

# 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

0.1	目的 . . . . .	3
1	<b>方法</b>	5
1.1	既存システムの説明 . . . . .	5
2	<b>結果</b>	7
2.1	CLI の解説 . . . . .	13
2.1.1	Thor . . . . .	13
2.1.2	optparse . . . . .	13
2.2	Thor の初期化 . . . . .	15
2.2.1	コード . . . . .	15
2.3	コマンド表示と処理 . . . . .	16
2.3.1	Thor . . . . .	16
2.3.2	optparse . . . . .	17
2.3.3	コード . . . . .	18
2.4	CLI の実行 . . . . .	21
2.4.1	Thor . . . . .	21
2.4.2	optparse . . . . .	21
2.4.3	コード . . . . .	22
3	<b>参考文献</b>	26

## 目次

```
![[概要|hikiutils_yamane_abstract]]
![[序論|hikiutils_yamane_introduction]]
=====
!概要
![[hikiutils_yamane_abstract]]
!序論
```

## 0.1 目的

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して使いやすくする。

=====

**!既存システムの説明**

**図に従って hiki system の動作概要を説明する。**

**!!caption:hiki web system と hiki system の対応関係。**

**{{attach\_view(hikiutils\_yamane\_09\_copy.001.jpg)}}}**

hiki は、hiki 記法を用いた wiki clone です。wiki はワード・カニンガムが作った wikiwikiweb を源流とする home page 制作を容易にするシステムで、hiki も wiki の基本要求仕様を満足するシステムを提供しています。wiki の特徴である web 上で編集する機能を提供しています。これを便宜上 hiki web system と呼びます。図にある通り、一般的な表示画面の他に、編集画面が提供されており、ユーザーはこの編集画面からコンテンツを編集することが可能です。リンクやヘッダー、リスト、引用、表、図の表示などの基本テキストフォーマットが用意されています。

hiki web system の実際の基本動作は、hiki.cgi プログラムを介して行われています。こちらを便宜上 hiki system と呼びます。hiki system は、data/text に置かれた書かれたプレーンテキストを html へ変換します。この変換は hikidoc[1-1] という hiki フォーマット converter を使っています。また、添付書類は cache/attach に、一度フォーマットした html は parser に置かれており、それらを参照して html を表示する画面を hiki.cgi は作っています。

さらに hiki system では検索機能, 自動リンク作成などが提供されています

研究室内の内部文書, あるいは外部への宣伝資料, さらに wikipedia のように重要な研究成果の発信などに西谷研ではこの hiki system を利用しています. 初心者にも覚えやすい直感的な操作です. しかし, 慣れてくるとテキスト編集や画面更新にいちいち web 画面へ移行せねばならず, 編集の思考が停止します. そこで, テキスト編集に優れた editor との連携や, terminal 上の shell command と連携しやすいように hikiutils という cli(command line interface) を作成して運用しています. この hikiutils のコマンドオプションの実装をしておいて, より使いやすくすることが本研究の目的です.

# 1 方法

## 1.1 既存システム

図に従って hiki syste

図 1 hiki web system と hiki system の対応関係.

hiki は、hiki 記法を用いた wiki clone です。wiki はワード・カニングガムが作った wikiwikiweb を源流とする home page 制作を容易にするシステムで、hiki も wiki の基本仕様を満足するシステムを提供しています。wiki の特徴である web 上で編集する機能を提供しています。これを便宜上 hiki web system と呼びます。図にある通り、一般的な表示画面の他に、編集画面が提供されており、ユーザーはこの編集画面からコンテンツを編集することが可能です。リンクやヘッダー、リスト、引用、表、図の表示などの基本テキストフォーマットが用意されています。

hiki web system の実際の基本動作は、hiki.cgi プログラムを介して行われています。こちらを便宜上 hiki system と呼びます。hiki system は、data/text に置かれた書かれたプレーンテキストを html へ変換します。この変換は hikidoc[1-1] という hiki フォーマット converter を使っています。また、添付書類は cache/attach に、一度フォーマットした html は parser に置かれており、それらを参照して html を表示する画面を hiki.cgi は作っています。さらに hiki system では検索機能、自動リンク作成などが提供されてい

ます

研究室内の内部文書，あるいは外部への宣伝資料，さらに wikipedia のように重要な研究成果の発信などに西谷研ではこの hiki system を利用しています．初心者にも覚えやすい直感的な操作です．しかし，慣れてくるとテキスト編集や画面更新にいちいち web 画面へ移行せねばならず，編集の思考が停止します．そこで，テキスト編集に優れた editor との連携や，terminal 上の shell command と連携しやすいように hikiutils という cli(command line interface) を作成して運用しています．この hikiutils のコマンドオプションの実装をしておいて，より使いやすくすることが本研究の目的です．

## 2 結果

```
!コマンド名の検討
!!コマンドの命名原則
=====
//!コマンド名の検討
!コマンドの命名原則
```

機能ごとの動作はコマンドのオプションによって指定されます。このオプションにどのような名前をつけるかは、どれだけコマンドを覚えやすいかという意味で重要です。コマンドの振る舞いを的確に表す名称をつける必要があります。

この振る舞いとしてもっとも受け入れやすいのが shell で用意されているコマンドです。pwd, ls, rm, touch, open などのもっとも直感的に動作がわかるコマンドです。hikiutils の振る舞いを予測できるシェルコマンドと同じ名前でオプションを提供するようにします。

```
!!hikiutils があらかじめ想定している利用形態
ここで hikiutils があらかじめ想定している利用形態を解説しておきます。
```

```
!!!caption:hikiutils があらかじめ想定している利用形態
=====
!hikiutils があらかじめ想定している利用形態
ここで hikiutils があらかじめ想定している利用形態を解説しておきます。
```

```
!!caption:hikiutils があらかじめ想定している利用形態
```

hikiutils は、

- local PC と global server とが用意されており、
- それらのデータを rsync で同期する

ことで動作することを想定しています。これは、ネットに繋がっていないオフラインの状況でもテキストなどの編集が可能で、さらに不用意な書き換えを防ぐための機構です。さらに、どちらもが何かあった時のバックアップともなって、ミスによる手戻りを防いでいます。

これらの設定は、`/.hikirc` に yaml 形式で記述・保存されています。

---

```
1 bob% cat ~/.hikirc
```



図 2

```
2 :srcs:
3 - :nick_name: new_ist
4   :local_dir: "/Users/bob/Sites/new_ist_data/ist_data"
5   :local_uri: http://localhost/ist
6   :global_dir: nishitani@ist.ksc.kwansei.ac.jp:/home/
                  nishitani/new_ist_data/ist_data
7   :global_uri: http://ist.ksc.kwansei.ac.jp/~nishitani/
8 - :nick_name: dmz0
9   :local_dir: "/Users/bob/Sites/nishitani0/Internal/data"
10  :local_uri: http://localhost/~bob/nishitani0/Internal
11  :global_dir: bob@dmz0:/Users/bob/Sites/nishitani0/
                  Internal/data
12  :global_uri: http://nishitani0.kwansei.ac.jp/~bob/
                  nishitani0/Internal
```

---

また、一般的に一人のユーザがいくつものまとまりとしての local-global ペアを保持して管理することが普通です。それぞれに `nick_name` をつけて管理しています。

---

```
1 bob% hiki -s
2 hikiutils: provide utilities for helping hiki editing.
3 "open -a mi"
4 target_no:1
5 editor_command:open -a mi
```

```

6  id | name          | local directory
                                   | global uri
7  -----

8  0  | new_ist       | /Users/bob/Sites/new_ist_data/ist_data
                                   | http://ist.ksc.k
9  *1 | dmz0          | /Users/bob/Sites/nishitani0/Internal/
      data          | http://nishitani
10  2  | ist          | /Users/bob/Sites/hiki-data/data
                                   | http://ist.ksc.k
11  3  | new_maple    | /Users/bob/Sites/new_ist_data/
      maple_hiki_data | http://ist.ksc.k

```

---

とすると、それらの一覧と、いま target にしている nick\_name ディレクリが表示されます。

#### !!コメント名と振る舞いの詳細

=====

#### !コメント名と振る舞いの詳細

開発の結果コマンドを以下のように書き換えました。上部に記した、特によく使うコマンドに関しては、shell でよく使われるコマンド名と一致するようにしました。

それぞれの意図を動作の解説として記述しています。

!!! open FILE

=====

\* open FILE

ファイルを編集のために editor で open. Editor は /.hikirc に

```
:editor_command: open -a mi
```

として保存されている。open -a mi を emacs などに適宜変更して使用。

!!! ls [FILE]

=====

\* ls [FILE]

local\_dir にあるファイル名を [FILE\*] として表示。例えば, hikiutils\_yamane 以下の拡張

表 1

変更前	変更後	動作の解説
edit FILE	open	open file
list [FILE]	ls	list files
rsync	rsync	rsync files
update FILE	touch	update file
show	pwd	show nick_names
target VAL	cd	target を変える, change directory とのメタファ
move [FILE]	mv	move file
remove [FILE]	rm	remove files
add		add sources info
checkdb		check database file
datebase FILE		read datebase file
display FILE		display converted hikifile
euc FILE		translate file to euc
help [COMMAND]		Describe available commands or one specific command
version		show program version

子がついたファイルを表示, hiki システムでは text ディレクトリーは階層構造を取る  
 とができない, 西谷研では directory の代わりにスネーク表記で階層構造を表している,

```

!!! rsync
=====
* rsync

```

local\_dir の内容を global\_dir に rsync する, 逆方向は同期に誤差が生じたり, permission  
 がおかしくなるので, 現在のところ一方向の同期のみとしている, したがって, 作業手順  
 としてはテキストの変更は local\_dir で読み行うようにしている,

```

!!! touch FILE
loccal_dir で書き換えた FILE の内容を local_uri に反映させ, ブラウザで表

```

示. シェルコマンドの

touch によって, 変更時間を現在に変え, 最新状態とするのに似せてコマンド名を touch としている.

!!! pwd

nick\_name の一覧と target を表示, current target を current dir とみなして,

コマンド名を pwd とした.

!!! cd VAL

=====

\* touch FILE

loccal\_dir で書き換えた FILE の内容を local\_uri に反映させ, ブラウザで表示. シェルコマンドの

touch によって, 変更時間を現在に変え, 最新状態とするのに似せてコマンド名を touch としている.

\* pwd

nick\_name の一覧と target を表示, current target を current dir とみなして,

コマンド名を pwd とした.

\* cd VAL

target を変える, change directory とのメタファ. ただし, いまのところ nick\_name では対応しておらず, nick\_name の番号を VAL 入力することで変更する.

{{toc}} hikiutils のコマンドライン解析ツールを optparse から thor に換えることでコマンドの書き換えを行うことができた。また、thor で書かれた hikiutils は optparse で書かれたものよりもコードが短くなり、コマンドの解析も簡単に行えることができた。ここでは thor と optparse のコードを比較し thor の良さを確認する。

## 2.1 CLI の解説

### 2.1.1 Thor

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 [1-2]。

Thor の基本的な流れとしては

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

という流れである [1-2]。

start に渡す引数が空の場合、Thor はクラスのヘルプリストを出力する。また、Thor はサブコマンドやサブサブコマンドも作ることができる。

### 2.1.2 optparse

optparse 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 [1-3]。

利用頻度はあまり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 の基本的な流れとしては

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.on に登録する
3. opt.parse(ARGV) でコマンドラインを実際に parse する

という流れである。

OptionParser はコマンドラインのオプション取り扱うためのクラスであるためオブ

ジェクト `opt` を生成され `opt.on` にコマンドを登録することができる。しかし、`OptionParser#on` にはコマンドが登録されているだけであるため、`OptionParser#parse` が呼ばれた時、コマンドラインにオプションが指定されていれば実行される。optparse にはデフォルトとして `-help` と `-version` オプションを認識する [1-4]。

## 2.2 Thor の初期化

図 3

- Thor の initialize でのコード
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 では require で optparse を呼び optparse の initialize を定義する必要はないが、Thor は initialize を定義する必要がある。Thor の定義方法は require で Thor を呼び CLI クラスで継承し、initialize メソッドに super を書くことで Thor の initialize が呼ばれる。initialize メソッド内では Thor の初期設定がされていないため、スーパークラスのメソッドを読み出してくれる super を書き加えることで図のように initialize メソッド内で Thor の initialize メソッドが呼ばれ定義される。

### 2.2.1 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

```

---

## 2.3 コマンド表示と処理

### 2.3.1 Thor

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



図 4

Thor では desc で一覧を表示させるコマンド名，コマンドの説明を登録する．しかし，ここで記述したコマンドは一覧で表示させるものであり，実行されることはないので実際のコマンドと対応させる必要がある．Thor では処理実行を行うメソッドがコマンドとなる．しかし，それではコマンド名は 1 つしか使うことができない．ここで用いるものが map である．

`map A => B`

map とは B でしか読めないものを A でも読めるようにしてくれるものである．よって，これを使うことで別のコマンドも指定することができる．

### 2.3.2 optparse

1. OptionParser オブジェクト opt を生成
2. opt にコマンドを登録
3. 入力されたコマンドの処理のメソッドへ移動

optparse では OptionParser オブジェクト opt の生成を行い，コマンドを opt に登録することでコマンドを作成することができる．しかし，これはコマンドを登録しているだけでコマンドの一覧ではこれを表示することができるが，コマンドの実行を行うためには実行を行うためのメソッドを作成する必要がある．optparse でのコマンドの実行は opt で登

図 5

録されたコマンドが入力されることでそれぞれのコマンドの処理を行うメソッドに移動し処理を行う。しかし、このコマンド登録はハイフンを付けたコマンドしか登録ができず、ハイフンなしのコマンド登録はまた別の手段でやらなくてはならない。以上より、Thorではコマンドの指定と処理には desc,map, 処理メソッドだけで済むが、optparse ではコマンドを登録するためのメソッドと処理メソッドが必要になってくる。また、コマンドはThor では処理メソッドがコマンド名になるが、optparse ではコマンドを登録するための処理も必要となってくる。

### 2.3.3 コード

- Thor

---

```
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', @i_size, @n_size, @l_size,
        @g_size)
15
16     puts header
17     puts '-' * header.size
18
19     @src[:srcs].each_with_index{|src, i|
20         target = i == @src[:target] ? '*' : '_'
21         id = target + i.to_s
22         name = src[:nick_name]
23         local = src[:local_dir]
24         global = src[:global_uri]
25         puts command.display_format(id, name, local, global,
            @i_size, @n_size, @l_size, @g_size)
26     }
27     end

```

---

## • optparse

---

```

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.') {
5                 |v|
6                 opt.version = HikiUtils::VERSION
7                 puts opt.ver
8             }
9             opt.on('-s', '--show', 'show_sources') {show_sources}
10            opt.on('-a', '--add', 'add_sources_info') {
11                add_sources }
12            opt.on('-t', '--target_VAL', 'set_target_id') {|val|
13                set_target(val)}
14            opt.on('-e', '--edit_FILE', 'open_file') {|file|
15                edit_file(file) }
16            opt.on('-l', '--list_FILE', 'list_files') {|file|
17                list_files(file)}
18            opt.on('-u', '--update_FILE', 'update_file') {|file|
19                update_file(file) }
20            opt.on('-r', '--rsync', 'rsync_files') {rsync_files}
21            opt.on('--database_FILE', 'read_database_file') {|
22                file| db_file(file)}

```

```

16         opt.on('--display_FILE', 'display_converted_hikifile'
17             ) {|file| display(file)}
18         opt.on('-c', '--checkdb', 'check_database_file') {
19             check_db}
20         opt.on('--remove_FILE', 'remove_file') {|file|
21             remove_file(file)}
22         opt.on('--move_FILES', 'move_file1,file2', Array) {|
23             files| move_file(files)}
24         opt.on('--euc_FILE', 'translate_file_to_euc') {|file|
25             euc_file(file)}
26         opt.on('--initialize', 'initialize_source_directory')
27             {dir_init() }
28     end
29     begin
30         command_parser.parse!(@argv)
31     rescue=> eval
32     p eval
33     end
34     dump_sources
35     exit
36 end
37
38 def show_sources()
39     printf("target_no:%i\n",@src[:target])
40     printf("editor_command:%s\n",@src[:editor_command])
41     check_display_size()
42     header = display_format('id','name','local_directory',
43         'global_uri')
44
45     puts header
46     puts '-' * header.size
47
48     @src[:srcs].each_with_index{|src,i|
49         target = i==@src[:target] ? '*' : ' '
50         id = target+i.to_s
51         name=src[:nick_name]
52         local=src[:local_dir]
53         global=src[:global_uri]
54         puts display_format(id,name,local,global)
55     }
56 end
57
58 def add_sources
59     cont = {}
60     @data_name.each{|name|
61         printf("%s? ", name)
62         tmp = gets.chomp

```

```
56         cont [
57     }
58     @src[:s
59     show_so
60     end
```

---

## 2.4 CLI の実行

### 2.4.1 Thor

図 6

- 実行手順

1. hiki\_thor の Hikithor::CLI.start(ARGV) で hikiutils\_thor.rb の CLI クラスを呼ぶ
2. hikiutils\_thor.rb の CLI クラスのメソッドを順に実行していく

Thor では start(ARGV) を呼び出すことで CLI を開始する。Hikithor::CLI.start(ARGV) を実行されることにより require で呼ばれている hikiutils\_thor.rb の CLI コマンドを順に実行する。そして、入力されたコマンドと一致するメソッドを探し、そのコマンドの処理が実行される。

### 2.4.2 optparse

- 実行手順

図 7

1. Hiki の `HikiUtils::Command.run(ARGV)` で `hikiutils.rb` の `run` メソッドを呼ぶ
2. `new(argv).execute` で `execute` メソッドが実行される

一方, `optparse` では `HikiUtils::Command.run(ARGV)` を実行される. `require` で呼び出された `hikiutils.rb` で `run` メソッドが実行される. そこでコマンドを登録している `execute` メソッドへ移動し入力したコマンドと対応させる. そして, 対応したコマンドの処理が行われるメソッドに移動することで実行される. このように `optparse` では実行を行うためのメソッドが必要であるが, `Thor` ではクラスのメソッドを順に実行していくため `run` メソッドと `execute` メソッドは必要ない. また, `optparse` での実行手順はメソッドの移動回数が多く複雑であるが, `Thor` は単純で分かりやすいものとなっている.

### 2.4.3 コード

- Thor

---

```
1 #!/usr/bin/env ruby
2
3 require "hikiutils_thor"
4
5 Hikithor::CLI.start(ARGV)
```

---

---

```
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

```

---

- optparse

---

```

1 #!/usr/bin/env ruby
2
3 require "hikiutils"
4
5 HikiUtils::Command.run(ARGV)

```

---

```

1   def self.run(argv=[])
2     print "hikiutils: provide utilities for helping hiki
         editing.\n"
3     new(argv).execute
4   end

```

---

```

5
6  def execute
7      @argv << '--help' if @argv.size==0
8      command_parser = OptionParser.new do |opt|
9          opt.on('-v', '--version', 'show_program_Version.') {
10              |v|
11              opt.version = HikiUtils::VERSION
12              puts opt.ver
13          }
14          opt.on('-s', '--show', 'show_sources') {show_sources}
15          opt.on('-a', '--add', 'add_sources_info') {
16              add_sources }
17          opt.on('-t', '--target_VAL', 'set_target_id') {|val|
18              set_target(val) }
19          opt.on('-e', '--edit_FILE', 'open_file') {|file|
20              edit_file(file) }
21          opt.on('-l', '--list_FILE', 'list_files') {|file|
22              list_files(file) }
23          opt.on('-u', '--update_FILE', 'update_file') {|file|
24              update_file(file) }
25          opt.on('-r', '--rsync', 'rsync_files') {rsync_files}
26          opt.on('--database_FILE', 'read_database_file') {|
27              file| db_file(file)}
28          opt.on('--display_FILE', 'display_converted_hikifile'
29              ) {|file| display(f\
30              ile)}}
31          opt.on('-c', '--checkdb', 'check_database_file') {
32              check_db}
33          opt.on('--remove_FILE', 'remove_file') {|file|
34              remove_file(file)}
35          opt.on('--move_FILES', 'move_file1,file2',Array) {|
36              files| move_file(file\
37              s)}}
38          opt.on('--euc_FILE', 'translate_file_to_euc') {|file|
39              euc_file(file) }
40          opt.on('--initialize', 'initialize_source_directory')
41              {dir_init() }
42      end
43  begin
44      command_parser.parse!(@argv)
45  rescue=> eval
46      p eval
47  end
48  dump_sources
49  exit
50  end

```

---



コードからも Thor のほうが短くなっていることが分かる。よって、Thor と optparse でのコードの違いは以上の部分になるが全体的にも Thor のほうがコードが短くなり、コマンドの定義も簡単に行うことができる。また、実行手順も分かりやすくコードが読みやすいため書き換えもすぐ行うことができるので、より直感的なコマンドを実装することも可能となった。

### 3 参考文献

[1-1] hikidoc, <https://rubygems.org/gems/hikidoc/versions/0.1.0>, <https://github.com/hikidoc>,  
アクセス.

[1-2] 「Thor の使い方まとめ」, <http://qiita.com/succi0303/items/32560103190436c9435b>  
,2017/1/30 アクセス.

[1-3] 「15.5. optparse - コマンドラインオプション解析器」, [http://docs.python.jp/2/library/opt](http://docs.python.jp/2/library/optparse.html)  
,2017/1/30 アクセス.

[1-4] 「library optparse」, <https://docs.ruby-lang.org/ja/latest/library/optparse.html>  
,2017/1/30 アクセス.