my_help の Python 版の作成

関西学院大学理工学部情報科学科

西谷研究室

27020701 辻本和輝

2024 年 3 月

概 要

西谷の研究室ではメモをとるときに my_help[1] というユーザーメモソフトを使用している．my_help はプログラミングやコマンドに関する情報を手軽に整理・記録できるように設計されていて，メモを通じて知識を蓄積し，スキルを向上させることを目的としている．my_help の利用には Ruby の環境構築が必要となる．Ruby の環境になじんだユーザーには便利だが，他のユーザーは異なるバックグラウンドを持つため，Ruby の環境構築が未対応の場合がある．

よって，本研究では，my_help をより広く普及・活用するために，my_help を Python に書き換えたユーザーメモソフト kazuki_help[4] を開発した．Python は広く浸透しており，多様な利用者に適した環境を提供できると考えられる．

my_help の 9 つの機能を Python に書き換えて実行することができた．また，pip 化を行い PyPI にアップロードしたため，インストールやセットアップの手順が簡素化され kazuki_help の利用が容易になった．これらにより kazuki_help は，Ruby 以外のユーザーも手軽に利用でき、メモを通じて知識を蓄積し、スキルを向上させる手助けができるユーザーメモソフトになった．

目 次

図 目 次

第1章　序論

1.1　研究背景

西谷の研究室では，講義の内容や参照したサイトのリンク，コマンドのショートカット，学習の進歩管理等をメモするときに，my_help [1] というユーザーメモソフトを使用している．my_help はどのディレクトリにいても容易に参照・編集ができる．そのため，初心者プログラマーやシステム管理者が，プログラミングやコマンドに関する情報を手軽に整理・記録できるように設計されていて，メモを通じて知識を蓄積し，スキルを向上させることを目的としている．

my_help の利用には Ruby の環境構築が必要となる．既に Ruby 環境を構築している場合は問題ないが，他のユーザーは異なるバックグラウンドを持つため，Ruby の環境構築が未対応の場合がある．

本研究では，my_help をより広く普及・活用するために、my_help を Python に書き換えたユーザーメモソフト kazuki_help [4] を開発した．Python は広く浸透しており，多様な利用者に適した環境を提供できると考えられる．この変更により，Ruby 以外のユーザーも手軽に kazuki_help を利用でき、メモを通じて知識を蓄積し、スキルを向上させる手助けをすることが期待できる．

第2章　手法

2.1　my_help

2.1.1　my_help の特徴

my_help の特徴は 2 つ挙げられる.

1. ターミナルのみでの利用

   my_help は GUI を必要とせず，ターミナル上で動作するため，プログラミング中にメモを確認できる. 通常の GUI を備えたメモソフトウェアでは，別のウィンドウやアプリケーションを開いて情報を確認する必要があるが，my_help ではコマンドラインインターフェイス (CLI) を通じて直接ターミナル内で操作できる.

2. 自分独自のマニュアルを容易に作成・参照

   作業中のプロジェクトフォルダや，どのディレクトリにいるかに関係なく，ターミナル上で簡単にメモを呼び出し確認し，新しい情報を素早く追加・修正できる. この柔軟性により，プロジェクトごとに異なるメモやドキュメントを管理しやすくなる.

2.1.2　my_help の機能

my_help のコマンドの一覧は以下のようになる.

```
> my_help
my_help commands:
  my_help delete [HELP]       # delete HELP
  my_help edit [HELP]         # edit help
  my_help git [pull|push]     # git operations
```

```
my_help hello                # hello
my_help help [COMMAND]       # Describe available commands o...
my_help init                 # initialize my_help environment
my_help list [HELP] [ITEM]   # list helps
my_help new  [HELP]          # mk new HELP
my_help set [:key] [VAL]     # set editor or ext
my_help version              # show version
```

my_help の機能は，kazuki_help と同様である．機能の詳細は，付録 A に示している．

2.2 利用環境の構築

2.2.1 Python

Python は広く普及しており，多くのプログラマーなどに利用されている．そのため，Python で開発されたツールやアプリケーションは幅広いユーザにアクセスしやすくなっている．kazuki_help をより広く普及・活用するために Python で開発した．

2.2.2 Python のインストール

'kazuki-help' パッケージを使用するには，まず Python をインストールする必要がある [2]．最新の Python バージョンは公式ウェブサイトからダウンロードし，インストールする．

2.2.3 pip のインストール

Python のパッケージ管理ツールである pip をインストールする．pip は通常，Python と一緒にインストールされるが，確認のために以下のコマンドを実行して pip がインストールされていることを確認する．

```
> pip --version
```

2.2.4 Emacs のインストール

'kazuki-help' パッケージでは Emacs を使用してファイルを編集するため，Emacs をインストールする必要がある．Emacs は公式ウェブサイトからダウンロードし，インストールする [3].

第3章　結果と考察

3.1　kazuki_helpのインストール方法

利用環境を構築し終えたら，'kazuki-help' パッケージをインストールする. 以下のコマンドを実行する.

```
> pip install kazuki-help
```

これにより，'kazuki-help' パッケージとそれに関する依存関係がローカル環境にインストールされる.

3.2　kazuki_helpの機能

ユーザが kazuki_help を効率的に利用し，プログラミングやコマンドに関する情報を整理・記録し，メモを通じて知識を蓄積する上で主要な機能'new'，'edit'，'list' の 3 つのコマンドについて説明する.

3.2.1　new

kazuki_help の new 機能は新しい'.org' ファイルを作成する. 以下は，new のソースコードの主要部分である.

```
1    def create_file(self, filename):
2            try:
3                subprocess.call(["touch",
4                   expanduser("~")+"/.kazuki_help/"+filename+".org"])
5                print(f'{filename} ファイルを作成しました。')
```

```
6             except Exception as e:
7                 print(f' ファイルの作成中にエラーが発生しました: {e}')
```

3 行目では，'touch' コマンドを使用して新しいファイルを作成している．このコマンドは新しいファイルを作成するか，既存のファイルのタイムスタンプを更新する．4 行目では，org ファイルを作成するパスを指定している．expanduser("~") はユーザーのホームディレクトリを表し，~/.kazuki_help/ ディレクトリに新しいファイルが作成される．5 行目では，ファイルの作成が成功した旨が表示される．6 行目以降は，例外が発生した場合のエラー処理を行い，エラーが発生した旨を表示する．このようにして，'.org' ファイルを作成する機能である new を実装した．

3.2.2 edit

kazuki_help の edit 機能は Emacs を使用して指定された'.org' ファイルを編集する．以下は，edit のソースコードの主要部分である．

```
1    def edit_file_with_emacs(self, filename):
2        if os.path.exists(expanduser("~")+"/.kazuki_help/"+filename+".org"):
3            try:
4                subprocess.run(['emacs',
5                    expanduser("~")+"/.kazuki_help/"+filename+".org"])
6                print(f'{filename} ファイルを Emacs で編集しました。')
7            except Exception as e:
8                print(f'Emacs でファイルを編集中にエラーが発生しました: {e}')
9        else:
10            print(f'{filename} は存在しません。編集できません。')
```

2 行目では，指定したファイルが存在するかどうか確認している．os.path.exists はファイルやディレクトリの存在を確認する関数だ．4 行目では，subprocess モジュールを使用して Emacs を呼び出し，指定したファイルを編集する．6 行目では，ファイルの編集が成功したことを表示し，7 行目では例外が発生した場合の処理を行い，エラーが発生した

旨を表示する. 9 行目以降は，指定されたファイルが存在しない場合のエラーメッセージを表示する. このようにして，Emacs で指定された'.org' ファイルを編集する機能である edit を実装した.

3.2.3 list

kazuki_help の list 機能は，指定されたディレクトリ内の'org' 拡張子を持つファイルや特定の'.org' ファイルの内容を表示する. この機能は my_help の機能をもとに作成したが，ソースコードに関して異なったアプローチをしている. 具体的に，どこが異なっているのか説明していく.

kazuki_help の list の機能にセクション内容の表示機能がある. 以下は，'kazuki_help list emacs' といったコマンドで list の後にファイルを指定すると，セクションが表示されている例である.

```
> kazuki_help list emacs
    ファイル: emacs のセクションの内容:
    - head
    - license
    - cursor_move
    - page_move
    - file_operation
    - edit_operation
    - window_operation
    - buffer_operation
    - quit_operation
```

また，以下は'kazuki_help list emacs cursor_move' といったコマンドでファイル名の後にセクションを指定すると，セクションの内容が表示されている例である.

```
> kazuki_help list emacs cursor_move
    * cursor_move
    - c-f, move Forward,              Move to forward or right
```

```
- c-b, move Backward,              Move to back or left
- c-a, go Ahead of line,           Move to the beginning of the line
- c-e, go End of line,             Move to the end of the line
- c-n, move Next line,      Move to next line
- c-p, move Previous line,  Move to previous line
```

my_help と kazuki_help ではファイルを指定している場合とファイル内にあるセクションを指定する場合の言語探索が異なる.

以下は my_help のセクション内容の表示機能である'list_help_with' メソッドのソースコードである.

```
1   def list_help_with(path, name, item)
2     @help_info = read_help(path)
3     output = ColorizedString["my_help called with name :
4                               #{name}, item : #{item}\n"].colorize(:cyan)
5     if item == nil
6       @help_info[:items].each_pair do |item, val|
7         item, desc = item.split(":")
8         desc ||= ""
9         output << "- %20s : %s\n" % [item, desc]
10      end
11    else
12      output << find_near(item)
13    end
14    return output
15  end
```

read_help メソッドを使用して指定されたファイルからヘルプ情報を読み込み，2 行目にある@help_info に格納する．3 行目では，ColorizedString を使用して，実行されたコマンドと指定されたアイテムの情報を含む初期の出力メッセージを生成する．5 行目では，item が nil（指定されていない）の場合，全てのアイテムとそれに対応する説明を表示す

る．もし item が指定されている場合，find_near メソッドを呼び出して指定されたアイテムに部分一致するアイテムを検索し，その内容を表示する．

以下は my_help のアイテムの検索機能である’find_near’ メソッドのソースコードである．

```
1   def find_near(input_item)
2    candidates = []
3    @help_info[:items].each_pair do |item, val|
4        candidates << item if item.include?(input_item)
5    end
6    if candidates.size == 0
7        "Can’t find similar item name with : #{input_item}"
8    else
9        contents = candidates.collect do |near_item|
10        ColorizedString["item : #{near_item} \n"].colorize(:cyan) +
11        @help_info[:items][near_item]
12       end
13       contents.join("\n")
14    end
15  end
```

3行目では，@help_info[:items] 内の各アイテムに対して，指定されたアイテム（input_item）が含まれているかを検索する．6 行目では，candidates が空であれば，”Can’t find similar item name with : #{input_item}” というエラーメッセージを返す．8 行目では，candidates が空でなければ，candidates 内の各アイテムに対して，そのアイテムの情報を修正して表示するための処理を行う．

このように list_help_with メソッドはアイテムの検索と表示をし，find_near メソッドは指定されたアイテムに部分一致する類似アイテムを検索し，それらの内容を修正して表示する．

一方で，kazuki_help のセクション内容の表示機能のソースコードは以下である．

```
1    def _display_section_content(self, filename, section=None):
```

```
2       try:
3           with open(filename, 'r') as file:
4               content = file.read()
5               lines = content.split('\n')
6               in_section = False
7               array = []
8               for line in lines:
9                   if line.startswith('* '):
10                      if section is None:
11                          print('- ' + line[2:])
12                      else:
13                          if section in line.strip('* '):
14                              in_section = True
15                          else:
16                              in_section = False
17                  if in_section:
18                      array.append(line)
19              for line in array:
20                  print(line)
21      except Exception as e:
22          print(f'ファイル内容の取得中にエラーが発生しました: {e}')
```

3 行目では，with open(filename, 'r') as で指定されたファイルを読み込む．4 行目では，content = file.read() によって，ファイルの内容全体を取得する．5 行目では，ファイルの内容を行ごとに分割し，各行を処理する．in_section 変数はセクション内にいるかどうかを示すフラグだ．もし行が*で始まる（セクションの境界）場合，以下のような手順が実行される．10 行目では，section が None であれば，その行をハイフンで始まる形に変換して表示する．13 行目では，section が指定されている場合，その行がセクションの境界に位置しているかどうかを判定する．もしセクションの境界に位置している場合，14 行目で in_section を True に設定し，これは現在の行がセクションの範囲内にあることを示す．

セクションの範囲外にある場合，in_section は False に設定し，これは現在の行がセクションの範囲外にあることを示す．14 行目にある in_section が True の場合，セクション内の各行を array に追加し，最終的にセクション内の行を改行ごとに表示する．

このように，kazuki_help のセクション内容の表示機能では，*で始まる行がセクションの境界とされ，section の指定により，特定の範囲を抽出して表示する．

このため，my_help はアイテム単位の検索と表示が中心であるのに対し，kazuki_help は*で始まる行をセクションの境界とし，セクションに焦点を当てたというアプローチを取っている．

3.3 pip 化

3.3.1 背景と動機

Python プロジェクト開発では，依存関係の管理や再利用性，可搬性の向上が課題となっている．プロジェクトが成長するにつれて，複数のライブラリやモジュールへの依存が増加し，手動でこれらの依存関係を管理することが困難になってくる．これらの課題を解決し，効率的なプロジェクト管理を可能にするために，Python コミュニティでは pip という標準的なパッケージ管理ツールが開発された [5].

3.3.2 pip の概要

pip は Python 標準的なパッケージ管理ツールであり，Python Package Index(PyPI) からパッケージを容易にインストール，アンインストールできるツールである．pip を使用すると，インストール時に依存するライブラリやパッケージも自動的に解決され，環境構築が簡単に行える [6].

3.3.3 pip 化の手順

以下の手順を行い，プログラムをパッケージ化した [7].

1. pip がインストールされているか確認

   下記のコマンドを使用して，最新版をインストールし使用しているか確認する．

   ```
   > python3 -m pip install --upgrade pip
   ```

2. ローカルに以下のファイルを作成

   Python プロジェクトをパッケージ化するために，必要なファイルと構造を作成した．プロジェクトが共通化され，他の開発者が理解しやすい構造で提供できる．図 3.1はファイル構造を示している．

```
├── LICENSE                  # ライセンス情報を含むファイル
├── pyproject.toml           # プロジェクトのメタデータや依存関係の情報を含むファイル
├── README.md                # ドキュメンテーションを提供するためのファイル
├── src/
│   └── kazuki_help/          # Pythonモジュールが含まれている
│       ├── __init__.py
│       └── プログラム全体
└── tests/                   # プロジェクトのテストコードを格納するディレクトリ
```

図 3.1: パッケージ化の典型的なファイル構造．

3. メタデータを設定

   プロジェクトは一意なパッケージ名を使用しており，他の人たちがアップロードするパッケージとの衝突を回避している．図 3.2のとおり，ここではプロジェクトの基本情報，クラス分類，URL 情報，スクリプト情報が書かれている．これによって，ユーザーはプロジェクトの詳細を理解し，必要な情報を得るのに役に立つ．pyproject.toml ファイルに図 3.2のようにプログラムを書いた．

4. LICENSE ファイルを作成

   パッケージをインストールするユーザに対して，そのパッケージの使用条件を明示するためには，図 3.3に示されているように，ライセンス情報の明示，権利と責任を述べることが必要となっている．この方法により，ユーザーはプロジェクトの利用規約を理解し，ライセンスの機能について把握することができる．現在のライセンスは図 3.3のようになっている．

```toml
[project]
name = "kazuki_help"
version = "0.0.1"
authors = [
{ name="Kazuki Tsujimoto", email="fpd69555@kwansei.ac.jp" },
]
description = "A small example package"
readme = "README.md"
requires-python = ">=3.8"
classifiers = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: MIT License",
    "Operating System :: OS Independent",
]

[project.urls]
Homepage = "https://github.com/pypa/sampleproject"
Issues = "https://github.com/pypa/sampleproject/issues"

[project.scripts]
kazuki_help = "kazuki:main"
```

図 3.2: pyproject.toml の内容.

```
Copyright (c) 2018 The Python Packaging Authority

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

図 3.3: MIT ライセンスの内容.

5. 配布物アーカイブを生成

パッケージの配布物パッケージを生成する．まず，PyPA のビルドの最新版がインストールされているか確認する．下記のコマンドを利用してビルドをインストールする．

```
> python3 -m pip install --upgrade build
```

このコマンドを使用して，最新版をインストールし使用しているか確認する．ビルドをインストールすることで，ソースコードやプロジェクトの成果物を整備し，利用者が簡単に利用できる形に整形することが可能になる．

次に，pyproject.toml ファイルがあるのと同じディレクトリで下記のコマンドで実行する．

```
> python3 -m build
```

このコマンドを実行すると，下記の二つのファイルが生成される．

```
dist
├── kazuki_help-0.0.1-py3-none-any.whl
└── kazuki_help-0.0.1.tar.gz
```

.whl ファイルはビルド済配布物であり，tar.gz ファイルはソースコード配布物である．この二つのファイルを生成し，配布物アーカイブを生成することができた．

6. 配布物アーカイブをアップロード

配布物アーカイブをアップロードするために，PyPI [8] にアカウントを登録する．ユーザ登録が済むと次にパッケージを安全かつ効率的にアップロードするツールである twine のインストールする必要がある．下記のコマンドを使用して twine をインストールする．

```
> python3 -m pip install --upgrade twine
```

次にパッケージをアップロードするために下記のコマンドを使用する.

```
> python3 -m twine upload dist/*
```

このコマンドを使用すると，PyPI で登録したアカウントの認証情報の入力が促される. 入力が完了すると，配布物アーカイブをアップロードすることができる.

これらの pip 化によって，パッケージのインストールやセットアップの手順が格段に簡素化された. その結果，多くのユーザーや開発者に利用されることが期待される.

第4章　まとめ

本研究では，my_help の 9 つの機能を Python に書き換えて実行することができた．また，pip 化も行い PyPI にアップロードしたため，インストールやセットアップの手順が簡素化され kazuki_help の利用が容易になった．これらより，kazuki_help は，Ruby 以外のユーザーも手軽に利用でき，メモを通じて知識を蓄積し，スキルを向上させる手助けができるユーザーメモソフトになったと考えている．

謝辞

この研究を進めるにあたり，西谷滋人教授からは終始多くのご指導とご鞭撻を賜りました．心より感謝申し上げます．また，研究室の同輩や先輩方には，進行中において様々な助力と知識を提供していただき，この研究を成し遂げることができました．本当にありがとうございました．

参考文献

[1] ”my_help/README.org”, Daddygongon, https://github.com/daddygongon/my_help, (accessd on 25th Jan 2024).

[2] ”Python install”, Python 公式サイト, https://www.python.org/, (accessd on 10th Jan 2024).

[3] ”Emacs install”, GNU Emacs, https://www.gnu.org/software/emacs/download.html, (accessd on 21th Jan 2024).

[4] ”kazuki-help 0.0.2”, PyPI(kazuki_help), https://pypi.org/project/kazuki-help/, (accessd on 1st Feb 2024).

[5] ”Python プロフェッショナル プログラミング第 3 版”, 株式会社ビープラウド (著), (株式会社秀和システム, 2018/6/18)59-66.

[6] ”Python プロフェッショナル プログラミング第 3 版”, 株式会社ビープラウド (著), (株式会社秀和システム, 2018/6/18)67-91.

[7] ”Python のプロジェクトをパッケージングする”, Python Packaging User Guide, https://packaging.python.org/ja/latest/tutorials/packaging-projects/,(accessd on 17th Jan 2024).

[8] ”Python Package Index を使って Python パッケージを検索・インストール・公開する”, PyPI The Python Package Index, https://pypi.org/,(accessd on 21th Jan 2024).

付 録A　kazuki_helpの機能

A.1　kazuki_helpのコマンドの一覧

kazuki_help のコマンドの一覧です.

```
new     = 新しいファイルを作成
delete  = 指定したファイルを削除
edit    = 指定したファイルを Emacs で編集
version = ファイルのバージョンを表示
init    = プログラムおよび環境を初期化およびセットアップ
list    = ファイルを一覧表示
command = コマンドの説明を表示
git     = Git を使用した操作
set     = 設定を行う
```

A.2　各コマンドの詳細

A.2.1　init

1. 使用方法

   ```
   > kazuki_help init
   ```

2. 動作

   (a) 'kazuki_help init' コマンドが実行されると，プログラムおよび関連する環境の初期化が始まる.

   (b) '.kazuki_help' ディレクトリがユーザのホームディレクトリに作成される.

   (c) 標準出力は以下のように表示される.

   プログラムと環境を初期化およびセットアップします。

   (d) 初期化が正常に完了した場合，標準出力に以下のように表示されます.

   `.kazuki_help` ディレクトリを作成しました。

3. 注意事項 '.kazuki_help' ディレクトリが既に存在する場合，再度作成されない.

A.2.2 new

1. 使用方法

   ```
   > kazuki_help new [ファイル名]
   ```

   ここでの，ファイル名は作成するファイルの名前を示している．

2. 使用例

   ```
   > kazuki_help new example
   ```

   上記の例では，'kazuki_help new' コマンドを使用して，'example.org' という名前の新しいファイルを作成する．

3. 動作

   (a) 'kazuki_help new' コマンドが実行されると，指定されたファイル名の新しいファイルが'.kazuki_help' ディレクトリ内に作成される．

   (b) ファイルが正常に作成された場合，標準出力に以下のように表示される．

   `{ファイル名} ファイルを作成しました。`

   (c) ファイル作成中にエラーが発生した場合，標準出力に以下のように表示される．

   `ファイルの作成中にエラーが発生しました:{エラーメッセージ}`

4. 注意事項

   (a) 作成されるファイルはデフォルトで'.kazuki_help' ディレクトリ内に保存される．

   (b) すでに同じ名前のファイルが存在する場合，新しいファイルが上書きされる．

A.2.3 delete

1. 使用方法

   ```
   > kazuki_help delete [ファイル名]
   ```

2. 使用例

   ```
   > kazuki_help new example
   ```

   上記の例では，'kazuki_help delete' コマンドを使用して，'.kauzki_help' ディレクトリ内にある'example.org' という名前のファイルを削除する．

3. 動作

   (a) 'kazuki_help delete' コマンドが実行されると，指定されたファイル名のファイルが'.kazuki_help' ディレクトリ内から削除される．

   (b) 削除が正常に完了した場合，標準出力に以下のように表示される．

   `{ファイル名} ファイルを削除しました。`

   (c) 指定したファイルが存在しない場合，標準出力に以下のように表示される．

   `{ファイル名} は存在しません。削除できません。`

A.2.4 edit

1. 使用方法

   ```
   > kazuki_help edit [ファイル名]
   ```

2. 使用例

   ```
   > kazuki_help edit example
   ```

   上記の例では，'kazuki_help edit' コマンドを使用して，'example.org' という名前のファイルを Emacs で編集する．

3. 動作

   (a) 'kazuki_help edit' コマンドが実行されると，指定されたファイル名のファイルが'.kazuki_help' ディレクトリ内で検索される．

   (b) 発見された場合，Emacs が呼び出され，指定されたファイルが Emacs で開かれる．

   (c) 編集が正常に完了した場合，標準出力は以下のように表示される．

   `{filename}` ファイルを `Emacs` で編集しました。

   (d) 指定したファイルが存在しない場合，標準出力は以下のように表示される．

   `{filename}` は存在しません。編集できません。

4. 注意事項

   (a) Emacs がユーザの環境にインストールされている必要がある．

A.2.5 version

1. 使用方法

   ```
   > kazuki_help version
   ```

2. 動作

   (a) 'kazuki_help version' コマンドが実行されると，ファイルに埋め込まれたバージョン情報が表示される．

   (b) 標準出力は以下のように表示される．

   ファイルのバージョン: 1.0

A.2.6 list

1. 使用方法

   ```
   > kazuki_help list [ディレクトリまたはファイル名] [オプション]
   ```

   指定されたディレクトリまたはファイル内の.org ファイルを一覧表示する．ディレクトリが指定された場合はそのままのファイル一覧を表示する．ファイルが指定された場合はオプションによりセクションごとの内容を表示する．

2. 使用例

(a) `> kazuki_help list`
上記のコマンドでは，ユーザーがホームディレクトリ内にある.kazuki_help ディレクトリ内の.org ファイルを一覧表示する．

```
> kazuki_help list
    ファイル: todo.org
    ファイル: template.org
    ファイル: kazuki.org
    ファイル: emacs.org
    ファイル: faile.org
```

(b) `> kazuki_help list emacs`
上記のコマンドでは，ユーザーがホームディレクトリ内にある.kazuki_help ディレクトリ内の emacs.org ファイルの全セクションのタイトルを表示する．

```
> kazuki_help list emacs
    ファイル: emacs のセクションの内容:
    - head
    - license
    - cursor_move
    - page_move
    - file_operation
    - edit_operation
    - window_operation
    - buffer_operation
    - quit_operation
```

(c) `> kazuki_help list emacs cursor_move`
上記のコマンドでは，ユーザーがホームディレクトリ内にある.kazuki_help ディレクトリ内の emacs.org ファイルの指定されたタイトルのセクションとその中身を表示する．

```
> kazuki_help list emacs cursor_move
    * cursor_move
    - c-f, move Forward,              Move to forward or right
    - c-b, move Backward,             Move to back or left
    - c-a, go Ahead of line,          Move to the beginning of the line
    - c-e, go End of line,            Move to the end of the line
    - c-n, move Next line,      Move to next line
    - c-p, move Previous line,  Move to previous line
```

(d) `> kazuki_help list emacs move`
上記のコマンドでは，ユーザーがホームディレクトリ内にある.kazuki_help ディレクトリ内の emacs.org ファイルのタイトルに'move' が含まれているセクションとその中身を表示する．

```
> kazuki_help list emacs move
    * cursor_move
    - c-f, move Forward,              Move to forward or right
    - c-b, move Backward,             Move to back or left
    - c-a, go Ahead of line,          Move to the beginning of the line
    - c-e, go End of line,            Move to the end of the line
    - c-n, move Next line,      Move to next line
    - c-p, move Previous line,  Move to previous line
    * page_move
    - c-v, move Vertical,           Move to next page
    - M-v, move reversive Vertical,Move to previous page
    - c-l, centerise Line,          Move to center on current line
    - M-<, move Top of file,          Move to the top of the file
    - M->, move Bottom of file,     Move to the end of the file
```

3. 動作

(a) 'kazuki_help list' コマンドを実行すると，ディレクトリまたはファイルが指定されていない場合，デフォルトでホームディレクトリ内にある .kazuki_help ディレクトリ内の .org ファイルを検索して一覧表示する．

(b) ディレクトリが指定された場合，そのディレクトリ内の .org ファイルを検索して一覧表示する．

(c) 指定されたディレクトリが存在しない場合，エラーメッセージが表示される．

(d) ファイルが指定された場合，オプションにより動作が変化する．
・オプションが指定されない場合: ファイル内の全セクションのタイトルが表示される．
・セクションが指定された場合: ファイル内の指定されたセクションの内容が表示される．セクションが見つからない場合もエラーメッセージが表示される．

A.2.7 command

1. 使用方法

```
> kazuki_help command
```

2. 動作

(a) 'kauzki_help command' コマンドが実行されると，利用可能なコマンドとそれに対応する機能の一覧が表示される．

(b) 標準出力は，[3.3.1 kazuki_help のコマンドの一覧] のように表示される．

A.2.8 git

1. 使用方法

```
> kazuki_help git [Git コマンド]
```

ここでの，[Git コマンド] は実際の Git コマンドを指定する．

2. 使用例

```
> kazuki_help git push
```

上記の例では，'kazuki_help git' コマンドを使用して Git の'push' 操作を行う．

3. 対応する Git コマンド一覧

(a) 'push':ローカルの変更をコミットし，それをリモートリポジトリにアップロードする．これにより，他の開発者との協力や変更の共有が可能となる．

(b) 'pull':リモートリポジトリから変更をダウンロードし，ローカルの作業ディレクトリに取り込む．これにより，他の開発者が行った変更を取得し，自分の作業と統合できるようになる．

(c) 'git status'，'git commit' など，一般的な Git コマンドも利用できる．これらは変更の状態や新しい変更を確認，変更をコミットしたりするために使用される．

4. 動作

(a) 'kazuki_help git' コマンドが実行されると，指定された Git コマンドが実行される．

(b) 標準出力には実行した Git コマンドに関するメッセージが表示される．

A.2.9 set

1. 使用方法

   ```
   > kazuki_help set Emacs org
   ```

   上記の例のように，エディタと拡張子は Emacs および org に固定されており，ユーザーはこれを変更することができない.

2. 動作

   (a) 指定されたエディタ（Emacs）と拡張子（org）が 'config.txt' ファイルに保存される.

   (b) 設定が正常に保存された場合，標準出力は以下のように表示される.

   設定を保存しました。

   (c) 保存中にエラーが発生した場合，標準出力は以下のように表示される.

   設定の保存中にエラーが発生しました: {エラーメッセージ}

3. 注意事項

   (a) 設定は 'config.txt' ファイルに保存され，エディタと拡張子は変更できない.