ユーザーメモソフト my_help の動作環境の拡張

関西学院大学理工学部
情報科学科 西谷研究室
27016635
岡端啓吾

2020 年 3 月

目 次

第1章　序論

1.1　背景

西谷研究室では，ユーザーメモソフトの my_help が使用されている．my_help を使用することで，ゼミのノート取りや，自身の課題などの進捗状況を，自分の言葉で書き込むことができている．私自身も使用しており，とても便利であると感じている．特にプログラマーにとっては，既存のメモソフトではなく，my_help を使用することにより，効率的に作業を進めることができるだろう．

1.2　目的

先ほどプログラマーにとっては，既存のメモソフトよりも，my_help を使用することで効率よく作業を進めることができると述べた．しかし，現在は my_help があまり普及していない．思い当たる原因としては，

- テストがない．
- mac ユーザーのみを対象としている．
- エディタは emacs しか使用できない．

これらの 3 点があげられる．テストがない件については，同研究室生の山口が担当している．mac ユーザーのみを対象としている，エディタは emacs しか使用できない件については，my_help が OSX の emacs 上で使用されることを想定しているからだ．そこで他の OS へ対応するため，まずは Linux の vim 上で，my_help を使用できるよう移植する．その後，windows にも対応させることで，my_help を使用できる OS の増加を目指す．エディタは，emacs 以外にも対応するため，エディタを変更できるメソッドを作成することで，ユーザーが好みのエディタを選択できるものを目指す．

これらの作業を行うにあたって，my_help のコマンドラインツールとして用いられている gli を thor に書き換える．本研究では，これらの工程を踏むことで，my_help のユーザーの数を増加させることを目的とする．

1.3　my_help

my_help とは，CUI(CLA) ヘルプの Usage 出力を真似て，user 独自の help を作成・提供する gem である．特徴としては，

- user が自分にあった man を作成
- 雛形を提供
- おなじ format, looks, 操作, 階層構造
- すぐに手が届く
- それらを追加・修正・削除できる

の 5 点があげられている [1]．メモを作る際に，決まった形が提供されるため，そこからユーザー好みに編集を行うことで，自分にあった manual の作成が可能となる．雛形が提供されることにより，一度操作について覚えてしまうと使い方を忘れることがない．プログラミングなどの作業中でも，ディレクトリの移動をすることなく，編集，参照することができるため，メモのように書き込みしておくことで，ユーザーの助けとなる．my_help では，orgmode が用いられている．orgmode は太字や斜体に対応しているだけでなく，図や表の挿入への対応もしている．他にもタイムスタンプを押すことで，日付管理なども行うこともでき，HTML や LaTeX への変換も容易であるため，プログラマーにとってのメモに適している．実際に my_help を使用した例が図 1.1である．

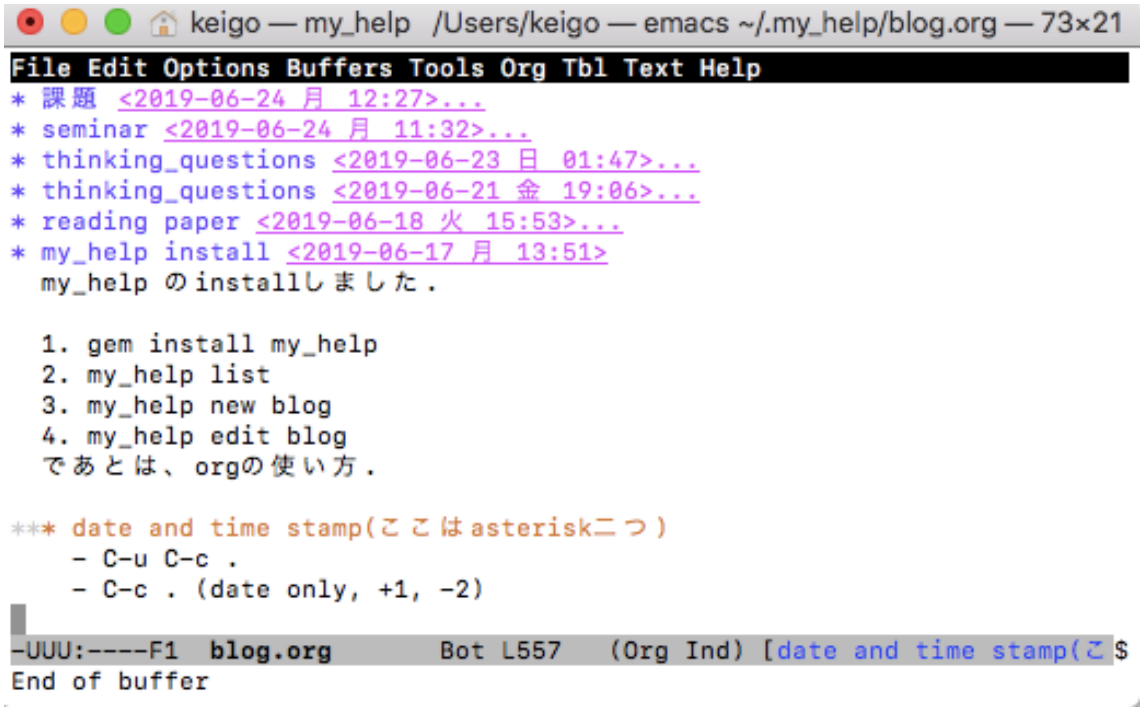

図 1.1: my_help_sample.

第2章　使用法

2.1　インストール

ターミナル上で，gem を用いてインストールすることで，my_help を使用することができる．

```
> gem install my_help
```

インストール後，一度動作確認を行う．

```
> my_help
Commands:
  my_help delete HELP             # delete HELP
  my_help edit HELP               # edit HELP
  my_help git [push|pull]         # git push or pull
  my_help help [COMMAND]          # Describe available commands or one specif...
  my_help list [HELP] [ITEM]      # list all helps, specific HELP, or ITEM
  my_help new HELP                # make new HELP
  my_help set_editor EDITOR_NAME  # set editor to EDITOR_NAME
  my_help setup                   # set up the test database
  my_help version                 # show version
```

このように，コマンド一覧が表示されれば，インストールは完了である．今回表示しているコマンド一覧は，my_help のバージョン 0.8.5 である．バージョンは version メソッドを用いることで確認することができる．

```
> my_help version
0.8.5
```

2.2　エディタの選択

my_help を使用するために，まずはエディタを登録する．

```
> my_help set_editor emacs
set editor 'emacs'
```

上記の emacs の部分に使用したい editor 名を入力することで，好みのエディタで my_help を起動することができる．一度登録したエディタを変更したい場合も，set_editor メソッドを用いることで，異なるエディタに変更することができる．

2.3　作成

独自の help を作成する方法として，sample という help を例にあげる．新たに help を作成するには，new メソッドを用いる．

```
> my_help new sample
"/Users/keigo/.my_help/sample.org"
"/Users/keigo/.rbenv/versions/2.5.1/lib/ruby/gems/2.5.0/gems/
my_help-0.8.5/lib/templates/help_template.org"
cp /Users/keigo/.rbenv/versions/2.5.1/lib/ruby/gems/2.5.0/gems/
my_help-0.8.5/lib/templates/help_template.org /Users/keigo/.my_help/sample.org
```

この操作により，sample という help が用意される．

2.4　編集

さきほど用意した sample を編集するには，edit メソッドを用いる．

```
> my_help edit sample
"/Users/keigo/.my_help/sample.org
```

この操作により，編集可能となる．以下の図 2.1が，edit メソッドにより開かれた sample である．

図 2.1: edit_sample.

このとき，sample は emacs の org 形式で格納されている．

2.5 削除

sample を削除する場合は，delete メソッドを用いる．delete メソッドを用いると，本当に削除するかが問われるので，Y を入力することで，削除することができる．

```
> my_help delete sample
Are you sure to delete /Users/keigo/.my_help/sample.org?[Yn] Y
rm /Users/keigo/.my_help/sample.org
```

この操作により，sample の削除が完了する．削除するかを問われた際に，Y 以外の文字を入力，もしくは何も入力せずに return キーを入力した場合，削除は行われない．

第3章　gliからthorへの書き換え

3.1　gliとthor

my_helpでは，コマンドラインツールとして，gliがgemとして用いられていた．以前，gliのバージョンを更新した際に，my_helpが起動不可となった．gliのバージョンを落とすことで，起動不可の問題は解決したが，今後バージョンを上げることができなくなるのは，脆弱性などの不具合が予想される．gliと振る舞いが変わらないコマンドラインツールとしてthorがある．

> Build command-suite CLI apps that are awesome. Bootstrap your app, add commands, options and documentation while maintaining a well-tested idiomatic command-line app

gliに関しては，コマンドに合った素晴らしいCLIアプリを構築する．アプリ，コマンドの追加，オプション，文書を実証済みの慣用的なコマンドラインアプリとして維持しながら起動する．と記されている [2].

> Thor is a toolkit for building powerful command-line interfaces.

thorに関しては，thorは力強いコマンドラインインターフェイスを構築するためのツールキットです．と記されている [3]．そこで，gliとthorを比較したものが，下の表3.1である [4].

表 3.1: gliとthorの比較.

	gli	thor
言語	Ruby	Ruby
ダウンロード数	約1600万	約30000万
コードの書きやすさ	Level3	Level5

gliとthorは，共にRubyを用いたコマンドラインツールとなっている．現在，gliは約1600万ダウンロードされているが，thorは約3億ダウンロードされているので，thorの方が利用者が多い分，サンプル数も豊富であると考えられる．そして，コードが書きやすいとされており，より簡単にコードを書くことができる．よって，my_help自体をgliからthorへ書き換えることで，脆弱性の面を解決できるだけでなく，my_helpの仕様の変更を容易にすることができるため，今までよりも便利なものにできるのではと考える．

3.2　書き換え方について

今回は，プログラムを書き換えることで開発を行った．まずは，github 内の daddygongon の my_help にアクセスする．my_help を自身のディレクトリにフォークし，bundle 上で動作することを確認する．動作確認後，gli で書かれた部分を全て thor に書き換えると，作業完了である．この際，書き換えを行うことで，それぞれのメソッドの動きが変わることのないように進めていく．最終的に，書き換え前後で全てのメソッドが同じ動きを実現できれば，開発は完了である．

3.3　書き換えについて

my_help の new メソッドを例に，gli と thor それぞれのプログラムを比較する．new メソッドとは，my_help において，新たな help を作成するメソッドである．new メソッドは，gli，thor どちらも独自の DSL(Domain Specific language) で書かれている．

```
desc 'make new HELP_NAME help'
arg_name 'HELP_NAME'
command :new do |c|
  c.action do |global_options,options,args|
    $control.init_help(args[0])
  end
end
```

gli の desc，arg_name は ruby の method 構文で書かれている．block 引数 (c) は順番が固定されたリストであるため，args を引き出すためには，いくつかの不必要な変数も呼び出す必要がある．

```
desc "new HELP", "make new HELP"
def new(help_name)
  invoke :setup
  $control.init_help(help_name)
end
```

一方，thor が提供する DSL は，ほぼ全てが method 形式を使っている．そのため不必要な変数を呼び出す必要がなく，予測がつきやすい．また，取れる引数が少ないため，素直に変数に代入することが可能である．

このように，gli は多くの機能を実現するために，my_help では不要なコード記述が必要であり，その結果，プログラムの行数が増えている．しかし，thor に書き換えることで，それぞれのメソッド毎にプログラムを短縮できるため，最終的に大幅なプログラムの短縮が可能となる．

3.4 結果

gli から thor への書き換え自体は問題なく，my_help を動かすことができた．それぞれのコマンドについても，振る舞いについても変わることなく動作した．gli から thor に書き換えたことで，プログラムの行数を約 40%削減することができ，結果的に，my_help のプログラムの大幅な短縮に成功した．現在，thor で書き換えたことによる不具合は発生しておらず，my_help に用いるコマンドラインツールとしては，gli よりも thor が適していることが確認できた．

第4章　他の環境への対応

4.1　移植について

現在，my_help は OSX 上の emacs で動作することを想定したプログラムとなっている．第 3 章で述べた通り，gli から thor に書き換えることで，汎用性をもたせることができ，中間発表にて山口が報告した通り，my_help のテストを書くことができた [5]．これにより，他の環境への移植が容易になることが期待できる．ここでの他の環境とは，OSX 以外の OS または emacs 以外のエディタのことである．本章では実際に移植を実行する．

4.2　移植の仕方について

まずは，mac 以外の OS にて my_help の動作確認を行うために，ubuntu をインストールし，仮想環境を用いることで，mac の中に Linux の環境を構築する．Linux 上で my_help が動作することを確認後，emacs 以外のエディタで my_help の起動を目指すため，vim で my_help の編集が行えるように，プログラムを書き換える．vim で my_help の編集が行えることを確認後，エディタを選択できるメソッドを追加する．これらの作業が完了後，mac と Linux，emacs と vim のどちらの OS とエディタを組み合わせても，my_help を使用することができれば，開発は完了である．

4.3　仮想環境の構築

今回は，Vagrant を用いて仮想環境の構築を行った．まずは公式サイトより，Vagrant のインストーラーをインストール [6]．今回は，macOS 64-bit を選択した．ダウンロードしたファイルを開き，vagrant.pkg を開く．インストール画面がでるので，手順に沿ってインストール．インストール終了後，ターミナルに移動し，

```
> which vagrant
/usr/local/bin/vagrant
```

vagrant のインストールが成功していることを確認する．確認後，vagrant のためのディレクトリを作成する．

```
> mkdir vagrant_test
```

作成したディレクトリの中に移動する．

```
> cd vagrant_test
```

vagrant_test の中に ubuntu をインストールする．今回は，bento/ubuntu-18.04 を選択した．

```
> vagrant init bento/ubuntu-18.04
```

ls コマンドを入力し，vagrant_test の中に Vagrantfile が作成されていれば，インストール成功である．

```
~/vagrant_test> ls
Vagrantfile
```

Vagrantfile に，

```
#config.vm.network "private_network", ip:"192.168.33.10"
```

と記述された箇所があるので，文頭でコメントアウトしている＃を取り除く．

```
config.vm.network "private_network", ip:"192.168.33.10"
```

仮想環境を構築する準備が整ったので，

```
> vagrant up
```

を行うことで，ubuntu が起動し，仮想環境の構築は完了である．

4.4 Linux への対応

仮想環境の構築を行ったことにより，自身が使っている OSX のターミナル上で，ubuntu を立ち上げることが可能となった．まずは ubuntu を起動する．

```
> vagrant ssh
Welcome to Ubuntu 18.04.3 LTS (GNU/Linux 4.15.0-65-generic x86_64)
```

自身の mac であることを表していた部分が，

```
keigo@MAC-no-MacBook-Air ~/vagrant_test
```

コマンドを入力することで，

```
vagrant@vagrant:~
```

に変わっていることが確認することができれば，vagrant の ssh 接続で ubuntu の環境への接続が完了している．その後，my_help をインストールし，起動する．

```
$ gem install my_help
```

```
$ my_help
Commands:
  my_help delete HELP             # delete HELP
  my_help edit HELP               # edit HELP
  my_help git [push|pull]         # git push or pull
  my_help help [COMMAND]          # Describe available commands or one specif...
  my_help list [HELP] [ITEM]      # list all helps, specific HELP, or ITEM
  my_help new HELP                # make new HELP
  my_help set_editor EDITOR_NAME  # set editor to EDITOR_NAME
  my_help setup                   # set up the test database
  my_help version                 # show version
```

my_help が起動できることが確認することができたので，移植成功である．終了する際は，

```
$ exit
logout
Connection to 127.0.0.1 closed.
```

を行うことで，仮想環境への接続の解除が完了する．

4.5 vim への対応

my_help の起動はできたが，実際に編集を行うと，emacs で編集画面が開かれる．ubuntu では，エディタとして emacs よりも vim が主流となっているので，vim で my_help を動かすことを考える．現在は，emacs で開くことが前提となっているため，my_help にエディタに関する記述はされていない．my_help の動作に関するプログラムは，

```
my_help/lib/my_help/my_help_controll.rb
```

に記述されているので，このファイルを編集する．initialize メソッドに，my_help の起動に関する定義がされているので，エディタを emacs で起動することを記述する．

```
@editor = 'emacs'
```

これを記述することで，my_help 自体に影響はなく，今まで通り emacs で編集することができた．今回は，エディタを vim に変えたいので，emacs の部分を vim に書き換え，

```
@editor = 'vim'
```

この状態でテンプレートを編集すると，図 4.1のように emacs ではなく，vim で編集することができる．

vim と書き込んだ部分に使用したいエディタ名を入力することで，簡易的ではあるが，自身の好みのエディタで起動させることができるようになった．

図 4.1: my_help_vim.

4.6 全てのエディタへの対応

my_help の initialize メソッドで，直接エディタを指定することで，簡易的に好みのエディタで，my_help を起動させることが可能となった．開発する側としては，この部分を書き換えるとエディタが変わることが分かっているので問題はないが，ユーザー側に書き換えをしてもらうとなると，違う部分の書き換えや，必要な部分を削除してしまうなどの誤操作によりエラーが発生する可能性があるため，できればユーザーには簡単かつ安全な作業でエディタを選択してもらいたい．そこで，エディタを設定してもらうメソッドとして，新たに set_editor メソッドを作成する．

```
def set_editor(editor)
      @editor = editor
      file_name = '.my_help_conf.yml'
      @conf_file = File.join(@local_help_dir, file_name)
      conf = {editor: editor}
      File.open(@conf_file, 'w'){|f| YAML.dump(conf, f)}
      puts "set editor '#{@editor}'"
end
```

set_editor メソッドでは，引数として editor をとっている．ここに使用したいエディタ名を入力することで，.my_help_conf.yml に保存される．エディタを登録が成功すると，登録したエディタ名を表示するものになっている．コマンド例としてエディタを vim に変更する場合，

```
> my_help set_editor vim
set editor 'vim'
```

とすることで，変更が完了する．

このように，set_editor メソッドに，使用したいエディタ名を入力することで，好みのエディタで my_help を操作可能となる．

4.7 結果

結果としては，my_help を Linux 上で起動することができた．そして，エディタを指定することで，好みのエディタで my_help を編集することが可能となった．その後，set_editor メソッドを追加したことにより，my_help の動作に関わるプログラムをユーザーが直接書き換えることなく，簡単かつ安全な作業でエディタを選択することが可能となった．これにより，OS やエディタに縛られることなく, user 好みの環境で my_help を使用することが可能となるため，新規 user が増加することが期待できる．

第5章　プレリリース

5.1　プレリリースについて

my_help を移植するにあたって，構築した仮想環境上で動作することは確認することができた．しかし，他の環境への対応を確かめるために，様々な OS やエディタ環境での動作確認が必要となる．今回は，西谷教授の授業を受けている大学院生 22 名の方々に，第 2 章で述べた使用法の通り，インストール，エディタの選択，help の作成，編集，削除までの一連の動作確認，そして version メソッドを用いて my_help のバージョン確認を行っていただいた．

5.2　報告結果

今回 22 名の方々に，my_help のバージョン 0.8.2 をインストールしていただき，それぞれの環境での動作について報告していただいた．使用している環境としては，mac，linux，windows の 3 種類であった．

5.2.1　mac

mac を使用している方々は，もともと my_help が OSX 上の emacs で動作することを想定したものであるので，問題なく my_help を使用することができていた．ruby のバージョンが異なることにより，インストールは成功するが，起動はできないという問題があったが，ruby のバージョンを上げていただくことで，起動の確認をすることができた．また，emacs だけでなく，atom，vim での起動を確認することもできた．

5.2.2　linux

linux を使用している方々も，第 3 章にて仮想環境を用いて起動確認を行ったこともあり，my_help をインストール，起動ができたという報告が多かった．一部，thor がインストールされていないことにより，起動できないという報告があったが，thor をインストールしていただくと，無事起動することができた．

5.2.3 windows

windows を使用している方々にはエラーが目立った．インストールは成功するが，メソッドを使用する際にエラーが起きるという報告が多く見られた．これは，windows10 上の powerwhell の Encoding の default が Windows-31J であるためだ．これを書き換えるため，force_encoding を参考にする [7].

> The associated Encoding of a String can be changed in two different ways. First, it is possible to set the Encoding of a string to a new Encoding without changing the internal byte representation of the string, with String#force_encoding. This is how you can tell Ruby the correct encoding of a string.
>
> 二つの異なる方法で，関連付いた文字に書き換えることができる．一つ目は，String#force_encoding により，文字列の内部バイト表現を変えることなく，新たな文字コードへと書き換えることができる．これによって，Ruby に文字列の正しいエンコーディングを伝えられます．

```
string
#=> "R\xC3\xA9sum\xC3\xA9"
string.encoding
#=> #<Encoding:ISO-8859-1>
string.force_encoding(Encoding::UTF_8)
#=> "R\u00E9sum\u00E9"
```

> Second, it is possible to transcode a string, i.e. translate its internal byte representation to another encoding. Its associated encoding is also set to the other encoding. See String#encode for the various forms of transcoding, and the Encoding::Converter class for additional control over the transcoding process.
>
> 二つ目は，文字列をトランスコードすることも可能です．すなわち，内部バイト表現を他の Encording に翻訳することです．それに関連したエンコーディングは，他のエンコーディングにセットされます．トランスコーディングのいろいろな形式は，String#encode を見てください．そして，トランスコーディングプロセスの追加的な制御は Encoding::Converter クラスを見てください．

```
string
#=> "R\u00E9sum\u00E9"
string.encoding
#=> #<Encoding:UTF-8>
string = string.encode!(Encoding::ISO_8859_1)
#=> "R\xE9sum\xE9"
string.encoding
#=> #<Encoding::ISO-8859-1>
```

なので，もともと

```
if m = line.match(/^\* (.+)/)
```

だったものを，

```
m=line.force_encoding(Encoding::UTF_8).match(/^\* (.+)/u)
if m
```

とした．これは，line の Encoding を UTF-8 に強制的に書き換えて，それを match(/. . . /u) で utf で matching をかけるという手法である [8]．この作業により，windows でもメソッドを使用することが可能となった．

5.3 改善点

今回大学院生の方々にインストールしていただいた結果，いくつか指摘をいただいた．まずは，delete メソッドについてである．

```
%my_help delete test
my_help delete test
Are you sure to delete /home/user/.my_help/test.org?[Ynq]
%my_help delete test
my_help delete test
Are you sure to delete /home/user/.my_help/test.org?[Ynq] y
%my_help delete test
my_help delete test
Are you sure to delete /home/user/.my_help/test.org?[Ynq] Y
rm /home/user/.my_help/test.org
```

delete メソッドを実行すると，delete メソッドを誤まって使用したことで作成した help が削除しないように，yes or no を簡単化し，[Ynq] として本当に削除するかどうかが問われる．この際，y を小文字で入力しても削除は実行されず，大文字で Y を入力したときのみ，削除が実行される．

> my_help delete 実行時にそのまま Enter を入力しても削除が行われない
>
> [Ynq] の選択肢のうち，　y が大文字なので未入力の場合は Y(削除) が実行されるべきと思われる

との指摘をいただいた．選択肢に大文字と小文字を用いた場合，何も入力せずに実行すると，大文字の選択肢がデフォルトで実行されるというものである．こちらは，小文字の y ではなく，大文字の Y にするという行程を加えることで，本当に削除する意志があるのかを確かめることができるので，既存のままにさせていただいた．

次にいただいたのは，my_help の起動に関わる，initialize メソッドに関してである．

don't work on version check.

change line order between

load_conf

and

set_dir ...

との指摘をいただいた．my_help のバージョンの確認ができないので，load_conf メソッドと set_help_dir_if_not_exists メソッドの順番を変えるべきというものである．

```
def load_conf
  file_name = '.my_help_conf.yml'
  # @conf_file = File.join(Dir.pwd, file_name)
  @conf_file = File.join(@local_help_dir, file_name)
  begin
    conf = YAML.load_file(@conf_file)
    @editor = conf[:editor]
  rescue => e
    puts e.to_s.red
    puts 'make .my_help_conf.yml'.green
    set_editor(@editor)
  end
end
```

load_conf メソッドは，set_editor メソッドで登録したエディタを読み込むものである．load_conf メソッドには例外処理を組み込んでおり，エディタを読み込むことができなかった場合は，そのエラーを出力し，そのまま処理を続けるというものになっている．

```
def set_help_dir_if_not_exists
  return if File::exist?(@local_help_dir)
  FileUtils.mkdir_p(@local_help_dir, :verbose=>true)
  Dir.entries(@template_dir).each{|file|
    next if file=='help_template.org'
    file_path=File.join(@local_help_dir,file)
    next if File::exists?(file_path)
    FileUtils.cp((File.join(@template_dir,file)),@local_help_dir,:verbose=>true)
  }
end
```

set_help_dir_if_not_exists メソッドは，local_help_dir が存在しない場合に，ディレクトリを作成し，設定するというものである，

指摘をいただいた通り，これらのメソッドの順番を入れ替えることで，my_help のバージョン確認や，エディタの登録をバグを起こすことなくできるようになった．

最後にいただいたのは，list メソッドに関してである．

例えば、以下のように new_help.org を作成します：

```
#* license
   - cc by Shigeto R. Nishitani, 2016
#* head
  - 新しいヘルプ
#* item1_example
  - item1
#* item2_example
  - item2
```

> my_help list new_help を実行すると、"item1_example"に対応するアイテム名は"-i"と設定されますが、"item2_example"に対応するアイテム名は設定されません：

```
- 新しいヘルプ
     , head           : head
     , license        : license
   -i, item1_example  : item1_example
     , item2_example  : item2_example
```

> この状態で my_help list new_help -i を実行すると、"item1_example"のみに関するヘルプが表示されます：

```
- 新しいヘルプ
-----
item1_example
- item1
```

> org ファイルにある見出しの頭文字が ITEM 名になるので、このような挙動になるのだと思います。頭文字が同じである場合は、番号をつけるか、自分で ITEM 名を指定できるようなオプションが欲しいです。よろしくお願いします。

との指摘をいただいた．アイテム名の頭文字に重複が存在すると、アイテム名を指定してヘルプを表示できないというものである．

簡易的な方法ではあるが，現在のところは図 5.1のように help の中の item1_example と item2_example の順番を入れ替え，list メソッドを実行することで，item2_example のヘルプを表示できることを確認できた．

```
> my_help list new_help -i
- 新しいヘルプ
-----
item2_example
- item2
```

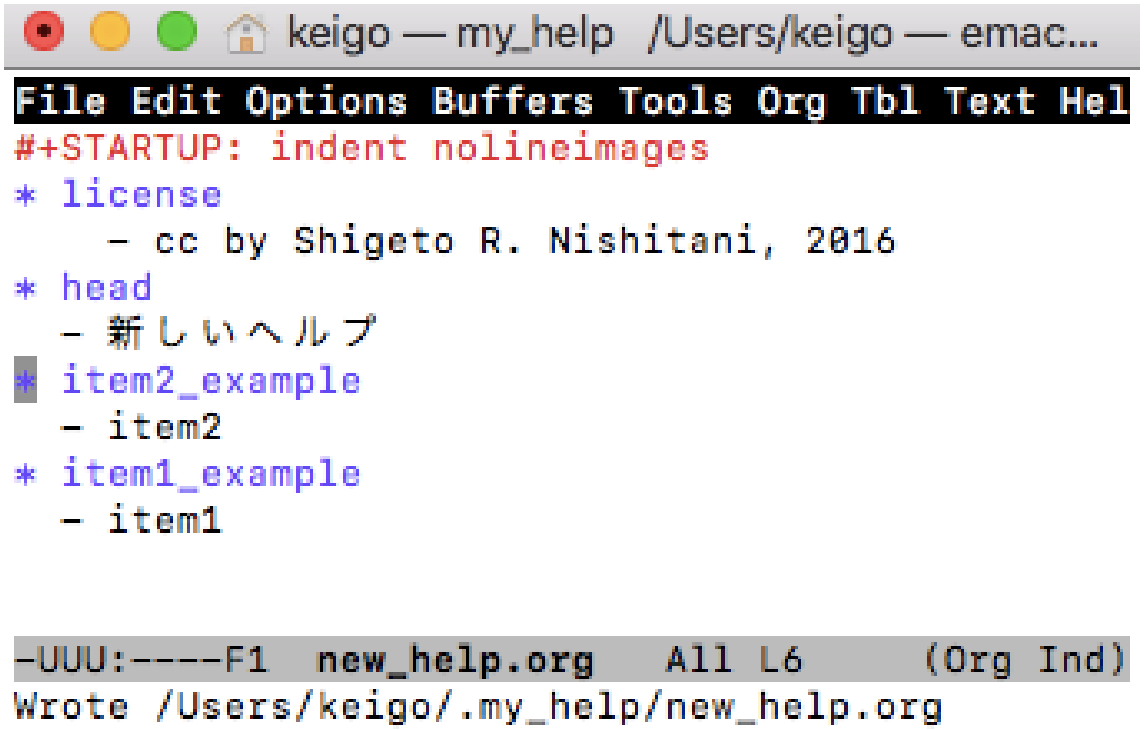

図 5.1: my_help_item.

今回の指摘のように，見出しの頭文字が重複する場合には，list メソッドでは上の見出しが優先して表示されるので，仮に同じ頭文字を使用する場合は，変更していただきたい．もしくは，別の頭文字を使用することで，list メソッドで表示できない問題を避けていただきたい．簡易的な解決策を提示することができたが，問題の解決には至っていないので，この件に関しては今後の課題としたい．

第6章　総括

本研究では，my_help を改良するべく，開発を進めた．以下に，開発内容を示す．

- コマンドラインツールとして使用されている gem を，gli から thor に変更．
- エディタを選択するメソッドとして，set_editor メソッドを追加．
- 文字コードを書き換えることで，windows への対応．

上記の開発により，もともと my_help は OSX の emacs のみを対象としていたが，他の OS やエディタでも my_help を使用可能となった．これにより，OS やエディタに縛られることなく, user 好みの環境で my_help を使用することが可能となるため，新規ユーザーの増加を期待できると考える．

謝辞

本研究を行うにあたり，終始多大なる御指導，御鞭撻をいただいた西谷滋人教授に対し，深く御礼申し上げます．また，本研究の進行に伴い，様々な助力，知識の供給を頂きました西谷研究室の同輩，先輩方に心から感謝の意を示します．本当にありがとうございました．

参考文献

[1] my_help, https://github.com/daddygongon/my_help, (accessed on 6 Feb 2020).

[2] gli, https://rubygems.org/gems/gli, (accessed on 6 Feb 2020).

[3] thor, https://rubygems.org/gems/thor, (accessed on 6 Feb 2020).

[4] glivsthor, https://ruby.libhunt.com/compare-gli-vs-thor, (accessed on 6 Feb 2020).

[5] 山口修平, "ユーザーメモソフト my_help のテスト作成及びコマンドの改良", 関西学院大学理工学部卒業論文 (2020).

[6] vagrant_download, https://www.vagrantup.com/downloads.html, (accessed on 6 Feb 2020).

[7] Encoding(Ruby 2.2.0), https://ruby-doc.org/core-2.2.0/Encoding.html, (accessed on 6 Feb 2020).

[8] Regexp (Ruby 2.2.0), https://ruby-doc.org/core-2.2.0/Regexp.html#class-Regexp-label-Encoding, (accessed on 6 Feb 2020).