卒業論文

コマンドラインツール作成ライブラリ Thor による hikiutils の書き換え

関西学院大学　理工学部　情報科学科

27013554 山根亮太

2017 年 3 月

指導教員　　西谷　滋人　教授

目次

コマンドラインツール作成ライブラリ Thor による hikiutils の書き換え関西学院大学理工学部情報科学科 27013554 山根亮太

目次

1　概要

本研究では hiki の編集作業をより容易にするためのツールの開発を行った．hiki は通常 web 上で編集を行っているが,GUI と CUI が混在しており, 操作に不便な点がある．そこで, 編集操作が CUI で完結するために開発をされたのが hikiutils である．しかし, そのユーザインタフェースにはコマンドが直感的でないという問題点がある．そこで，Thor というコマンドラインツール作成ライブラリを用いる．optparse というコマンドライン解析ライブラリを使用している hikiutils を新たなコマンドライン解析ライブラリを使用することコマンドを書き換え，より直感的なコマンドにすることが可能である．

2 序論

2.1 目的

2.2 既存システムの背景

2.2.1 hiki

hiki とはプログラミング言語 Ruby を用いられることで作られた wiki クローンの 1 つである．hiki の主な特徴として

- オリジナル wiki に似たシンプルな書式
- プラグインによる機能拡張
- 出力する HTML を柔軟に変更可能
- ページにカテゴリ付けできる
- CSS を使ったテーマ機能
- 携帯端末可能
- InterWiki のサポート
- HikiFarm に対応
- ページの追加，編集がしやすい

等がある

2.2.2 hikiutils

hikiutils は hiki の編集作業を容易に行うことができるよう にするツール群であり, プログラミング言語 Ruby のライブ ラリである gem フォーマットに従って提供されている．hikiutils は CLI(Command Line Interface) で操作するため, オプション解析をおこなう必要がある．gem には, この用途 に適合したライブラリがいくつも提供されている．この中 で, あまり利用頻度は高くないが古くから開発され, 使用例 が広く紹介されている optparse を利用している．

3　結果

3.1　thor と optparse のコードの比較

3.1.1　Thor とは

Thor とは，コマンドラインツールの作成を支援するライブラリのことである． git や bundler のようにサブコマンドを含むコマンドラインツールを簡単に作成することができる．

■Thor の基本的な流れ

1. Thor を継承したクラスのパブリックメソッドがコマンドになる
2. クラス.start(ARGV) でコマンドラインの処理をスタートする

3.1.2　optparse とは

optparse モジュールとは，getopt よりも簡便で，柔軟性に富み，かつ強力なコマンドライン解析ライブラリである．optparse では，より宣言的なスタイルのコマンドライン解析手法，すなわち OptionParser のインスタンスでコマンドラインを解析するという手法をとっている．これを使うと，GNU/POSIX 構文でオプションを指定できるだけでなく，使用法やヘルプメッセージの生成も行える．

■optparse の基本的な流れ

1. OptionParser オブジェクト opt を生成する
2. オプションを取り扱うブロックを opt に登録する
3. opt.parse(ARGV) でコマンドラインを実際に parse する

3.1.3　コードの解説

■hikiutils の実行

- Thor の実行コード

```
1 # -*- coding: utf-8 -*-
2 require 'thor'
3 require 'kconv'
```

```
4  require 'hikidoc'
5  require 'erb'
6  require "hikiutils/version"
7  require "hikiutils/tmarshal"
8  require "hikiutils/infodb"
9  require 'systemu'
10 require 'fileutils'
11 require 'yaml'
12 require 'pp'
13
14 module Hikithor
15
16   DATA_FILE=File.join(ENV['HOME'],'.hikirc')
17   attr_accessor :src, :target, :editor_command, :browser, :
         data_name, :l_dir
18
19   class CLI < Thor
20    def initialize(*args)
21       super
22       @data_name=['nick_name','local_dir','local_uri','
             global_dir','global_uri']
23       data_path = File.join(ENV['HOME'], '.hikirc')
24       DataFiles.prepare(data_path)
25
26       file = File.open(DATA_FILE,'r')
27       @src = YAML.load(file.read)
28       file.close
29       @target = @src[:target]
30       @l_dir=@src[:srcs][@target][:local_dir]
31       browser = @src[:browser]
32       @browser = (browser==nil) ? 'firefox' : browser
33       p editor_command = @src[:editor_command]
34       @editor_command = (editor_command==nil) ? 'open␣-a␣mi'
             : editor_command
35    end
36 HTML_TEMPLATE = <<EOS
37 <!DOCTYPE html
38     PUBLIC "-//W3C//DTD␣HTML␣4.01␣Transitional//EN"
39     "http://www.w3.org/TR/html4/loose.dtd">
40 <html lang="ja">
41 <html>
42 <head>
43   <meta http-equiv="Content-Language" content="ja">
44   <meta http-equiv="Content-Type" content="text/html;␣
         charset=UTF-8">
45   <title><%= title %></title>
46 </head>
```

```
47 <body>
48   <%= body %>
49 </body>
50 </html>
51 EOS
```

図 1

1. Hikithor::CLI.start(ARGV) が呼ばれる
2. initialize メソッドが呼ばれる
3. これでは Thor の initialize メソッドが呼ばれない
4. super を書くことで Thor の initialize メソッドが呼ばれる

- optparse の実行コード

```
1 # -*- coding: utf-8 -*-
2 require 'kconv'
3 require 'hikidoc'
4 require 'erb'
5 require "hikiutils/version"
6 require "hikiutils/tmarshal"
7 require "hikiutils/infodb"
8 require 'systemu'
9 require 'optparse'
```

```
10 require 'fileutils'
11 require 'yaml'
12 require 'pp'
13
14 module HikiUtils
15   DATA_FILE=File.join(ENV['HOME'],'.hikirc')
16   attr_accessor :src, :target, :editor_command, :browser, :
       data_name, :l_dir
17
18   class Command
19
20
21 HTML_TEMPLATE = <<EOS
22 <!DOCTYPE html
23     PUBLIC "-//W3C//DTD␣HTML␣4.01␣Transitional//EN"
24     "http://www.w3.org/TR/html4/loose.dtd">
25 <html lang="ja">
26 <html>
27 <head>
28   <meta http-equiv="Content-Language" content="ja">
29   <meta http-equiv="Content-Type" content="text/html;␣
       charset=UTF-8">
30   <title><%= title %></title>
31 </head>
32 <body>
33   <%= body %>
34 </body>
35 </html>
36 EOS
37
38     def self.run(argv=[])
39       print "hikiutils:␣provide␣utilities␣for␣helping␣hiki␣
           editing.\n"
40       new(argv).execute
41     end
42
43     def initialize(argv=[])
44       @argv = argv
45       @data_name=['nick_name','local_dir','local_uri','
           global_dir','global_uri']
46       data_path = File.join(ENV['HOME'], '.hikirc')
47       DataFiles.prepare(data_path)
48       read_sources
49     end
```

1. Hiki の HikiUtils::Command.run(ARGV) で hikiutils.rb の run メソッドを呼ぶ

図 2

2. new(argv).execute で execute メソッドが実行される

■コマンドの表示と実行

- Thor のコード

```
1     desc 'show,--show', 'show␣sources'
2     map "--show" => "show"
3     def show
4       printf("target_no:%i\n",@src[:target])
5       printf("editor_command:%s\n",@src[:editor_command])
6       @i_size,@n_size,@l_size,@g_size=3,5,30,15 #i,g_size
            are fixed
7       n_l,l_l=0,0
8       @src[:srcs].each_with_index{|src,i|
9         n_l =(n_l= src[:nick_name].length)>@n_size? n_l:
              @n_size
10        l_l =(l_l= src[:local_dir].length)>@l_size? l_l:
              @l_size
11      }
12      @n_size,@l_size=n_l,l_l
13      command = Command.new
14      header = command.display_format('id','name','local␣
```

```
          directory','global␣uri\
15 ',@i_size,@n_size,@l_size,@g_size)
16
17        puts header
18        puts '-' * header.size
19
20        @src[:srcs].each_with_index{|src,i|
21          target = i==@src[:target] ? '*':'␣'
22          id = target+i.to_s
23          name=src[:nick_name]
24          local=src[:local_dir]
25          global=src[:global_uri]
26          puts command.display_format(id,name,local,global,
               @i_size,@n_size,@l_siz\
27 e,@g_size)
28        }
29      end
30
31      desc 'version,--version,-v', 'show␣program␣version'
32      map "--version" => "version"
33      map "-v" => "version"
34      def version
35        puts HikiUtils::VERSION
36      end
37
38      desc 'add,--add', 'add␣sources␣info'
39      map "--add" => "add"
40      option :add
41      def add
42        cont = {}
43        @data_name.each{|name|
44          printf("%s␣?␣", name)
45          tmp = STDIN.gets.chomp
46          cont[name.to_sym] = tmp
47        }
48        @src[:srcs] << cont
49        show
50      end
51
52      desc 'target␣VAL,--target␣VAL', 'set␣target␣id'
53      map "--target" => "target"
54      def target(val)
55        @src[:target] = val.to_i
56        show
57      end
58
59      desc 'edit␣FILE,--edit␣FILE', 'open␣file'
```

```
60      map "--edit" => "edit"
61      def edit(file)
62        t_file=File.join(@l_dir,'text',file)
63        if !File.exist?(t_file) then
64          file=File.open(t_file,'w')
65          file.close
66          File.chmod(0777,t_file)
67        end
68        p command="#{@editor_command} #{t_file}"
69        system command
70      end
71
72      desc 'list [FILE],--list [FILE]', 'list files'
73      map "--list" => "list"
74      def list(file)
75        file ='' if file==nil
76        t_file=File.join(@l_dir,'text')
77        print "target_dir :"+t_file+"\n"
78        print `cd #{t_file} ; ls -lt #{file}*`
79      end
80
81      desc 'update FILE,--update FILE', 'update file'
82      map "--update" => "update"
83      def update(file0)
84        file = (file0==nil) ? 'FrontPage' : file0
85        t_file=File.join(@l_dir,'cache/parser',file)
86        FileUtils.rm(t_file,:verbose=>true)
87        info=InfoDB.new(@l_dir)
88        info.update(file0)
89        l_path = @src[:srcs][@target][:local_uri]
90        p command="open -a #{@browser} \'#{l_path}/?#{file}\'"
91        system command
92        p "If you get open error, try rackup from the src_dir.
              "
93        p "If you get 整形式になっていませ
              ん, try login as a valid user."
94      end
95
96      desc 'rsync,--rsync', 'rsync files'
97      map "--rsync" => "rsync"
98      option :rsync
99      def rsync
100       p local = @l_dir
101       p global = @src[:srcs][@target][:global_dir]
102       p command="rsync -auvz -e ssh #{local}/ #{global}"
103       system command
104     end
```

```
105
106     desc 'datebase␣FILE,--database␣FILE', 'read␣datebase␣
            file'
107     map "--database" => "database"
108     def database(file_name)
109       info=InfoDB.new(@l_dir)
110       p info.show(file_name)
111     end
112
113     desc 'display␣FILE,--display␣FILE', 'display␣converted␣
            hikifile'
114     map "--display" => "display"
115     def display(file)
116       body = HikiDoc.to_html(File.read(file))
117       source = HTML_TEMPLATE
118       title = File.basename(file)
119       erb = ERB.new(source)
120       t = File.open(file+".html",'w')
121       t.puts(erb.result(binding))
122       t.close
123       system "open␣#{t.path}"
124     end
125
126     desc 'checkdb,--checkdb', 'check␣database␣file'
127     map "--checkdb" => "checkdb"
128     def checkdb
129       result= InfoDB.new(@l_dir).show_inconsist
130       print (result=='') ? "db␣agrees␣with␣text␣dir.\n" :
              result
131     end
132
133     desc 'remove␣[FILE],--remove␣[FILE]', 'remove␣files'
134     map "--remove" => "remove"
135     def remove(file_name)
136       p text_path = File.join(@l_dir,'text',file_name)
137       p attach_path = File.join(@l_dir,'cache/attach',
            file_name)
138       begin
139         File.delete(text_path)
140       rescue => evar
141         puts evar.to_s
142       end
143       begin
144         Dir.rmdir(attach_path)
145       rescue => evar
146         puts evar.to_s
147       end
```

```
148
149         info=InfoDB.new(@l_dir)
150         p "delete␣"
151         del_file=info.delete(file_name)
152         info.show_link(file_name)
153         info.dump
154       end
155
156       desc 'move␣[FILE],--move␣[FILE]', 'move␣file'
157       map "--move" => "move"
158       def move(files)
159         begin
160           p file1_path = File.join(@l_dir,'text',files[0])
161           p file2_path = File.join(@l_dir,'text',files[1])
162         rescue => evar
163           puts evar.to_s
164           puts "error␣on␣move_files,␣check␣the␣input␣format,␣
                  especially␣comma␣sep\
165 aration."
166           exit
167         end
168         return if file1_path==file2_path
169         if File.exist?(file2_path) then
170           print ("moving␣target␣#{files[1]}␣exists.\n")
171           print ("first␣remove#{files[1]}.\n")
172           return
173         else
174           File.rename(file1_path,file2_path)
175         end
176
177         info=InfoDB.new(@l_dir)
178
179         db = info.db
180
181         pp file0=db[files[0]]
182         db.delete(files[0])
183         db[files[1]]=file0
184         db[files[1]][:title]=files[1] if db[files[1]][:title
              ]==files[0]
185         pp db[files[1]]
186
187         db.each{|ele|
188           ref = ele[1][:references]
189           if ref.include?(files[0]) then
190             p link_file=ele[0]
191             link_path = File.join(@l_dir,'text',link_file)
192
```

```
193             cont = File.read(link_path)
194             if Kconv.iseuc(cont) then
195               print "euc\n"
196               utf8_cont=cont.toutf8
197               utf8_cont.gsub!(/#{files[0]}/,"#{files[1]}")
198               cont = utf8_cont.toeuc
199             else
200               cont.gsub!(/#{files[0]}/,"#{files[1]}")
201             end
202
203             File.write(link_path,cont)
204
205             ref.delete(files[0])
206             ref << files[1]
207
208             p cache_path = File.join(@l_dir,'cache/parser',
                    link_file)
209             begin
210               File.delete(cache_path)
211             rescue => evar
212               puts evar.to_s
213             end
214           end
215         }
216
217         info.dump
218       end
219
220       desc 'euc␣FILE,--euc␣FILE', 'translate␣file␣to␣euc'
221       map "--euc" => "euc"
222       def euc(file)
223         p file_path = File.join(@l_dir,'text',file)
224         cont = File.readlines(file_path)
225         cont.each{|line| puts line.toeuc }
226       end
227     end
```

show メソッドから euc メソッドまではコマンドの表示と実行を行う.

1. コマンド名，コマンドの説明を一覧に表示させる
2. パブリックメソッドのコマンドを別のコマンド名でも実行できるようにする
3. コマンドの命令の実行コード

- optparse のコード

図 3

```
1     def execute
2       @argv << '--help' if @argv.size==0
3       command_parser = OptionParser.new do |opt|
4         opt.on('-v', '--version','show␣program␣Version.') {
              |v|
5           opt.version = HikiUtils::VERSION
6           puts opt.ver
7         }
8         opt.on('-s', '--show','show␣sources') {show_sources}
9         opt.on('-a', '--add','add␣sources␣info') {
              add_sources }
10        opt.on('-t', '--target␣VAL','set␣target␣id') {|val|
              set_target(val)}
11        opt.on('-e', '--edit␣FILE','open␣file') {|file|
              edit_file(file) }
12        opt.on('-l', '--list␣[FILE]','list␣files') {|file|
              list_files(file)}
13        opt.on('-u', '--update␣FILE','update␣file') {|file|
              update_file(file) }
14        opt.on('-r', '--rsync','rsync␣files') {rsync_files}
15        opt.on('--database␣FILE','read␣database␣file') {|
              file| db_file(file)}
16        opt.on('--display␣FILE','display␣converted␣hikifile'
              ) {|file| display(file)}
17        opt.on('-c', '--checkdb','check␣database␣file') {
```

```
            check_db}
18          opt.on('--remove␣FILE','remove␣file') {|file|
                remove_file(file)}
19          opt.on('--move␣FILES','move␣file1,file2',Array) {|
                files| move_file(files)}
20          opt.on('--euc␣FILE','translate␣file␣to␣euc') {|file|
                 euc_file(file)}
21          opt.on('--initialize','initialize␣source␣directory')
                 {dir_init() }
22        end
23        begin
24          command_parser.parse!(@argv)
25        rescue=> eval
26          p eval
27        end
28        dump_sources
29        exit
30      end
31
32      def dir_init()
33        begin
34          p target_dir = File.readlines('./.hikirc')[0]
35        rescue
36          p target_dir=@src[:srcs][@target][:local_dir]
37          File.open('./.hikirc','w'){|file| file.print "#{
                target_dir}\n"}
38        end
39        cp_files=[['Rakefile_hiki_sync','Rakefile'],
40                  ['hiki_help.yml','hiki_help.yml']]
41        cp_files.each{|files|
42          p source = File.join(File.expand_path('..', __FILE__
                ),'templates',files[0])
43          p target = File.join(Dir.pwd,files[1])
44          FileUtils.cp(source,target,:verbose=>true)
45        }
46        ['figs','data'].each{|dir|
47          begin
48            Dir.mkdir(dir)
49          rescue => e
50            print e
51          end
52        }
53        begin
54          p cont=File.read('./.gitignore')
55          unless cont.include?('.hikirc')
56            File.open('./.gitignore','w'){|file| file.print(".
                  hikirc\n")}
```

```
57          end
58        rescue
59          File.open('./.gitignore','w'){|file| file.print(".
                hikirc\n")}
60        end
61      end
62
63      def display(file)
64        body = HikiDoc.to_html(File.read(file))
65        source = HTML_TEMPLATE
66        title = File.basename(file)
67        erb = ERB.new(source)
68        t = File.open(file+".html",'w')
69        t.puts(erb.result(binding))
70        t.close
71        system "open #{t.path}"
72      end
73
74      def euc_file(file)
75        p file_path = File.join(@l_dir,'text',file)
76        cont = File.readlines(file_path)
77        cont.each{|line| puts line.toeuc }
78      end
79
80      def move_file(files)
81        begin
82          p file1_path = File.join(@l_dir,'text',files[0])
83          p file2_path = File.join(@l_dir,'text',files[1])
84        rescue => evar
85          puts evar.to_s
86          puts "error on move_files, check the input format, 
                especially comma separation."
87          exit
88        end
89        return if file1_path==file2_path
90        if File.exist?(file2_path) then
91          print ("moving target #{files[1]} exists.\n")
92          print ("first remove#{files[1]}.\n")
93          return
94        else
95          File.rename(file1_path,file2_path)
96        end
97
98        info=InfoDB.new(@l_dir)
99
100       db = info.db
101
```

```
102       pp file0=db[files[0]]
103       db.delete(files[0])
104       db[files[1]]=file0
105       db[files[1]][:title]=files[1] if db[files[1]][:title
              ]==files[0]
106       pp db[files[1]]
107
108       db.each{|ele|
109         ref = ele[1][:references]
110         if ref.include?(files[0]) then
111           p link_file=ele[0]
112           link_path = File.join(@l_dir,'text',link_file)
113
114           cont = File.read(link_path)
115           if Kconv.iseuc(cont) then
116             print "euc\n"
117             utf8_cont=cont.toutf8
118             utf8_cont.gsub!(/#{files[0]}/,"#{files[1]}")
119             cont = utf8_cont.toeuc
120           else
121             cont.gsub!(/#{files[0]}/,"#{files[1]}")
122           end
123
124           File.write(link_path,cont)
125
126           ref.delete(files[0])
127           ref << files[1]
128
129           p cache_path = File.join(@l_dir,'cache/parser',
                  link_file)
130           begin
131             File.delete(cache_path)
132           rescue => evar
133             puts evar.to_s
134           end
135         end
136       }
137
138       info.dump
139     end
140
141     def remove_file(file_name)
142       p text_path = File.join(@l_dir,'text',file_name)
143       p attach_path = File.join(@l_dir,'cache/attach',
              file_name)
144       begin
145         File.delete(text_path)
```

```
146         rescue => evar
147           puts evar.to_s
148         end
149         begin
150           Dir.rmdir(attach_path)
151         rescue => evar
152           puts evar.to_s
153         end
154 
155         info=InfoDB.new(@l_dir)
156         p "delete␣"
157         del_file=info.delete(file_name)
158         info.show_link(file_name)
159         info.dump
160       end
161 
162       def check_db
163         result= InfoDB.new(@l_dir).show_inconsist
164         print (result==’’) ? "db␣agrees␣with␣text␣dir.\n" :
              result
165       end
166 
167       def db_file(file_name)
168         info=InfoDB.new(@l_dir)
169         p info.show(file_name)
170       end
171 
172       def rsync_files
173         p local = @l_dir
174         p global = @src[:srcs][@target][:global_dir]
175 #"/Users/bob/Sites/nishitani0/Internal/data"
176 #"bob@dmz0:/Users/bob/nishitani0/Internal/data"
177 #       p command="rsync -auvz -e ssh #{local}/ #{global}"
178         p command="rsync␣-auvz␣-e␣ssh␣#{local}/␣#{global}"
179 #   system ’rsync -auvz -e ssh ~/Sites/nishitani0
      bob@nishitani0.kwansei.ac.jp:Sites/’
180         system command
181       end
182 
183       def update_file(file0)
184         file = (file0==nil) ? ’FrontPage’ : file0
185         #rm cache file
186         t_file=File.join(@l_dir,’cache/parser’,file)
187         begin
188           FileUtils.rm(t_file,:verbose=>true)
189         #update info file
190         info=InfoDB.new(@l_dir)
```

```
191         info.update(file0)
192
193         rescue
194           print "some␣errors␣on␣touch,␣but␣dont␣mind...\n"
195         end
196
197         #open file on browser
198         l_path = @src[:srcs][@target][:local_uri]
199 #        p command="open -a #{@browser} \'#{l_path}/?c=edit;p
    =#{file}\'"
200         p command="open␣-a␣#{@browser}␣\'#{l_path}/?#{file}\'"
201         system command
202         p "If␣you␣get␣open␣error,␣try␣rackup␣from␣the␣src_dir.
              "
203         p "If␣you␣get␣整形式になっていませ
              ん,␣try␣login␣as␣a␣valid␣user."
204       end
205
206       def list_files(file)
207         file ='' if file==nil
208         t_file=File.join(@l_dir,'text')
209         print "target_dir␣:␣"+t_file+"\n"
210         print `cd #{t_file} ; ls -lt #{file}*`
211       end
212
213       def edit_file(file)
214         t_file=File.join(@l_dir,'text',file)
215         if !File.exist?(t_file) then
216           file=File.open(t_file,'w')
217           file.close
218           File.chmod(0777,t_file)
219         end
220         p command="#{@editor_command}␣#{t_file}"
221         system command
222       end
223
224       def dump_sources
225         file = File.open(DATA_FILE,'w')
226         YAML.dump(@src, file)
227         file.close
228       end
229
230       def set_target(val)
231         @src[:target] = val.to_i
232         show_sources
233       end
234
```

```
235     def show_sources()
236       printf("target_no:%i\n",@src[:target])
237       printf("editor_command:%s\n",@src[:editor_command])
238       check_display_size()
239       header = display_format('id','name','local␣directory',
              'global␣uri')
240
241       puts header
242       puts '-' * header.size
243
244       @src[:srcs].each_with_index{|src,i|
245         target = i==@src[:target] ? '*':'␣'
246         id = target+i.to_s
247         name=src[:nick_name]
248         local=src[:local_dir]
249         global=src[:global_uri]
250         puts display_format(id,name,local,global)
251       }
252
253     end
254
255     def check_display_size
256       @i_size,@n_size,@l_size,@g_size=3,5,30,15 #i,g_size
              are fixed
257       n_l,l_l=0,0
258       @src[:srcs].each_with_index{|src,i|
259         n_l =(n_l= src[:nick_name].length)>@n_size? n_l:
                @n_size
260         l_l =(l_l= src[:local_dir].length)>@l_size? l_l:
                @l_size
261       }
262       @n_size,@l_size=n_l,l_l
263     end
264
265     def display_format(id, name, local, global)
266       name_length  = @n_size-full_width_count(name)
267       local_length = @l_size-full_width_count(local)
268       global_string= global.size < @g_size ? global : global
              [0..@g_size]
269       [id.to_s.rjust(@i_size), name.ljust(name_length),
270                local.ljust(local_length),
271                            global_string.ljust(@g_size)].join
                                ('␣|␣')
272     end
273
274     def full_width_count(string)
275       string.each_char.select{|char| !(/[ -~^^ef^^bd^^a1-^^
```

```
            ef^^be^^9f]/.match(char))}.count
276     end
277
278     def add_sources
279       cont = {}
280       @data_name.each{|name|
281         printf("%s␣?␣", name)
282         tmp = gets.chomp
283         cont[name.to_sym] = tmp
284       }
285       @src[:srcs] << cont
286       show_sources
287     end
288
289     def read_sources
290       file = File.open(DATA_FILE,'r')
291       @src = YAML.load(file.read)
292       file.close
293       @target = @src[:target]
294       @l_dir=@src[:srcs][@target][:local_dir]
295       browser = @src[:browser]
296       @browser = (browser==nil) ? 'firefox' : browser
297       p editor_command = @src[:editor_command]
298       @editor_command = (editor_command==nil) ? 'open␣-a␣mi'
              : editor_command
299     end
300   end
301 end
```

display メソッドから add_sources メソッドまでは opt で登録されたコマンドの実行コードが書かれている.