~
~
~
~
~
~
-- INSERT --
```

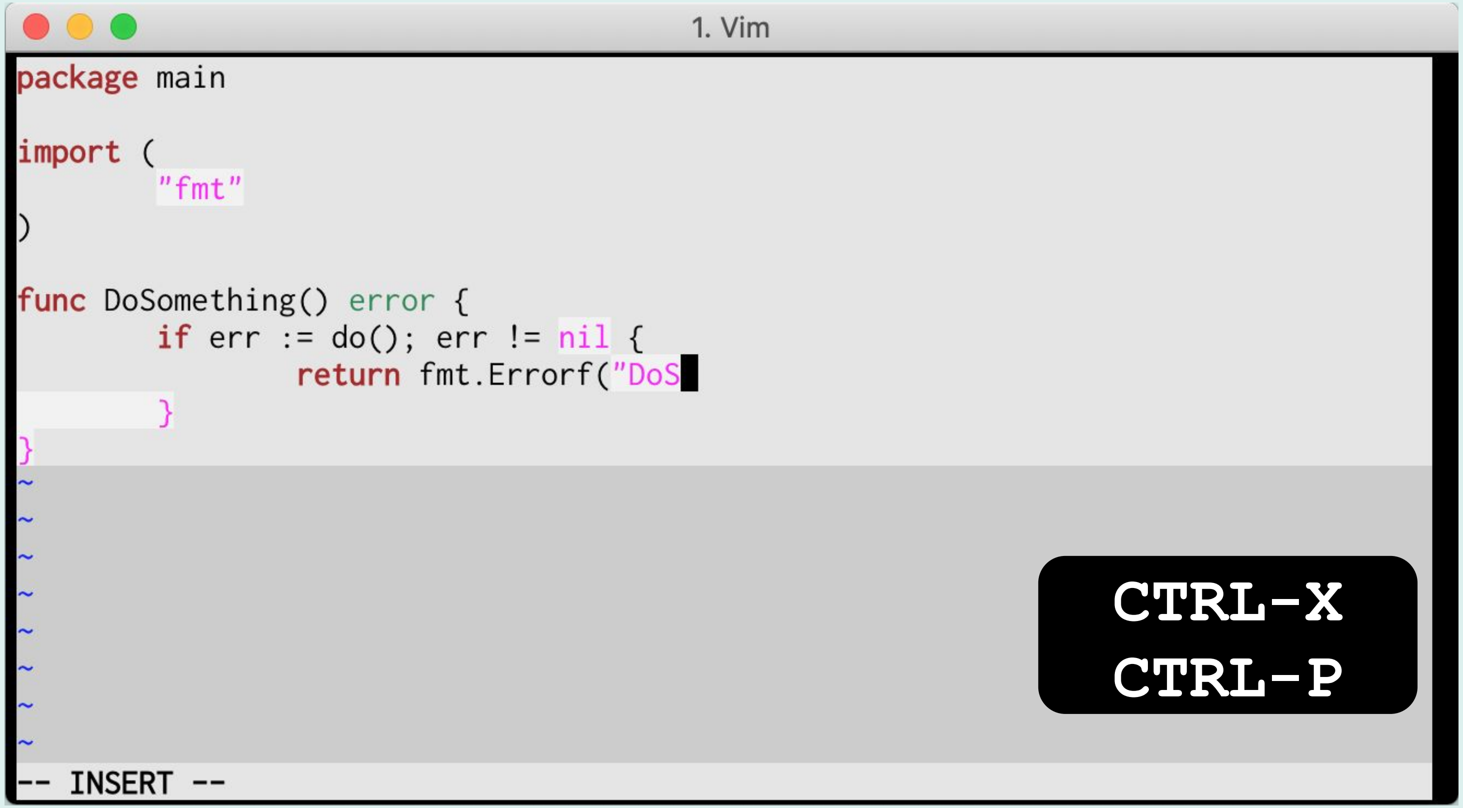
CTRL-X
CTRL-O

Omni completion


1. Vim

```
package main

import (
        "log"
)

func main() {
        l := log.New(
}                    func New(out io.Writer, prefix string, flag int) *log.Logger
~
~
~
~
~
~
~
~
~
~
-- Omni completion (^O^N^P) Back at original
```

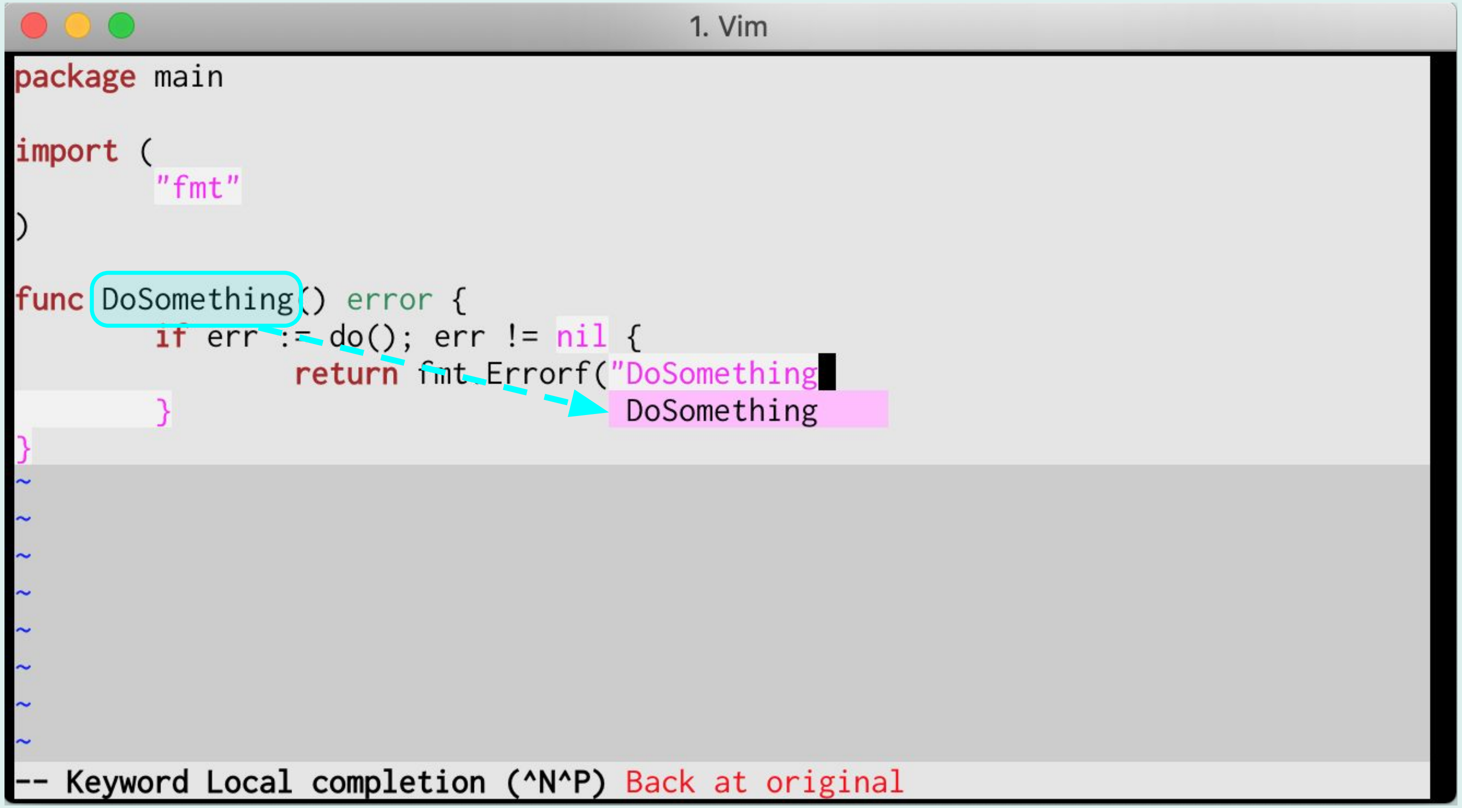
Completing keywords in current file


1. Vim

```
package main

import (
        "fmt"
)

func DoSomething() error {
        if err := do(); err != nil {
                return fmt.Errorf("DoS
        }
}
~
~
~
~
~
~
~
~
-- INSERT --
```

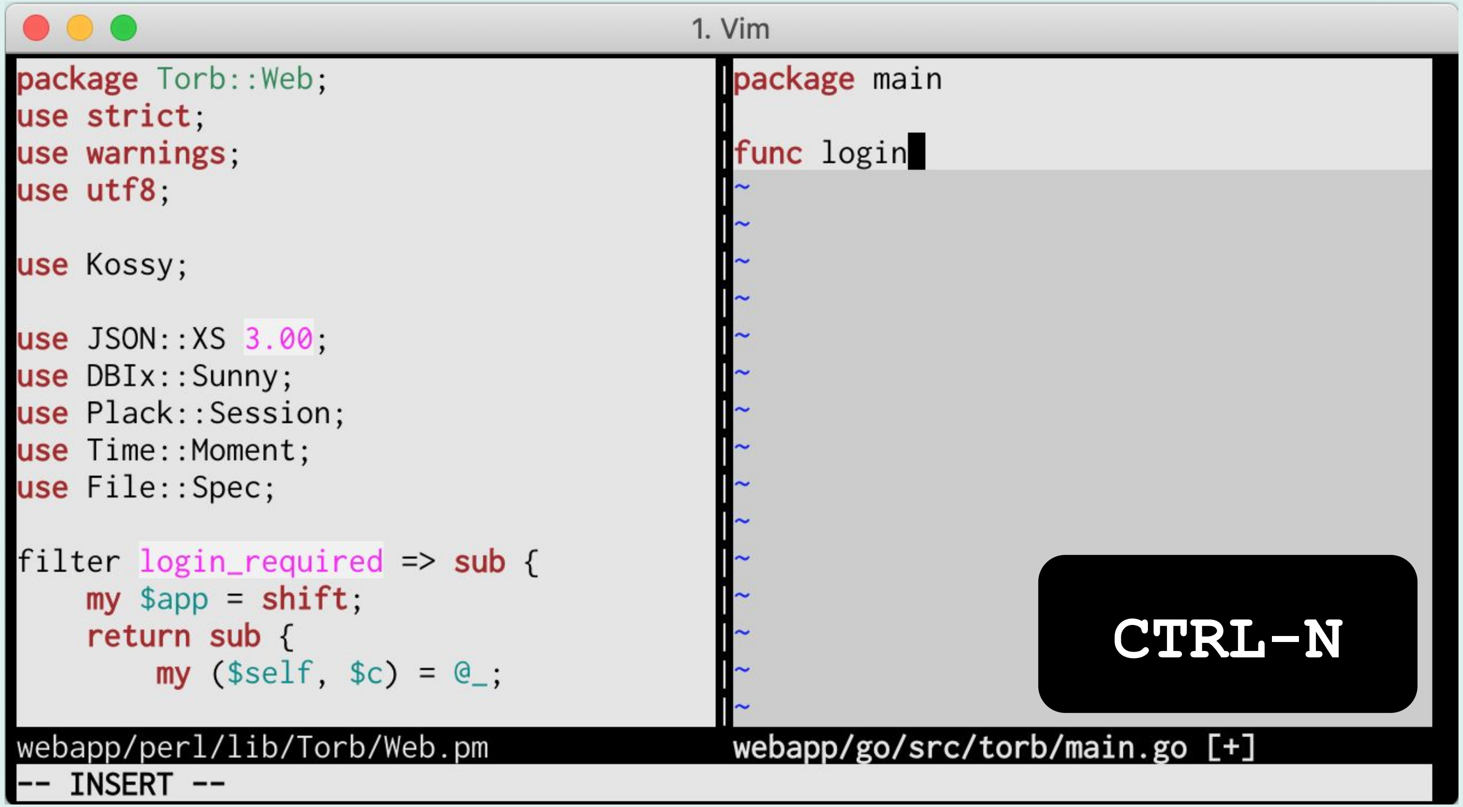
CTRL-X
CTRL-P

Completing keywords in current file



```
package main

import (
        "fmt"
)

func DoSomething() error {
        if err := do(); err != nil {
                return fmt.Errorf("DoSomething
                                  DoSomething
        }
}
~
~
~
~
~
~
~
~
-- Keyword Local completion (^N^P) Back at original
```

Completing keywords from different sources



1. Vim

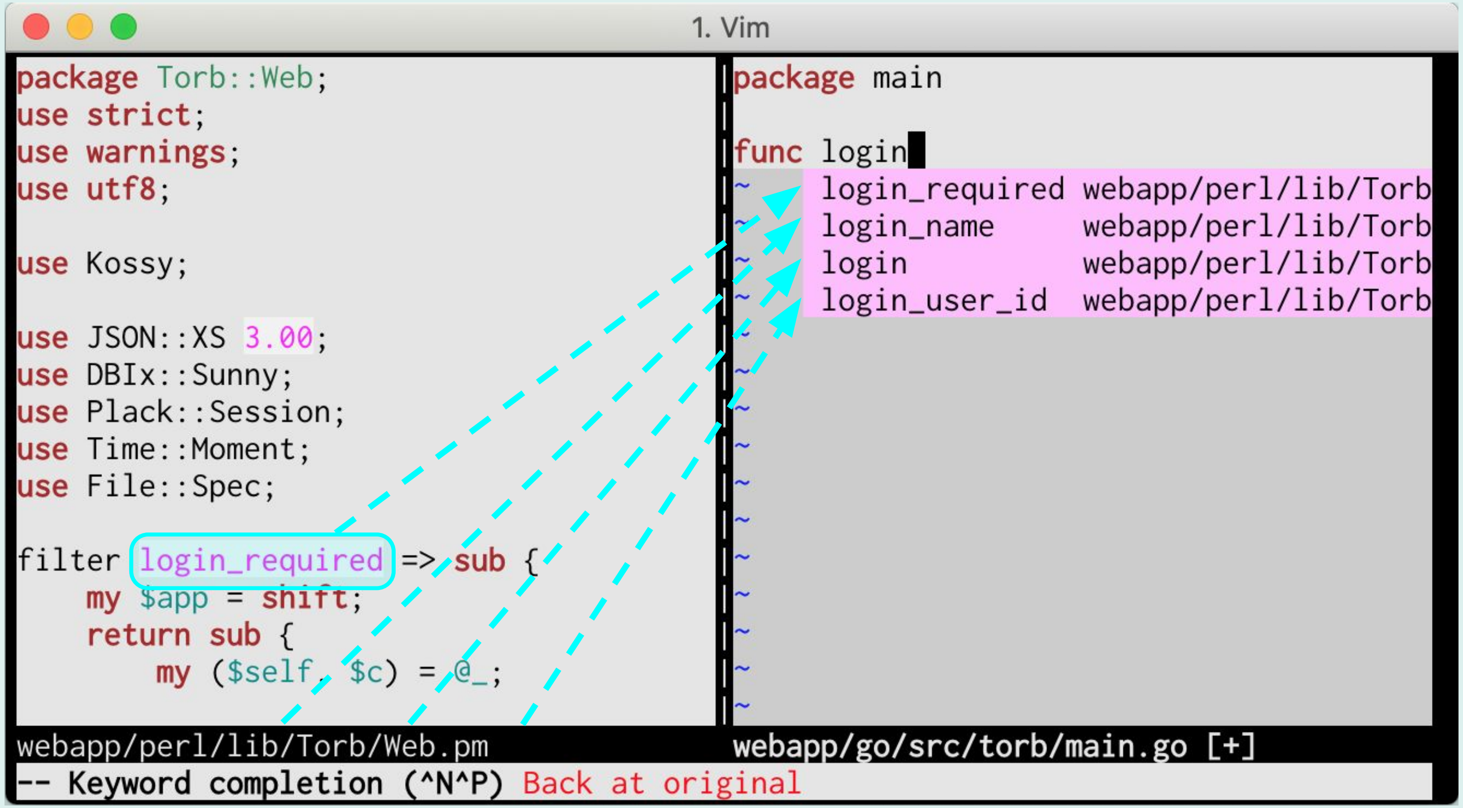
```
package Torb::Web;
use strict;
use warnings;
use utf8;

use Kossy;

use JSON::XS 3.00;
use DBIx::Sunny;
use Plack::Session;
use Time::Moment;
use File::Spec;

filter login_required => sub {
    my $app = shift;
    return sub {
        my ($self, $c) = @_;
```

```
package main

func login
~
```

```
webapp/perl/lib/Torb/Web.pm            webapp/go/src/torb/main.go [+]
-- INSERT --
```

CTRL-N

Completing keywords from different sources

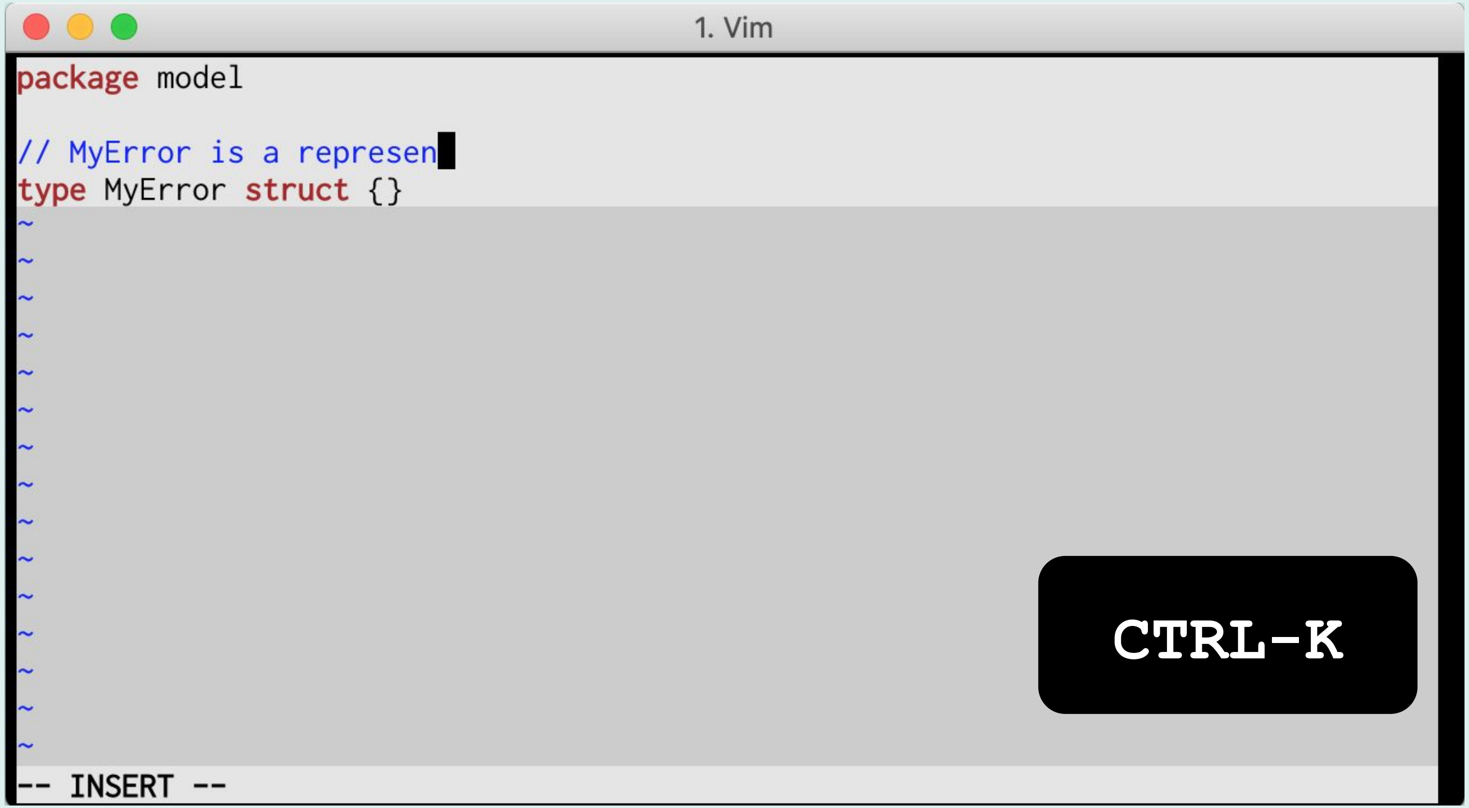


```
package Torb::Web;
use strict;
use warnings;
use utf8;

use Kossy;

use JSON::XS 3.00;
use DBIx::Sunny;
use Plack::Session;
use Time::Moment;
use File::Spec;

filter login_required => sub {
    my $app = shift;
    return sub {
        my ($self, $c) = @_;
webapp/perl/lib/Torb/Web.pm
```

```
package main

func login
     login_required webapp/perl/lib/Torb
     login_name     webapp/perl/lib/Torb
     login          webapp/perl/lib/Torb
     login_user_id  webapp/perl/lib/Torb
webapp/go/src/torb/main.go [+]
```

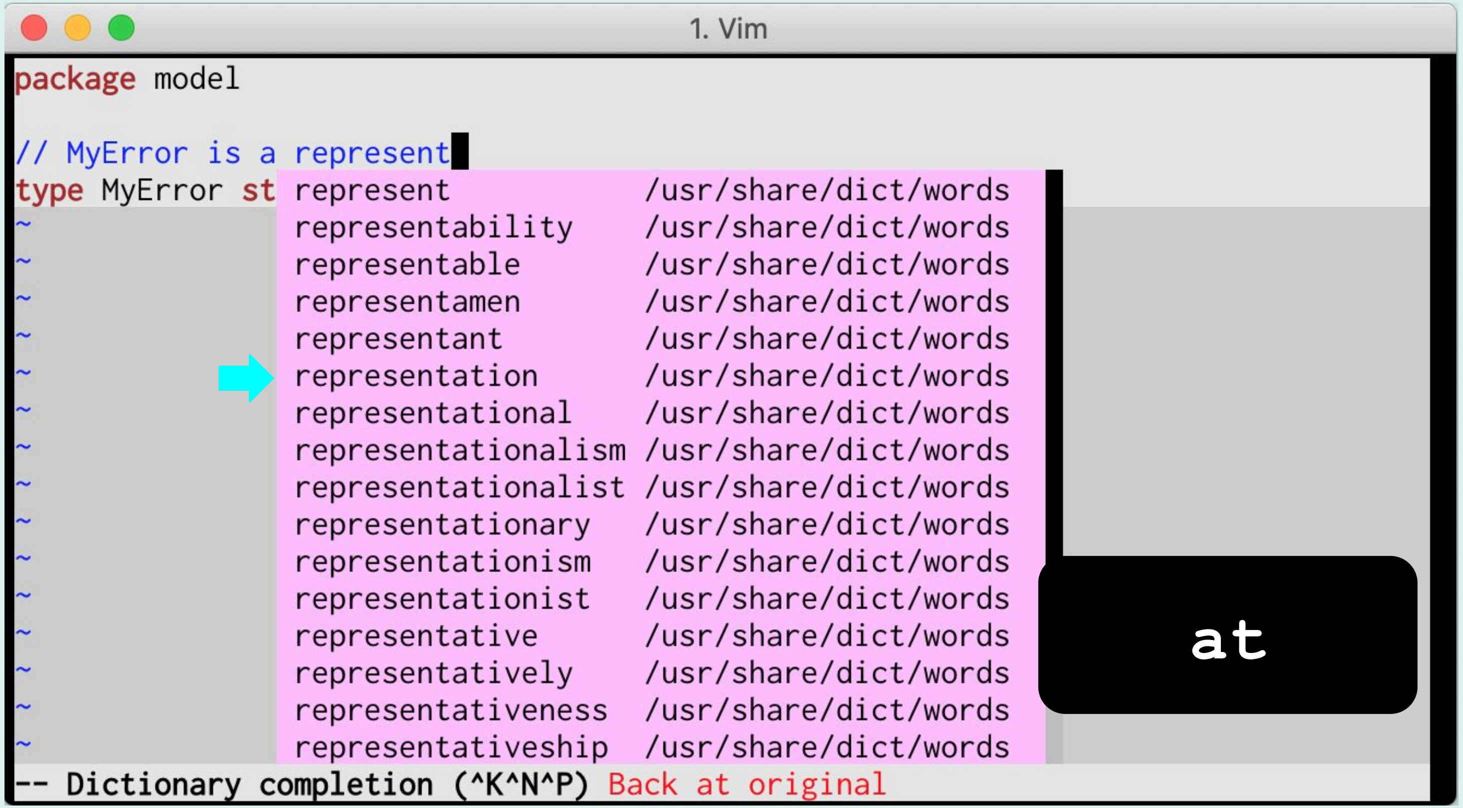
```
-- Keyword completion (^N^P) Back at original
```

Completing keywords in 'dictionary'



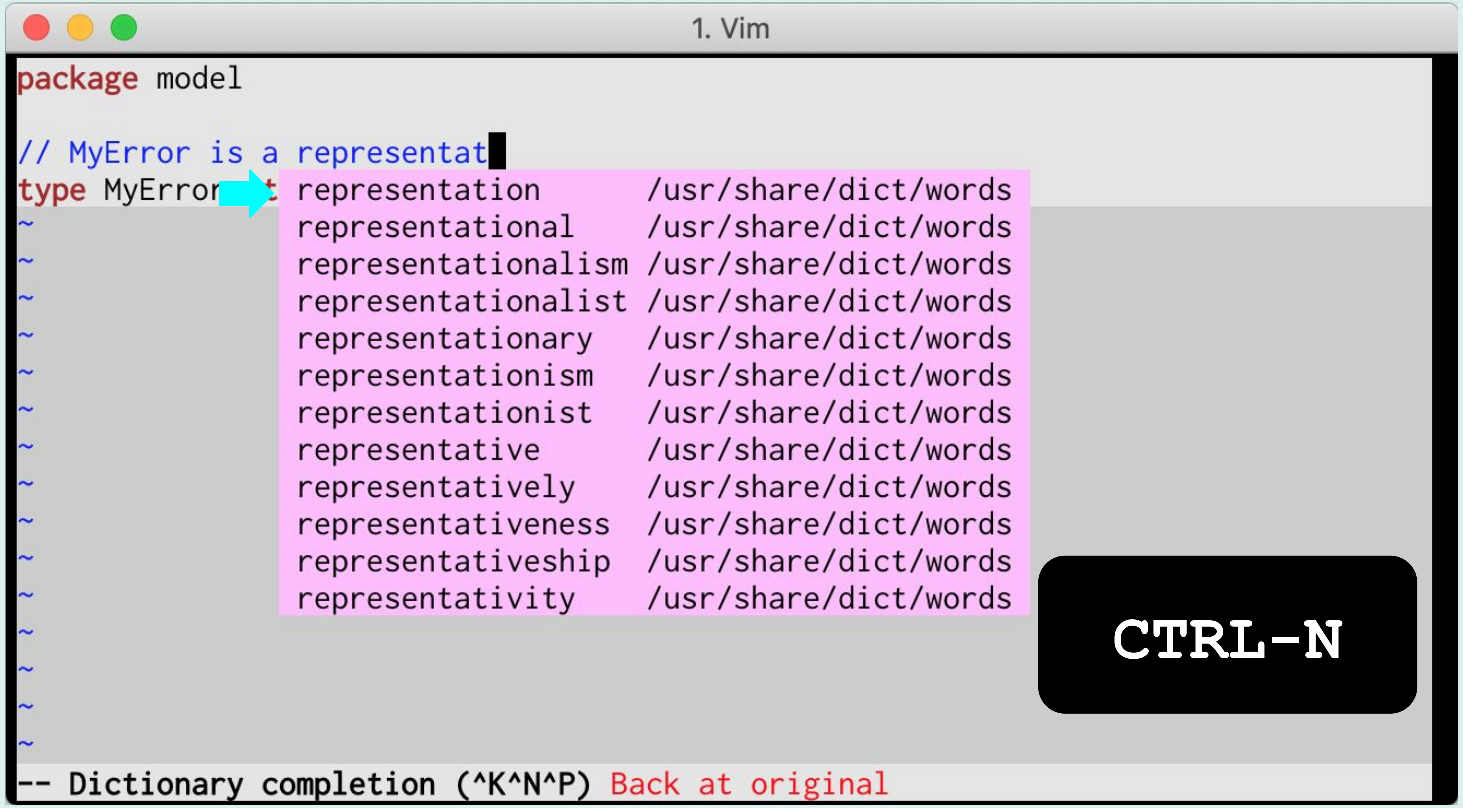
```
package model

// MyError is a represen
type MyError struct {}
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --
```

Completing keywords in 'dictionary'



```
package model

// MyError is a represent
type MyError st represent             /usr/share/dict/words
~               representability      /usr/share/dict/words
~               representable         /usr/share/dict/words
~               representamen         /usr/share/dict/words
~               representant          /usr/share/dict/words
~               representation        /usr/share/dict/words
~               representational      /usr/share/dict/words
~               representationalism   /usr/share/dict/words
~               representationalist   /usr/share/dict/words
~               representationary     /usr/share/dict/words
~               representationism     /usr/share/dict/words
~               representationist     /usr/share/dict/words
~               representative        /usr/share/dict/words
~               representatively      /usr/share/dict/words
~               representativeness    /usr/share/dict/words
~               representativeship    /usr/share/dict/words
-- Dictionary completion (^K^N^P) Back at original
```

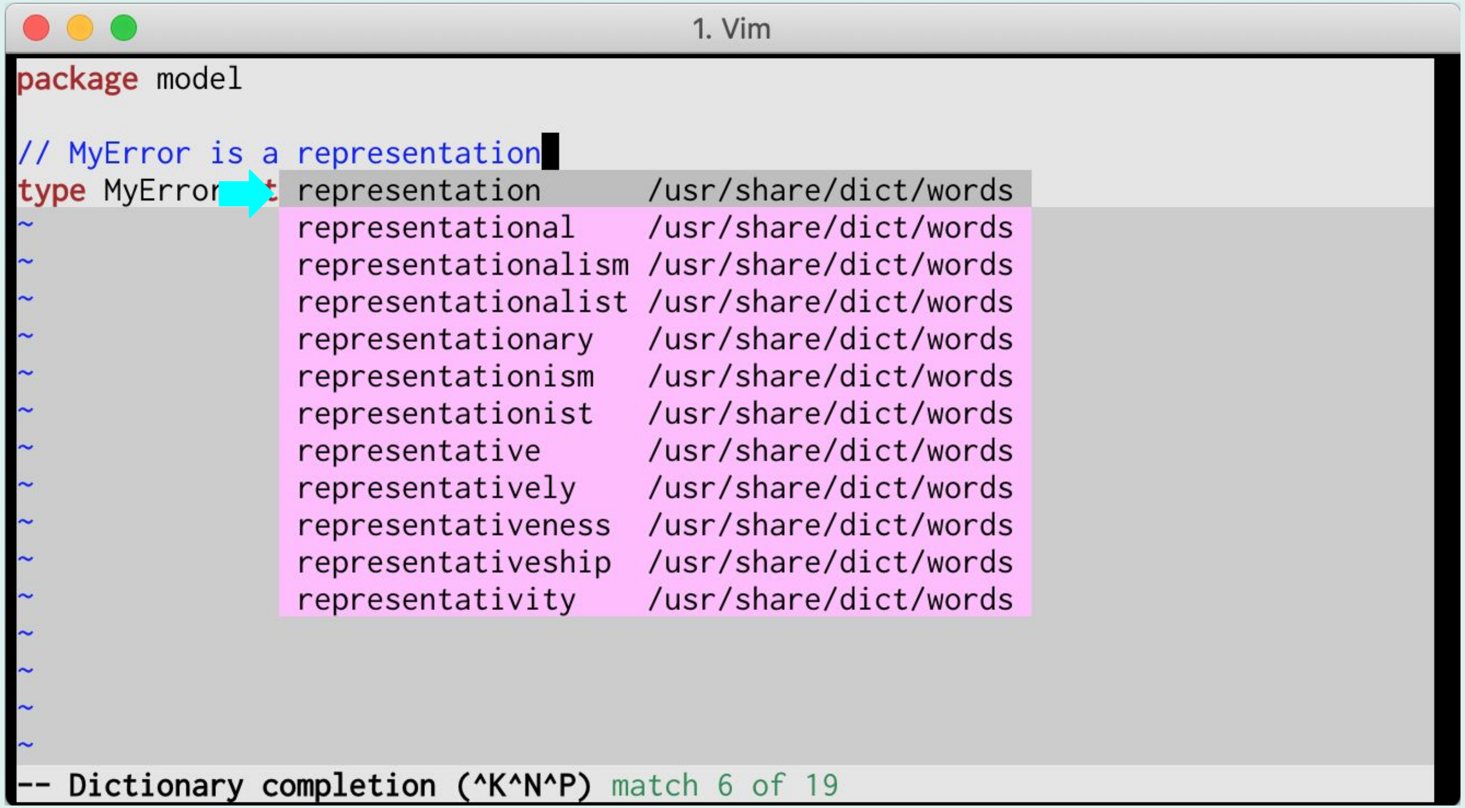
Completing keywords in 'dictionary'



```
package model

// MyError is a representat
type MyError    representation         /usr/share/dict/words
                representational       /usr/share/dict/words
                representationalism    /usr/share/dict/words
                representationalist    /usr/share/dict/words
                representationary      /usr/share/dict/words
                representationism      /usr/share/dict/words
                representationist      /usr/share/dict/words
                representative         /usr/share/dict/words
                representatively       /usr/share/dict/words
                representativeness     /usr/share/dict/words
                representativeship     /usr/share/dict/words
                representativity       /usr/share/dict/words
~
~
~
-- Dictionary completion (^K^N^P) Back at original
```

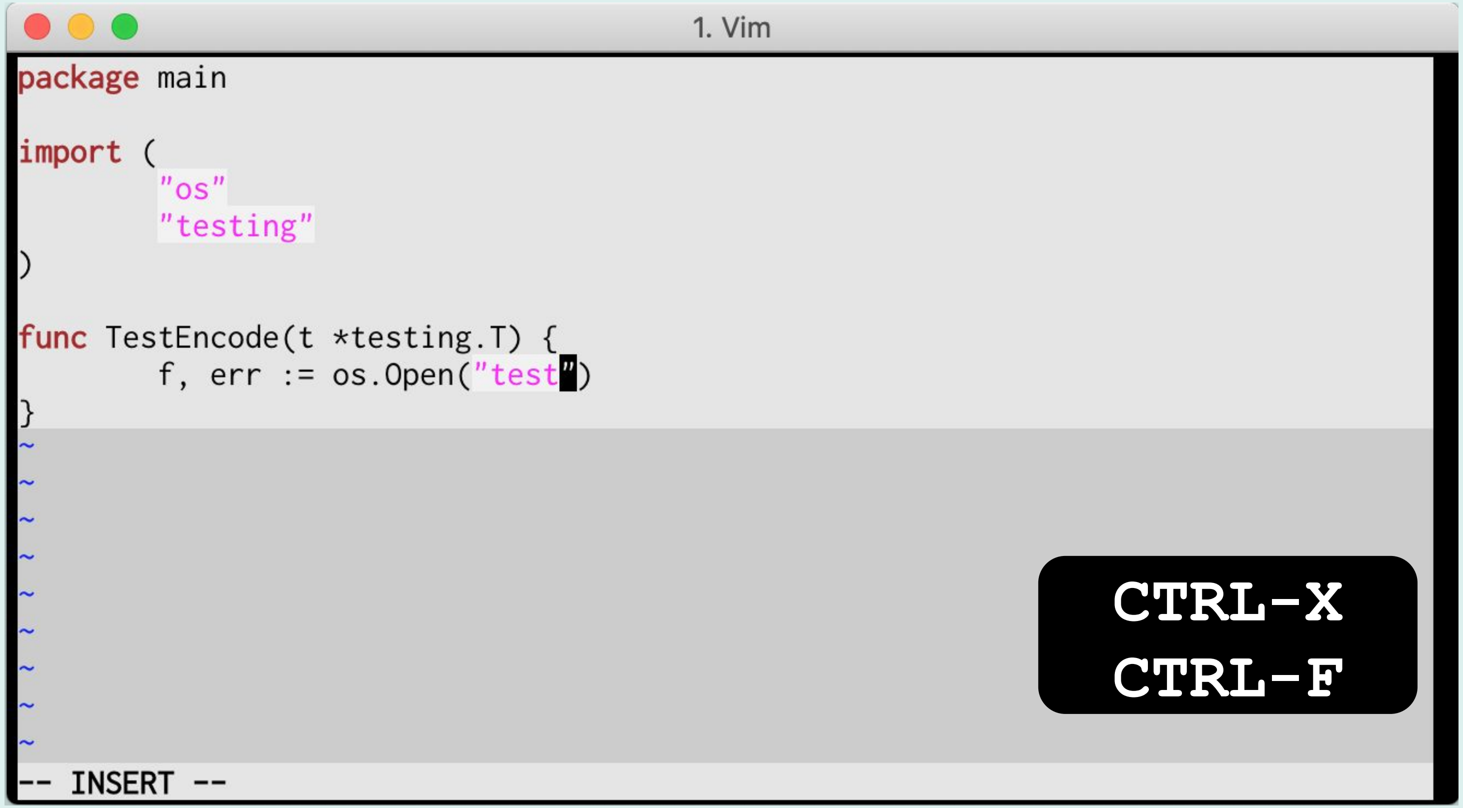
CTRL-N

Completing keywords in 'dictionary'



```
package model

// MyError is a representation
type MyError   representation          /usr/share/dict/words
~              representational        /usr/share/dict/words
~              representationalism     /usr/share/dict/words
~              representationalist     /usr/share/dict/words
~              representationary       /usr/share/dict/words
~              representationism       /usr/share/dict/words
~              representationist       /usr/share/dict/words
~              representative          /usr/share/dict/words
~              representatively        /usr/share/dict/words
~              representativeness      /usr/share/dict/words
~              representativeship      /usr/share/dict/words
~              representativity        /usr/share/dict/words
~
~
~
~
-- Dictionary completion (^K^N^P) match 6 of 19
```

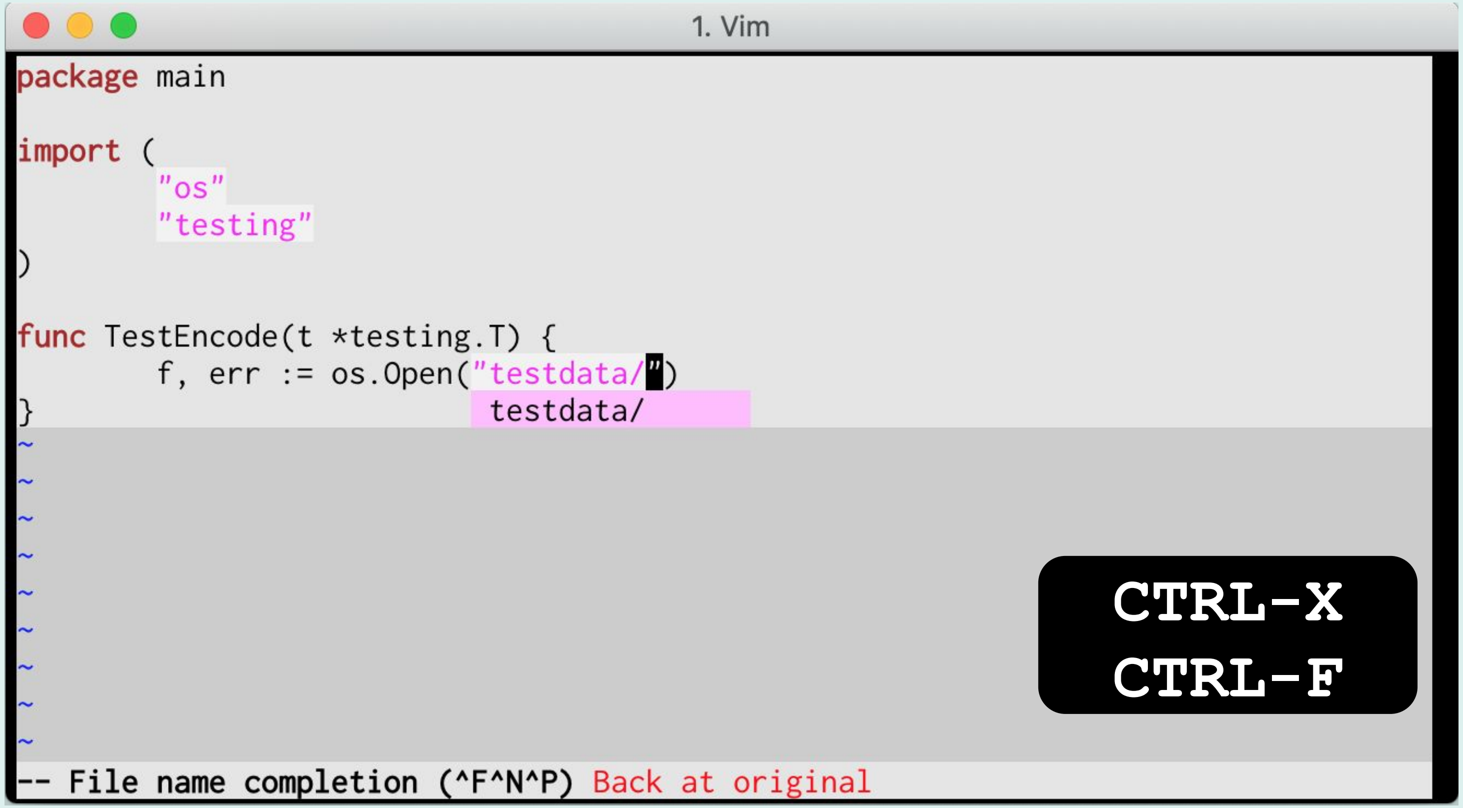
Completing file names


1. Vim

```
package main

import (
        "os"
        "testing"
)

func TestEncode(t *testing.T) {
        f, err := os.Open("test")
}
~
~
~
~
~
~
~
~
~
-- INSERT --
```

CTRL-X
CTRL-F

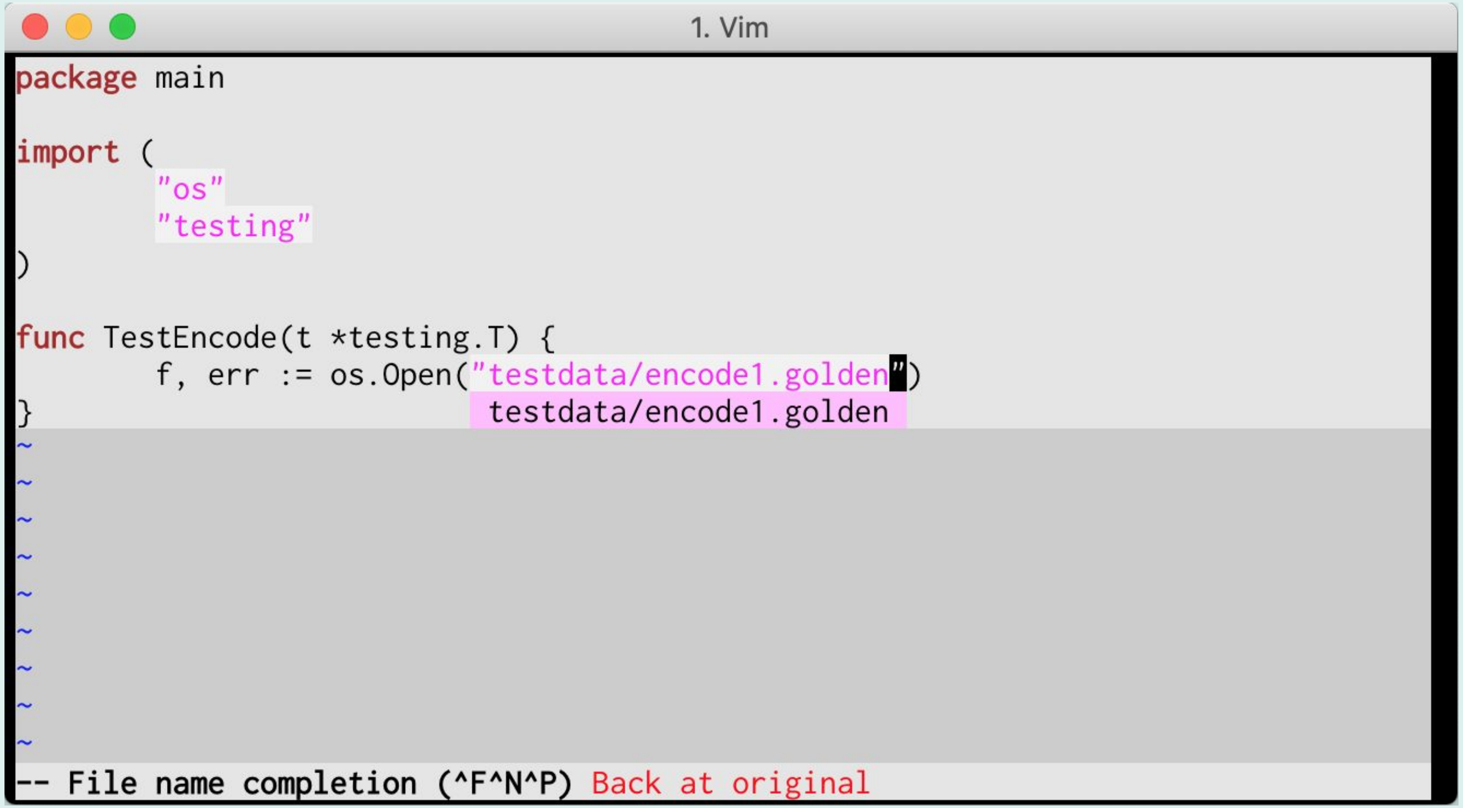
Completing file names



```
package main

import (
        "os"
        "testing"
)

func TestEncode(t *testing.T) {
        f, err := os.Open("testdata/")
}                         testdata/
~
~
~
~
~
~
~
~
~
-- File name completion (^F^N^P) Back at original
```

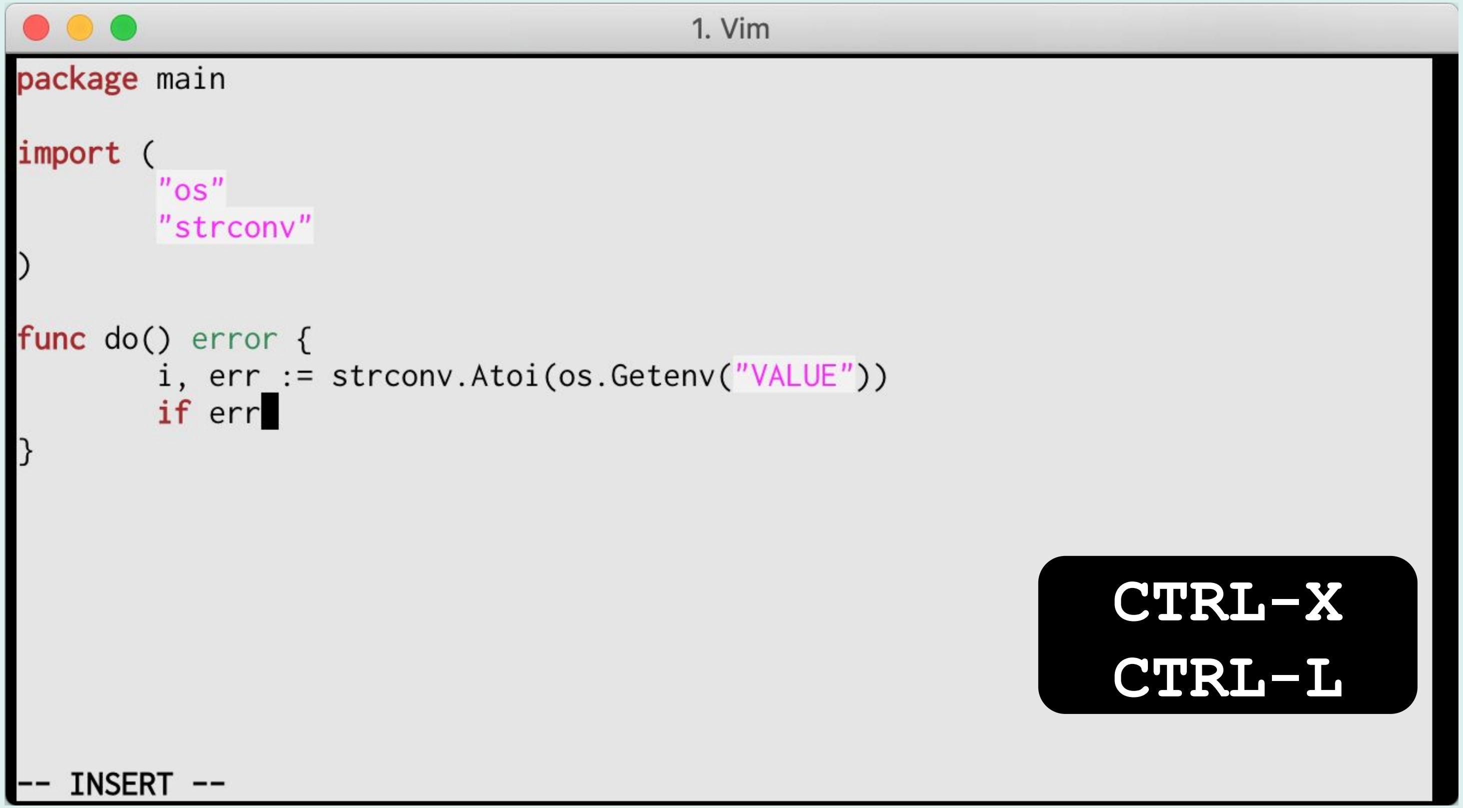
CTRL-X
CTRL-F

Completing file names


1. Vim

```
package main

import (
        "os"
        "testing"
)

func TestEncode(t *testing.T) {
        f, err := os.Open("testdata/encode1.golden")
}                         testdata/encode1.golden
~
~
~
~
~
~
~
~
~
-- File name completion (^F^N^P) Back at original
```

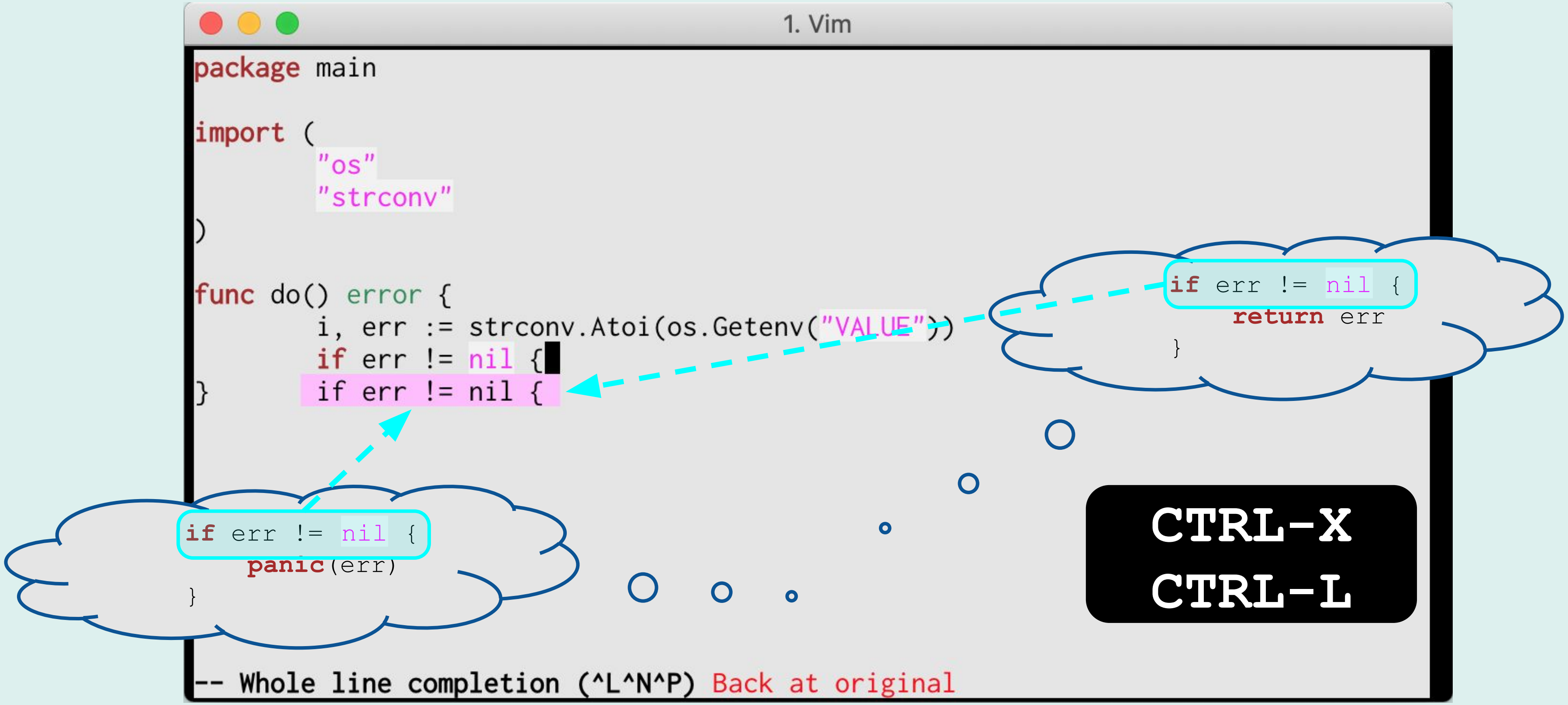
Completing whole lines



```
package main

import (
        "os"
        "strconv"
)

func do() error {
        i, err := strconv.Atoi(os.Getenv("VALUE"))
        if err
}

-- INSERT --
```

CTRL-X
CTRL-L

Completing whole lines



```
package main

import (
        "os"
        "strconv"
)

func do() error {
        i, err := strconv.Atoi(os.Getenv("VALUE"))
        if err != nil {
}       if err != nil {
```

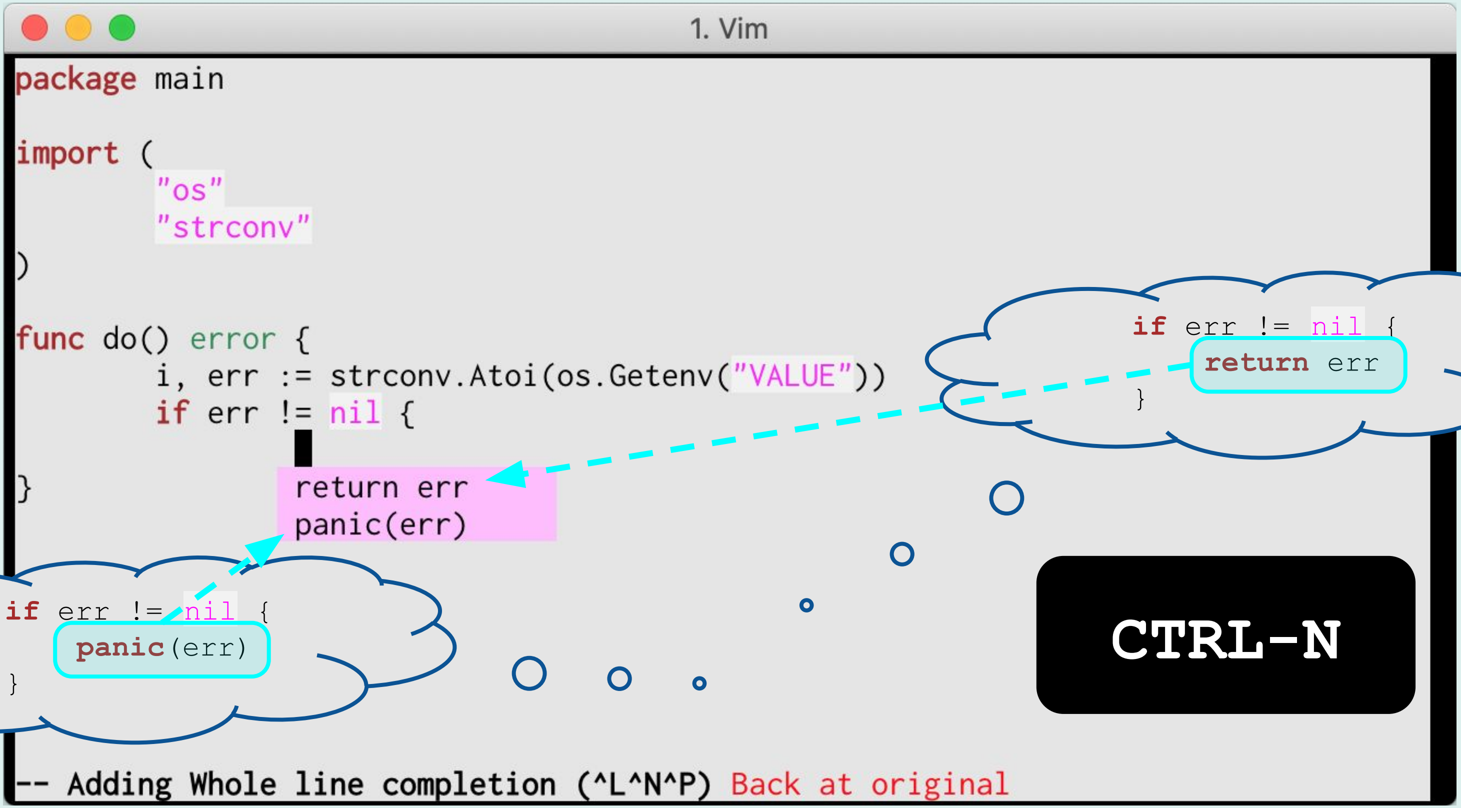
```
if err != nil {
        return err
}
```

```
if err != nil {
        panic(err)
}
```

```
-- Whole line completion (^L^N^P) Back at original
```

CTRL-X CTRL-L

Completing whole lines



```
package main

import (
        "os"
        "strconv"
)

func do() error {
        i, err := strconv.Atoi(os.Getenv("VALUE"))
        if err != nil {

}
```

```
return err
panic(err)
```

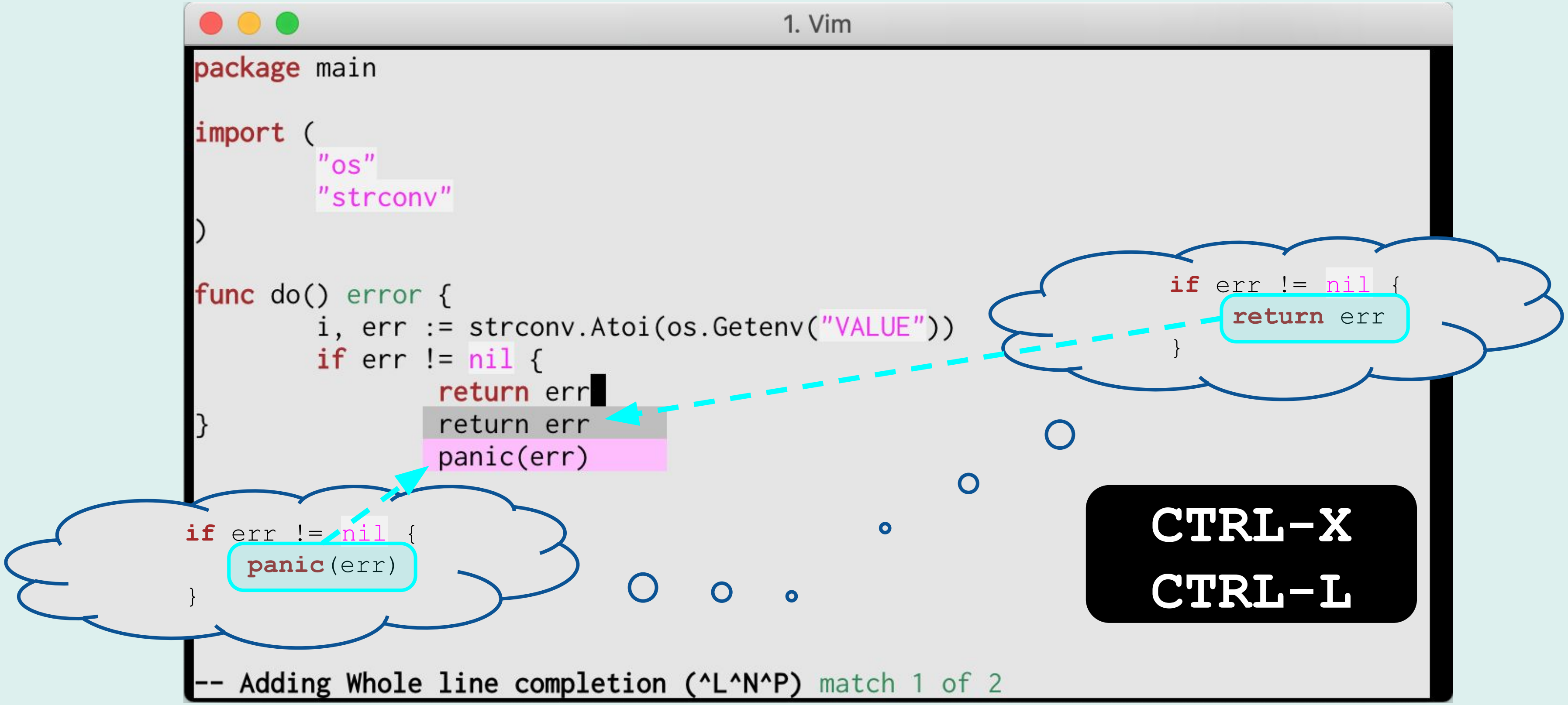
```
if err != nil {
    return err
}
```

```
if err != nil {
    panic(err)
}
```

CTRL-N

```
-- Adding Whole line completion (^L^N^P) Back at original
```

Completing whole lines



```
package main

import (
        "os"
        "strconv"
)

func do() error {
        i, err := strconv.Atoi(os.Getenv("VALUE"))
        if err != nil {
                return err
}               return err
                panic(err)
```

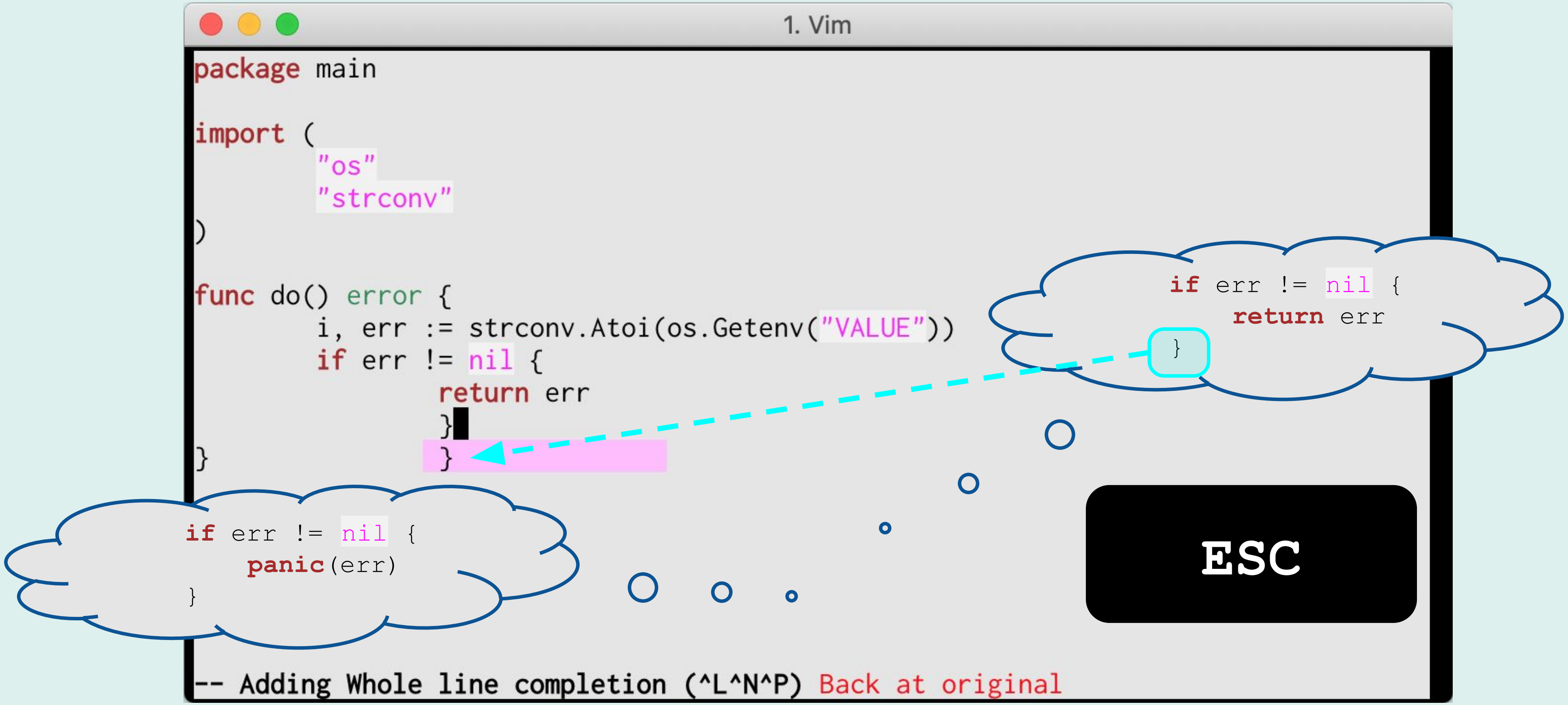
```
if err != nil {
        return err
}
```

```
if err != nil {
        panic(err)
}
```

CTRL-X
CTRL-L

```
-- Adding Whole line completion (^L^N^P) match 1 of 2
```

Completing whole lines



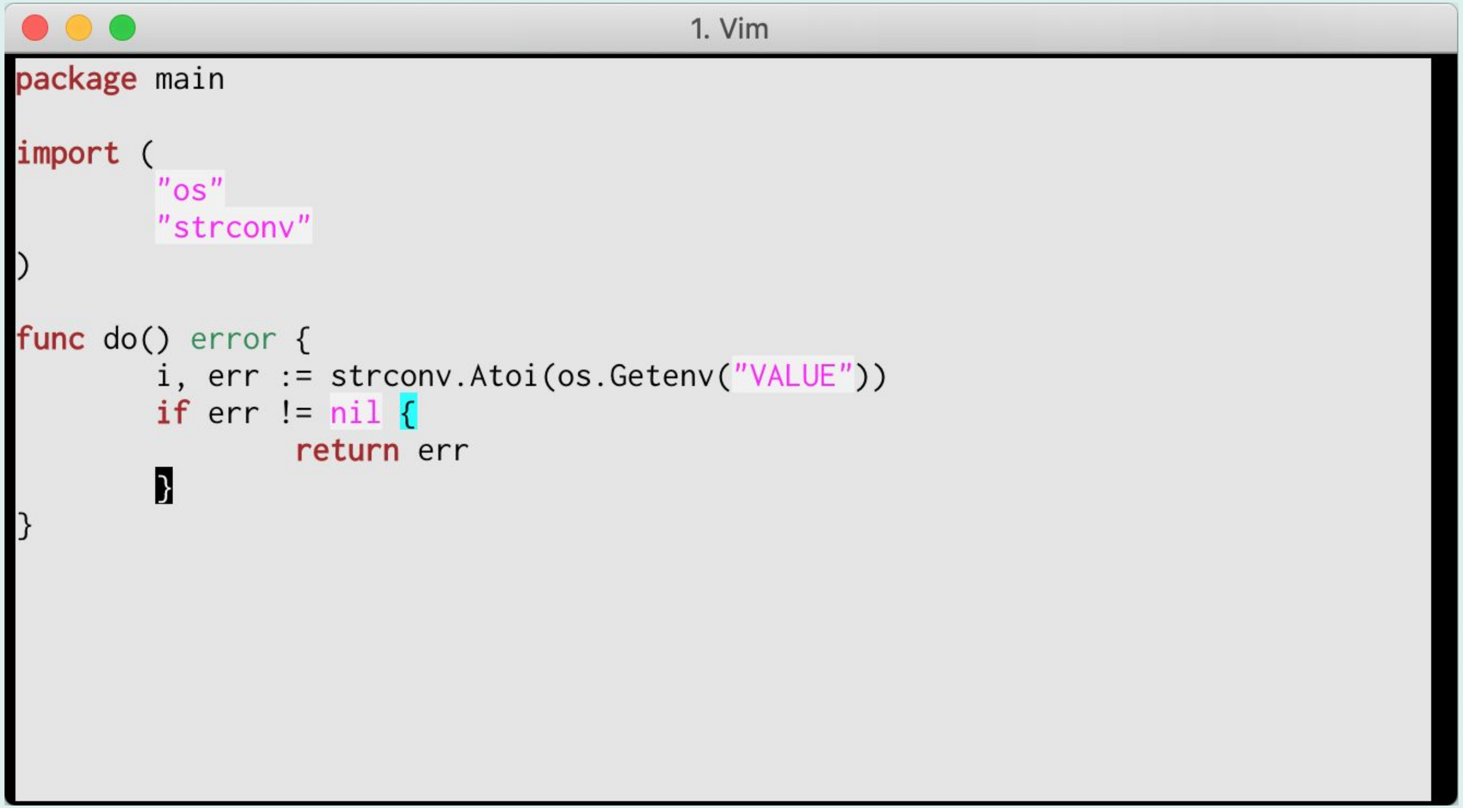
```
package main

import (
        "os"
        "strconv"
)

func do() error {
        i, err := strconv.Atoi(os.Getenv("VALUE"))
        if err != nil {
                return err
                }
                }
}
```

```
if err != nil {
        return err
}
```

```
if err != nil {
        panic(err)
}
```

```
-- Adding Whole line completion (^L^N^P) Back at original
```

Completing whole lines


1. Vim

```
package main

import (
	"os"
	"strconv"
)

func do() error {
	i, err := strconv.Atoi(os.Getenv("VALUE"))
	if err != nil {
		return err
	}
}
```

De-neobundle(Plugin manager)

Requirements:

- Load plugins
- Install plugins
- Update plugins
- Lazy loading
    - Something for faster startup

⬇

Packages + `system()`, job, timer

Packages

- Load plugins from "pack/*/start" under `packpath` automatically
  - $HOME/.vim/pack/bundle/start/*
  - $VIM/vimfiles/pack/*/start/*
  - $VIMRUNTIME/pack/dist/start/*
  - etc.
- Load plugins from "pack/*/opt" under `packpath` with `:packadd {name}`
  - $HOME/.vim/pack/bundle/opt/*
  - $VIM/vimfiles/pack/*/opt/*
  - $VIMRUNTIME/pack/dist/opt/*
  - etc.
- There is no feature to install or update plugins

Install plugins

- Shell command

```
git clone <url> ~/.vim/pack/bundle/opt/<plugin name>
```

- Vim script

```
let s:plugins = []
call add(s:plugins, 'https://github.com/vim-jp/vimdoc-ja')

function! InstallPlugins()
    for url in s:plugins
        let dst = expand('~/.vim/pack/bundle/opt/' . split(url, '/')[-1])
        if !isdirectory(dst)
            call system(printf('git clone %s %s', url, dst))
        endif
    endfor
endfunction
```

Update plugins

- Shell command

```
ls -d ~/.vim/pack/bundle/opt/* | xargs -I{} git -C {} pull --ff --ff-only
```

- Vim script

```
function! UpdatePlugins()
    split `='[update plugins]'` | setlocal buftype=nofile
    let s:idx = 0
    call timer_start(100, 'UpdateHandler', {'repeat': len(s:plugins)})
endfunction

function! UpdateHandler(timer)
    let dst = expand('~/.vim/pack/bundle/opt/' . split(s:plugins[s:idx], '/')[-1])
    let cmd = printf('git -C %s pull --ff --ff-only', dst)
    call job_start(cmd, {'out_io': 'buffer', 'out_name': '[update plugins]'})
    let s:idx += 1
endfunction
```

Lazy loading

- With autocmd

```
" filetype
autocmd FileType go call LoadGoPlugins()
function! LoadGoPlugins()
    packadd vim-go
endfunction

" command
autocmd CmdUndefined Template packadd sonictemplate-vim
```

Background loading

- With timer

```
if has('vim_starting')
  autocmd VimEnter * call timer_start(1, 'LoadHandler', {'repeat': len(s:plugins)})
endif

let s:idx = 0
function! LoadHandler(timer)
  execute 'packadd ' . split(s:plugins[s:idx], '/')[-1]
  let s:idx += 1
endfunction
```

De-unite(File finder)

Requirements:

- Listing and opening files
  - Under current or buffer path
  - MRU
  - Loaded plugins
- The following are not mandatory
  - Other sources
  - Incremental filtering

Make a candidate + Command to create buffer + Key operation to open file

Make a candidate

- Shell command

```
find <path> -type f
        or
dir <path> /b /s /a-d
```

- Variables
  - `v:oldfiles` for MRU
- EX commands
  - `:scriptnames` for loaded plugins

Command to create buffer

- :r[ead]!, setline()

```vim
" Configure the buffer to be created
command! -bar ToScratch setlocal buftype=nofile bufhidden=hide noswapfile

" List files by :read!
let s:files_cmd = 'find '
let s:files_opts = '-type f'
command! -bar -nargs=1 -complete=dir Files <mods> new | ToScratch |
            \ execute 'read! ' . s:files_cmd . ' "<args>" ' . s:files_opts

" Shorthand of :Files
command! FilesCurrent <mods> Files .
command! FilesBuffer <mods> Files %:p:h

" List files from v:oldfiles excluding unreadable
command! MRU <mods> new | ToScratch |
            \ call setline(1, filter(v:oldfiles, 'filereadable(expand(v:val))' ))

" List sourced scripts from :scriptnames
command! ScriptNames <mods> new | ToScratch |
            \ call setline(1, split(execute('scriptnames'), '\n'))
```

Key operations to open file

- Result of `:FilesBuffer`

```
/Users/daisuzu/work/app-engine-go/app.yaml
/Users/daisuzu/work/app-engine-go/src/app/app.go
/Users/daisuzu/work/app-engine-go/src/app/handler/handler.go
/Users/daisuzu/work/app-engine-go/src/app/model/model.go
```

  - Search by `/{pattern}`
  - Filter by `:g[lobal]/{pattern}/d[elete]` or `:v[global]/{pattern}/d[elete]`
- Normal mode commands

Key	Open the file under the cursor with
gf	current buffer
CTRL-W f or CTRL-W CTRL-F	new window
CTRL-W gf	new tab page

EX commands to open file

Key	Open the file with
:e[dit]	current buffer
:sp[lit]	new window
:vs[plit]	new window(vertical)
:tabe[dit]	new tab page

- wildcards
  - `*`, `**`, etc.
- filename-modifiers
  - `%`, `%:p:h`, `%:p:r`, etc.
- cmdline-completion
  - `<CTRL-D>`, `<Tab>`

De-vimfiler(File manager)

Requirements:

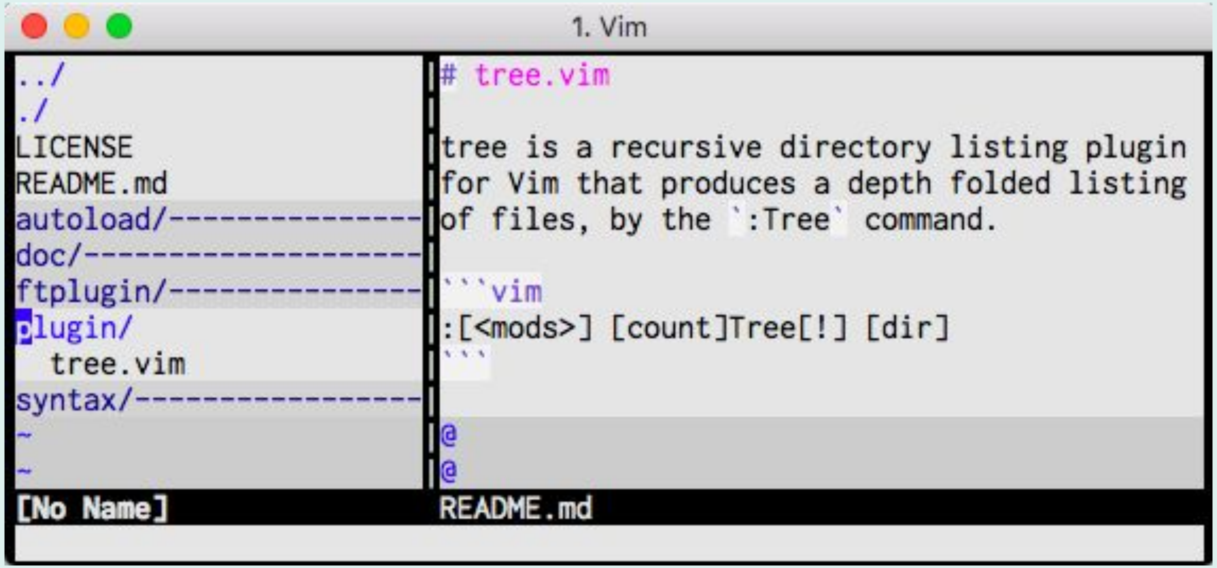
- Display the file tree on the side
  - File operation is not essential

Difficult with a few lines of Vim script

Created a plugin that suitable for my use

tree.vim

- Use result of *tree* command
- Fold subdirectories with `foldmethod=marker`
- Hide absolute path with `conceal`
- No mapping provided



```
../
./
LICENSE
README.md
autoload/---------------
doc/--------------------
ftplugin/---------------
plugin/
  tree.vim
syntax/-----------------
~
~
[No Name]
```

~~~
# tree.vim

tree is a recursive directory listing plugin
for Vim that produces a depth folded listing
of files, by the `:Tree` command.

```vim
:[<mods>] [count]Tree[!] [dir]
```

@
@
README.md
~~~

Summary

- Migrated huge plugins
  - Auto-completion … Insert mode completion
  - Plugin manager … Packages
  - File finder … Combination of commands, functions and key operations
  - File manager … Smaller plugin

- **Not everything can be replaced by standard features**
- **It requires some experience to reduce dependence on plugins**
  - **Similar to learn hjkl, modes and other operations**
- **That efforts made possible to use Vim stably**
  - **Can be used with the same operability in every environment (Unless customized)**
  - **No longer worry about something breaking by updating plugins**