卒業論文

my_helpのスマートフォンからの閲覧機能の追加

関西学院大学理工学部

情報科学科 西谷研究室

27020708 鈴木博大

2024年3月

概 要

西谷研では，メモソフトとして主に my_help を使用している．本研究では my_help で作成される org ファイルのスマートフォンやタブレットからの閲覧機能の追加を目的とした．メモは，頻繁に読み返すことが重要であるが，my_help では PC を起動させる必要がある．手軽にメモを読み返すためには，スマートフォンやタブレットなど PC 以外の端末からでも，メモの閲覧が可能となる必要がある．

閲覧機能を追加するにあたって，既存の Git での共有方法と比較した．今までのファイル共有システムでは，複数の Git コマンドの入力，共有できるのは研究室のグループ内，GitHub リポジトリを作成する際に GUI を使用する必要があるという三つの改善点が見られた．そこで本研究では，GitHub リポジトリを作成する自動化スクリプト hiroto_create.rb，GitHub リポジトリにファイルの変更をプッシュするための自動化スプリプト hiroto_push.rb を作成した．またこれらを誰でも使用できるように RubyGems に公開した．

スマートフォンやタブレットから閲覧できるようにしたことによって，PC が使えない状況でもメモを読み返すことができ，一連の操作を CUI のみで操作することが可能になり，課題を解決できたと考えている．

目 次

図 目 次

第1章　序論

西谷研究室ではGitHubを用いてプログラムコードの共有している．また，my_helpというCUI(CLI)のみで操作可能なメモソフトを使用している．my_helpで作成したメモを，移動時などPCが使えない状況で，スマートフォンやタブレットから閲覧することでメモを見返す習慣ができ，研究をしていくうえで役立つだろうと考えた．しかし現時点ではorgファイルで作成したプライベートなメモをスマートフォンやタブレットで閲覧することが難しいことに加え，メモをプログラムコードと一緒に共有することで誰もが読めてしまう状況である．そこで独自コマンドhiroto_createで作成したリポジトリに，独自コマンドhiroto_pushでファイルを簡単にGitHub[1]に共有することで，自分だけのメモを共有することが出来た．また，スマートフォンから閲覧することでPCを開く手間を省くことが出来た．そこでorgファイルで作成したプライベートなメモを，スマートフォンやタブレットから閲覧することを本研究の目的とした．

第2章　手法

2.1　Git

Git とは分散型バージョン管理システムで，ファイルの変更履歴を管理するために使用される．

2.1.1　Git で共有するメリット

Git の仕組みとしてファイルの変更履歴を記録したり，追跡することが出来る．共同作業を行う際，共有しているフォルダのコピーを自分の PC に作ることが出来る．GitHub 公式アプリをスマートフォンからインストールすると，ファイルの内容をスマートフォンからでも閲覧できる．これにより，外出時など PC が手元にない状況でも，ファイルの閲覧が可能になる．

2.2　Git の基本概念

1. リポジトリ (Repository)

   プロジェクトのデータと変更履歴が格納される場所で，GitHub のリモートリポジトリ，開発者のマシン上のローカルリポジトリがある．

2. コミット (Commit)

   ファイルやディレクトリの追加・変更・削除などを含む一連の変更を表す．コミットにはメッセージが添付され，変更の内容を記述することが出来る．

3. ブランチ (Branch)

変更の流れを分岐して記録していくためのもの．分岐したブランチはほかのブランチの影響を受けないため，同じリポジトリ内で複数の変更を同時に行うことが出来る．

4. ステージングエリア

   Git リポジトリで行われる変更を一時的に保存し，次に行うコミットに含めるファイルや変更を指定する領域．Git では'git commit' を実行する前に，ローカルリポジトリにコミットするファイルを指定しておく必要がある．この作業を「変更をステージングする」という．

2.3 Git の基本コマンド

1. 'git add'

   変更をステージングエリアに追加

2. 'git commit'

   変更されたファイルをローカルリポジトリに追加

3. 'git pull'

   リモートリポジトリから最新の変更をローカルリポジトリに保存

4. 'git push'

   ローカルリポジトリの変更を，リモートリポジトリにアップロード

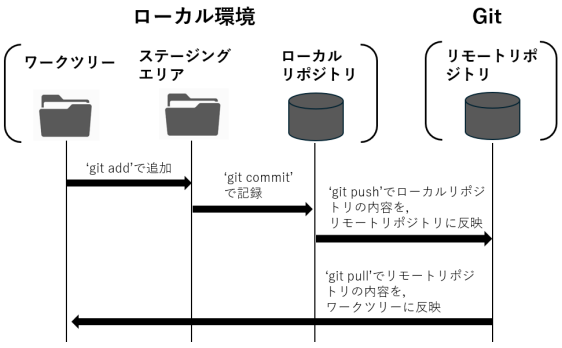

図 2.1: git のコマンドとその動作の概略図.

2.4 GitHub CLI

GitHub CLI[2] は、コンピューターのコマンドラインから GitHub を使用するためのコマンドラインツールである．GitHub CLI を使用することで，リポジトリの作成，コミットの作成とプッシュ，プルリクエストの作成など，様々な GitHub 関連の作業をコマンドラインで行うことが出来る．以下が今回使用する GitHub CLI のコマンド'gh' である．

- 'gh repo create'：新しいリポジトリを作成
- 'gh auth login'：GitHub の認証情報を設定

2.5 正規表現

いくつかの文字列を一つの形式で表現するための表現方法である．正規表現 [3] を使うことで，文字列の中から見つけたい文字列を検索することが出来る．今回の研究では，Git リポジトリのリモート URL から GitHub のユーザー名とリポジトリ名を抽出するために用いる．

2.6 RubyGems

RubyGems[4] は Ruby のプログラム言語のパッケージ管理システムであり, Ruby アプリケーションやライブラリの簡単なインストール，アップデートを可能にする．RubyGems を使用すると,「gem」というパッケージを利用して, 複雑な機能の実装でも簡単に行うことが可能になる.

gem のメリットとして，パッケージをインストールすると，Web アプリケーションの機能などを簡単に実装することが出来る．

また，依存関係を解決するための仕組みとして，Bundler[5] がある．プロジェクトごとに gem のバージョンを一元管理し，異なる環境で同じ依存関係を確実に再現するために利用する．例えば，インストールしようとしている gem 自身もほかの gem を利用している可能性がある．その場合，依存関係にある gem もあわせてインストールする必要がある．また，各 gem にはバージョンが設定されており，同じ gem でもバージョンが異なると，バグが発生する可能性がある．Bundler はこれらの問題を解決するために必要である．

Bundler は'Gemfile' と'gemfile.lock'[6] という二つのファイルを使用して依存関係を管理している．

Gemfile の役割として，Rails アプリを実行するのに必要な gem の一覧を管理するファイルである．'bundle install' というコマンドで gem を bundler を利用してインストールし，Gemfile に記載されている gem の一覧を参照することで，まだインストールされていない gem があれば，インストールを行うことが出来る．

Gemfile.lock の役割として，実際にインストールした gem の具体的なバージョンを記述し，Rails アプリに実際にインストールされた gem を依存関係にある gem も含めて一覧で表示することができる．

第3章　結果

3.1　hiroto_create

独自コマンド'hiroto_create' を実行すると，ubuntu 上で対話形式でリポジトリを作成することが出来る．

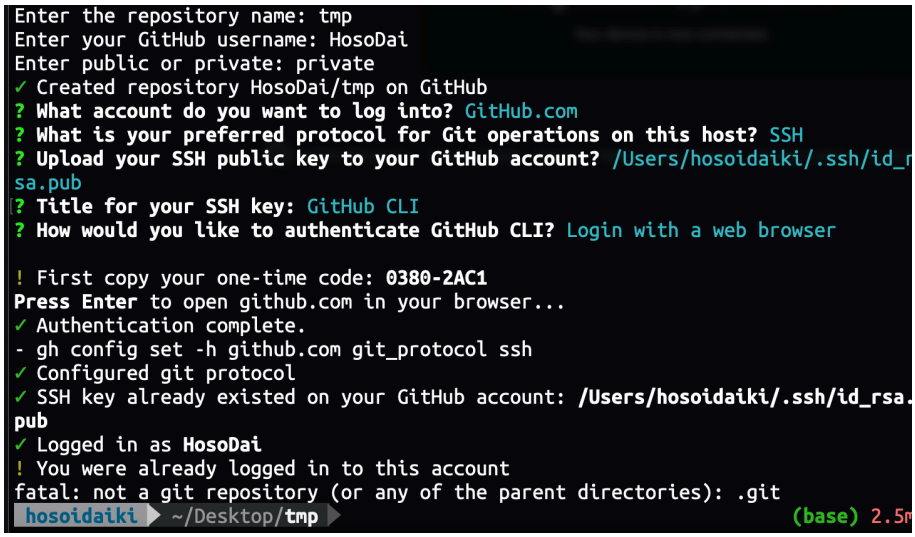

図 3.1: hiroto __ create 実行画面.

hiroto_create コマンドでは以下の操作が行われる．

- 現在の作業ディレクトリに「.git」というリポジトリを構成するディレクトリを作成
- リポジトリ名と GitHub のユーザー名を入力
- そのリポジトリが public か private か選択
- gh コマンドがインストールされているかの確認
- リモートリポジトリの作成
- GitHub にログイン
- リモートリポジトリへの接続

3.1.1 gh コマンドがない場合

hiroto_create では GitHub CLI が使用されている．そのため GitHub が提供しているコマンドラインから GitHub を使用するためのオープンソースツールである gh コマンドを使用しているので，gh コマンドをコンピュータのシステム上にインストールする必要がある．このために，macOS では brew を，Linux では apt を使ったコードを追加する．

1. gh コマンドがインストールされているかの確認'which gh' の出力結果を' > /dev/null' とすることで，標準出力を無視する．その結果 true を返せば，gh コマンドがインストールされているということになる．

2. gh コマンドが見つからない場合 1 の結果が false の場合，ユーザーがインストールするかを (yes, no) で確認する．

3. gh コマンドのインストールユーザーが'yes' を選択した場合，macOS の場合は Homebrew を使用，Ubuntu の場合は APT を使用して gh コマンドをインストールする．

4. インストールの成功確認インストール後，再度 gh コマンドの存在を確認して，インストールが成功したかどうかを確認して結果を表示する．

```
ghコマンドが見つかりません。
ghコマンドをインストールしますか？ (yes/no): yes
GitHub CLI (gh)のインストール中...
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  gh
0 upgraded, 1 newly installed, 0 to remove and 117 not upgra
ded.
Need to get 0 B/6242 kB of archives.
After this operation, 33.7 MB of additional disk space will
be used.
Selecting previously unselected package gh.
(Reading database ... 279751 files and directories currently
 installed.)
Preparing to unpack .../gh_2.4.0+dfsg1-2_amd64.deb ...
Unpacking gh (2.4.0+dfsg1-2) ...
Setting up gh (2.4.0+dfsg1-2) ...
Processing triggers for man-db (2.10.2-1) ...
GitHub CLI (gh)のインストールが完了しました。
```

図 3.2: gh コマンドがインストールされていないときの出力.

3.2 hiroto_push

独自コマンド'hiroto_push'を実行すると，hiroto_createで作成されたリポジトリに，ローカルリポジトリの変更をリモートリポジトリに反映させることが出来る．その際，コミットメッセージを入力することが可能である．

```
hiroto@DESKTOP-8ELO98Q ~/g/s/s/practice (main)> hiroto_push
[main 4417ac8] Update file
 1 file changed, 1 insertion(+), 1 deletion(-)
origin  git@github.com:SuzukiHiroto08/practice1.git (fetch)
origin  git@github.com:SuzukiHiroto08/practice1.git (push)
リモートリポジトリはすでに存在しています。追加処理はスキップ
されました。
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 296 bytes | 148.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 2 local
 objects.
To github.com:SuzukiHiroto08/practice1.git
   093325c..4417ac8  main -> main
Branch 'main' set up to track remote branch 'main' from 'ori
gin'.
```

図 3.3: hiroto __ push 実行画面.

hiroto_push コマンドでは以下の操作が行われる．

- Git リポジトリのリモート URL から GitHub のユーザー名とリポジトリ名を正規表現で抽出
- 変更のステージング
- コミット
- branch を master から main に変更
- リモートリポジトリに接続しているかどうかの判定接続していない場合は SSH で接続．接続している場合はスキップ
- ローカルリポジトリの変更をリモートリポジトリに反映

3.3 スマートフォンからの閲覧画面

下の図は org ファイルで作成されたメモを hiroto_push で GitHub のリモートリポジトリに反映させることで，html に表示形式を変えてスマートフォン上で表示させた画像で

ある．

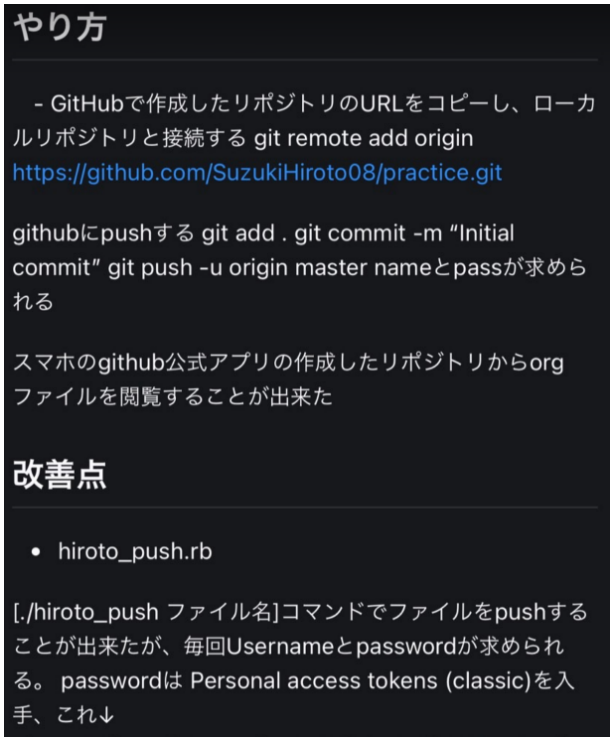

図 3.4: メモをスマートフォンから表示.

また，GitHub が提供している iPad 上の GUI アプリを使用すると，org のフォーマットのままでも，自動的に html に変換して表示してくれる．

3.4 gem の作り方

gem を使用するメリットは次の点,

1. 誰でも手軽にライブラリやソフトをインストールできる．
2. したがって，違った PC や違った OS で自分で作成したコマンドが正常に動作するのか検証することができる．

ことである [7].

3.4.1 事前準備

- GitHub に gem のソースコードを置くための新しいリポジトリの作成

- RubyGems.org のアカウント登録
- Bundler のインストール（'gem install bundler'）

3.4.2 雛形を作成

'bundle gem ./ -t' で必要なファイル群を自動で作成する．雛形のファイル構造を以下に表示する．

図 3.5: 雛形を作成したときのファイル構造.

3.4.3 gemspec の編集

gemspec のプログラムにおいて TODO の場所を以下の図 3.6 を参考に gem を作成する人に合わせて変更する．次に METADATA のプログラムをコメントアウトする．注意点として，この二点を編集していないと後に行う build が出来なくなる．図 3.7 が編集後のプログラムである．

3.4.4 gem の実装

自動作成した時点の Ruby ファイルは以下の図 3.8 である．

このプログラムの'# Your code goes here...' の箇所に今回実装する，hiroto_create と hiroto_push のプログラムをそれぞれ記載する．

spec.〇〇〇	記述内容
name	gemの名前
version	gemのバージョン
authors	gem作成者の名前
email	作成者のメールアドレス
summary	gemの簡単な説明
description	gemの細かい説明
homepage	gemの関連ページのURL

図 3.6: TODO の編集が必要な箇所と内容.

```ruby
Gem::Specification.new do |spec|
  spec.name = "hiroto_create"
  spec.version = HirotoCreate::VERSION
  spec.authors = ["SuzukiHiroto08"]
  spec.email = ["eyc69652@kwansei.ac.jp"]

  spec.summary = "hiroto_create"
  spec.description = "gem to create and connect remote repository to github on local PC"
  spec.homepage = "https://github.com/SuzukiHiroto08/hiroto_create"
  spec.license = "MIT"
  spec.required_ruby_version = ">= 2.6.0"

  # spec.metadata["allowed_push_host"] = "TODO: Set to your gem server 'https://example.com'"

  # spec.metadata["homepage_uri"] = spec.homepage
  # spec.metadata["source_code_uri"] = "TODO: Put your gem's public repo URL here."
  # spec.metadata["changelog_uri"] = "TODO: Put your gem's CHANGELOG.md URL here."

  # Specify which files should be added to the gem when it is released.
  # The `git ls-files -z` loads the files in the RubyGem that have been added into git.
  spec.files = Dir.chdir(__dir__) do
    `git ls-files -z`.split("\x0").reject do |f|
      (File.expand_path(f) == __FILE__) ||
        f.start_with?(*%w[bin/ test/ spec/ features/ .git appveyor Gemfile])
    end
```

図 3.7: gemspec 編集画面.

```ruby
# frozen_string_literal: true

require_relative "sample/version"

module Sample
  class Error < StandardError; end
  # Your code goes here...
end
```

図 3.8: Ruby ファイルの編集前.

3.4.5 実行コマンドの作成

今回はコマンドライン上で動かすことが目的なので，実行コマンドを作成する．

1. exe ディレクトリを作成（mkdir exe）
2. exe ディレクトリの下に実行ファイルを作成（touch exe/sample）
3. ファイルに実行権限を与える（’chmod -x exe/sample’）

3.5 gem の公開

- rubygems.org でのアカウント作成 [8]
- ’rake build’ して Git に push する
- 最後に’rake release’ で gem を公開

図 3.9: gem の公開.

3.6 RubyGems のバージョンアップ

一度 gem を公開すると，’rake release’ を実行してもバージョンアップは自動で行われない．

3.6.1 バージョンアップの手順

1. ソースコードに変更を加える

2. lib/hiroto_create/version.rb を書き換える今回は (VERSION = ”0.1.0” >> VERSION = ”0.1.1”) に書き換えた.

```ruby
# frozen_string_literal: true

module HirotoCreate
  VERSION = "0.1.1"
end
```

図 3.10: gems のバージョンのソースコード.

3. 変更点を commit，rake build，rake release

以上の作業を行うと，gem をアップデートすることが可能である.

第4章　まとめ

本研究では，hiroto_create と hiroto_push という二つの独自コマンドを作成し，それらを gem に公開した．しかし小規模で単純なスクリプトだったため，テストを書いていない．テストを書くことが品質が向上すると考えているため，テストを書く練習をしていきたいと考えた．

hiroto_push.rb プログラムにおいて，remeote_string に’system(git remote -v)’ コマンドを使用してリポジトリのリモート情報を取得しようとしたが，これは true, false で返すので失敗した．そこで’remote_string = IO.popen(’git remote -v’){—io— io.read}’ を使うことによって，コマンドの標準出力を読み取ることが出来た．

謝辞

本研究を進めるにあたり，様々なご指摘を頂いた西谷滋人教授に深く感謝いたします．また本研究の進行に伴い，様々な助力や協力を頂きました西谷研究室の同輩，先輩方に心から感謝いたします．本当にありがとうございました．

参考文献

[1] Git - https://backlog.com/ja/git-tutorial/ (accessd on 20 Jan 2021).

[2] GitHub CLI - https://docs.github.com/ja/github-cli/github-cli/about-github-cli (accessd on 20 Jan 2021).

[3] 正規表現 - https://userweb.mnet.ne.jp/nakama/ (accessd on 20 Jan 2021).

[4] RubyGems - https://and-engineer.com/articles/YUP-WBAAACMAu3gq (accessd on 20 Jan 2021).

[5] bundler - https://nishinatoshiharu.com/difference-gemfile-gemfilelock/ (accessd on 20 Jan 2021).

[6] Gemfile と Gemfile.lock - https://nishinatoshiharu.com/difference-gemfile-gemfilelock/ (accessd on 20 Jan 2021).

[7] gem の作り方 - https://qiita.com/9sako6/items/72994b8b1c00af4e61fe (accessd on 20 Jan 2021).

[8] RubyGems.org - https://rubygems.org/ (accessd on 20 Jan 2021).

付 録A　コード詳細

A.1　hiroto_create

これはGitHubリポジトリをCUIのみで作成する自動化スクリプトである．1~21行目では，リポジトリの名前やGitHubのユーザー名を入力することで，誰のGitHubにどのようなリポジトリを作成するか設定することが出来る．23~66行目では，使用するGithubCLIコマンドがPCにインストールされているか確認し，インストールされていない場合はインストールするためのプログラムである．68~79行目では，最初に入力したリポジトリ名とユーザー名を用いて，GitHubにログインし，リモートリポジトリに接続するためのプログラムである．

```
1 # frozen_string_literal: true
2
3 require_relative "hiroto_create/version"
4
5 module HirotoCreate
6   class Error < StandardError; end
7   def self.hiroto_create
8     # .git ファイルを作る
9     system("git init")
10
11    # リポジトリ名の入力
12    print 'Enter the repository name: '
13    repo_name = gets.chomp.strip
14
15    # GitHub ユーザー名の入力
```

```
16  print 'Enter your GitHub username: '
17  username = gets.chomp.strip
18
19  # public か private かを聞く
20  print 'Enter public or private: '
21  flag = gets.chomp.strip
22
23  # gh コマンドがインストールされているか確認
24  gh_installed = system('which gh > /dev/null')
25
26  unless gh_installed
27    puts "gh コマンドが見つかりません。"
28
29    # インストール確認
30    print "gh コマンドをインストールしますか？ (yes/no): "
31    user_input = gets.chomp.strip.downcase
32
33    case user_input
34    when 'yes'
35      # macOS の場合
36      if RUBY_PLATFORM.include?('darwin')
37        puts "GitHub CLI (gh) のインストール中..."
38        system('brew install gh')
39        # インストールの完了を待機する
40      end
41
42      # Linux の場合（ここでは Ubuntu）
43      if RUBY_PLATFORM.include?('linux')
44        puts "GitHub CLI (gh) のインストール中..."
```

```
45          system('sudo apt install gh')
46          # インストールの完了を待機する
47        end
48
49        # インストールが成功したか再度確認
50        gh_installed_after_install = system('which gh > /dev/null')
51        if gh_installed_after_install
52          puts "GitHub CLI (gh) のインストールが完了しました。"
53        else
54          puts "GitHub CLI (gh) のインストールに失敗しました。手動でインストールしてください。"
55          exit(1)
56        end
57
58      when 'no'
59        puts "インストールを中止しました。hiroto_create を使用するには手動で gh コマンドをインストールしてください。"
60        exit(1)
61
62      else
63        puts "無効な入力です。yes か no で答えてください。"
64        exit(1)
65      end
66    end
67
68     # リモートリポジトリの SSH
69     remote_repo_url = "git@github.com:#{username}/#{repo_name}.git"
70
71     # リモートリポジトリの作成
```

```
72      system("gh repo create #{username}/#{repo_name} --#{flag}")
73
74      # GitHub にログイン
75      system('gh auth login')
76
77      # リモートリポジトリへの接続
78      system("git remote add origin #{remote_repo_url}")
79    end
80  end
```

A.2 hiroto_push

これは GitHub リポジトリに変更をプッシュするためのスクリプトである．Git の基本コマンドである，'add, commit, pull, push' をスクリプトで書くことで、一つのコマンドでファイルを共有することが可能である．

```
1 # coding: utf-8
2 # frozen_string_literal: true
3
4 require_relative "hiroto_push/version"
5
6 module HirotoPush
7   class Error < StandardError; end
8   # coding: utf-8
9   def self.hiroto_push
10      # コミットメッセージの入力
11      print 'Enter the commit message: '
12      commit_message = gets.chomp.strip
13
14      remote_string = IO.popen('git remote -v'){|io| io.read}
```

```
15      url = remote_string.lines.first.chomp
16      pattern = "github\\.com:([^\\/]+)/([^\\.]+)\\.git"
17      matches = url.match(pattern)
18
19      username = matches[1]
20      repo_name = matches[2]
21
22      # リモートリポジトリの SSH
23      remote_repo_url = "git@github.com:#{username}/#{repo_name}.git"
24
25      # 変更のステージング
26      system("git add .")
27
28      # コミット
29      system("git commit -m '#{commit_message}'")
30
31      # master から main に変更
32      system('git branch -M main')
33
34      # remote リポジトリに接続してるかどうかの判定
35      existing_remote = system('git remote -v')
36
37      if existing_remote.nil?
38        system("git remote add origin #{remote_repo_url}")
39        puts "リモートリポジトリを追加しました。"
40      else
41        puts "リモートリポジトリはすでに存在しています。追加処理はスキップされました。"
42      end
```

```
43
44      # プッシュ()
45      system('git push -u origin main')
46    end
47  end
```