卒業論文
Qiitaへ記事を投稿するgem qiita_orgの開発

関西学院大学理工学部
情報科学科 西谷研究室
27017561 山本健太

2021年3月

概 要

研究を効率よく進めるためには，こまめに過程や考察をメモに残すことが重要であり，そのメモを指導者や共同研究者と共有することで円滑に研究が進む．西谷研究室では Github を使い，プログラムの共有を行なってきた．しかし，Github の共有だけでは研究の過程や考察の細部までは共有できておらず，逐次ゼミ内で口頭にて共有する必要があった．そこで

1. メモを記事として投稿できる Qiita を使い情報共有を行う，また，
2. 投稿工程を容易にするシステムを開発する

ことを目的とした．

本研究で開発した qiita_org は

1. gem として提供することにより，管理や配布が容易となる，
2. CLI builder gems の一つである thor を使いコマンドのコーディングをすることで作成が容易に行える，また，
3. Qiita:Team を使用することで研究の情報を記事として保存，研究室内でのみの共有が可能

となった．

qiita_org を研究過程に取り入れることで，情報共有が円滑になり研究が効率的になる．西谷の大学院の講義で，受講生に試用してもらい改善点を issue あるいは pull request で吸い上げた．

目 次

第 1 章　序論 **2**
1.1 研究の背景 2
1.2 研究の目標 2

第 2 章　手法 **3**
2.1 RubyGems 3
2.1.1 RubyGems とは 3
2.1.2 gem の使い方 3
2.2 thor 4
2.3 Qiita と Qiita:Team 5
2.4 Qiita API 6
2.5 ox-qmd 6
2.5.1 ox-qmd とは 6
2.5.2 使用方法 6

第 3 章　各コマンドの設計と考察 **8**
3.1 post 8
3.1.1 コマンドの引数 9
3.1.2 ファイルの読み込み 9
3.1.3 Title と Tag の取得 9
3.1.4 Markdown への変換 9
3.1.5 Markdown テキストの編集 10
3.1.6 投稿方法の確認 10
3.1.7 Qiita への送信 10
3.1.8 記事 ID の書き込みと Markdown テキストの削除 11
3.1.9 結果と改善案 11
3.2 upload 12
3.2.1 Selenium 12
3.2.2 cui と gui の組み合わせ 14
3.2.3 upload 手法の比較 14
3.3 config 15
3.3.1 設定ファイルの雛形と処理方法 15
3.4 template 16
3.4.1 用意されるテンプレートの内容 16

3.4.2 環境記載についての改善案 17
3.5 list と get 17
3.5.1 記事取得の手法 18
3.5.2 コマンドの利点と改善案 19
3.6 all 19
3.6.1 プログラムと仕様について 19
3.6.2 all 実行時の問題 20

第 4 章 Github による issue および pull request **21**
4.1 issue 21
4.1.1 Twitter と Qiita を紐づけし共有する 21
4.1.2 qiita_org 内の gems ディレクトリについて 22
4.1.3 未解決の issue について. 23
4.2 pull request 23
4.2.1 template コマンドの仕様変更 24
4.2.2 Twitter 投稿のバグについて 24
4.3 issue と pull request の重要性 24

第 5 章 総括 **25**

第1章　序論

1.1　研究の背景

西谷研では従来より，Github を用いてコードの共有をしている．しかしそれでは，研究の過程や結果，考察についてはゼミ内で口頭で説明する必要がある．口頭での説明より，文字媒体で記録に残したほうが後で確認もできるうえ，書くことにより記憶に残りやすく，文書が残ることで後続の研究にも役立つであろう．しかしその都度レポートを書き PDF 化するなどの手順を踏んでいては手間がかかり不便である．

そこで西谷研では，手軽に記事として文章を投稿でき，タグやタイトルが表示されるため確認も容易な Qiita という情報共有サービスを使用することとした．そして文書の作成，Qiita へ投稿し情報を共有する工程を円滑に，さらにその全てがターミナル上で完結できるようなシステムがあればいいのではないかと考えた．

1.2　研究の目標

先述したシステムを作成するにあたり，まず文書の作成には従来から西谷研で使い慣れている emacs 上の Org-mode を使用することとした．しかし，Qiita への投稿では Markdown 形式で書かれた文書が必要である．したがって，Org-mode で作成した文書を Markdown 形式の文書に変換する機能を作成しなければならない．

また，Qiita との通信をターミナル上だけで完結するためには，専用のコマンドを作成する必要がある．

以上のことから，Org-mode 文書を Markdown 文書に変換し，Qiita との通信を cui で行うための Ruby gem 'qiita_org' を開発することを本研究の目標とした．

第2章　手法

2.1　RubyGems

2.1.1　RubyGems とは

RubyGems[1] は Ruby 用のパッケージ管理システムであり，'gem' と呼ばれるライブラリの配布用標準フォーマットを提供している．gem の管理が容易にでき，gem を配布するためのサーバ機能も持っている．本研究ではこの gem の形式で qiita_org を開発していく．

2.1.2　gem の使い方

gem の雛形を作成するには bundler を使用する．bundler とは gem の依存関係やバージョンの管理をするためのツールである [2]．bundler は'gem install' コマンドでインストールすることができる．

```
> gem install bundler
```

'bundle' コマンドを用いて gem の雛形を作成する [3]．本研究で作成した'qiita_org' を作成する際は以下のように実行した．'-b' は実行可能なコマンドの雛形を，'-t' はテストコード配置用のディレクトリとテスト実行用のヘルプも同時に作成するオプションである．

```
> bundle gem qiita_org -b -t
Creating gem 'qiita_org'...
MIT License enabled in config
      create  qiita_org/Gemfile
      create  qiita_org/lib/qiita_org.rb
      create  qiita_org/lib/qiita_org/version.rb
      create  qiita_org/qiita_org.gemspec
      create  qiita_org/Rakefile
      create  qiita_org/README.md
      create  qiita_org/bin/console
      create  qiita_org/bin/setup
      create  qiita_org/.gitignore
      create  qiita_org/.travis.yml
      create  qiita_org/.rspec
      create  qiita_org/spec/spec_helper.rb
```

```
      create  qiita_org/spec/qiita_org_spec.rb
      create  qiita_org/LICENSE.txt
      create  qiita_org/exe/qiita_org
Initializing git repo in /Users/kentayamamoto/Github/qiita_org
Gem 'qiita_org' was successfully created. For more information on making a RubyGem
visit https://bundler.io/guides/creating_gem.html
```

このファイル群の lib の中にプログラムを作成していくことでコマンドができる．本研究では'lib/qiita_org/' ディレクトリに小分けしたプログラムを，'lib/qiita_org.rb' にコマンドを実行するための記述を thor を用いて行なった．

2.2 thor

コマンドの作成に使われる CLI builder gems にはいくつかの種類があるが，その中でも thor はダウンロード数が 2 番目に多く，広く使われている [4]．そのため，本研究の根幹では thor を用いてコマンドの管理をすることとした．

thor[5] とは Ruby のコマンドラインインターフェース (CLI) ツールの作成を支援する gem である．thor クラスを継承し，メソッドを定義することで該当するメソッドがコマンドになる．例えば入力した名前に対して挨拶するコマンドを作成するには以下のようにする．

```
require 'thor'

class CLITest < Thor
  def initialize(*argv)
    super(*argv)
  end

  desc "hello name", "Greeting to [name]"
  def hello(*name)
    name = (name[0] or "world")
    puts "Hello #{name}."
  end
end

CLITest.start(ARGV)
```

このプログラムを実行すると以下のように表示される．

```
> ruby thor_test.rb
Commands:
  thor_test.rb hello name      # Greeting to [name]
  thor_test.rb help [COMMAND]  # Describe available commands or one specific ..
```

このようにコマンドがプログラム上の desc(description) に記載した内容で表示され，Commands の説明通りに実行することでコマンドが使用できる．本研究ではこれを用いて，qiita_org のコマンドを作成した．

2.3 Qiita と Qiita:Team

本研究の目的のひとつは研究室内の情報共有を円滑に行うことである．今回は Qiita という情報共有サービスを用いることとした．Qiita とはプログラミングに関する知識を文章や，シンタックスハイライトされたコードを記事として投稿できるプログラマのための技術情報共有サービスである [6]. しかし，Qiita は個人が記事を載せるサイトであり誰にでも閲覧できるサイトである．限定共有投稿という公開されていない記事を作成することもできるが，共同体の中で記事を共有するには，投稿するたびに URL を知らせるなどしなければならない．

そこで会社や組織等のメンバーで投稿をまとめる機能を持つ，Qiita のサービス，Qiita:Team[7] や Qiita Organization[8] を使用することを考えた．表 2.1にこの二つの違いをまとめた．Qiita:Team は社内向けに利用するサービスであるが，Qiita Organization は社外に向けた自社の能力のアピールに利用する事が多い．その為，Qiita Organization は 1 社につきひとつしか作れないが無料であり，Qiita:Team は用途に合わせ有料で複数作成できる．また，記事は Qiita Organization は通常の Qiita 同様全ユーザーが閲覧できる為，記事内容に審査があり coding に関することのみであるが，Qiita:Team は閲覧可能者は参加しているメンバーのみなので記事内容に審査がなく議事録や日報にも使用できる．

表 2.1: Qiita:Team と Qiita Organization の違い．

	Qiita:Team	Qiita Organization
料金	有料 (1,490~)	無料
閲覧者	メンバーのみ	全ユーザー
記事内容	なんでも	coding に関すること
作成時の審査	なし	あり
その他	1 社につき複数作成可	1 社につきひとつ

研究室で使用するためには，

- ゼミの議事録など coding 以外の記事を扱いたい
- 個人情報やセキュリティ上の問題で非公開が望ましい

主に上記のような条件がある．Qiita Organization は純粋に会社や組織の coding 能力をアピールするために使うことが多く，基本は Qiita と同様公開された記事で構成されているため，研究室内の情報を共有するには秘匿性がなく不向きである．そこで，今回は Qiita:Team を西谷研究室の情報共有のために使用することとした．

2.4 Qiita API

Qiitaとの通信を行うにはQiitaが提供しているAPIを使用する．Qiita APIはQiitaの提供しているサービスからデータを取得，投稿したり，ストックなどの操作のできるWeb APIである．本研究ではQiita API v2ドキュメント[9]を参考にコマンドを作成した．

2.5 ox-qmd

2.5.1 ox-qmdとは

ox-qmdとは'0x60df'という方が作成したOrg-modeのテキストをQiita準拠のMarkdownに変換するemacs上のパッケージである[10]．前提として，Qiitaへ記事を投稿するためにはMarkdownの形式でテキストを用意する必要がある．本研究ではemacsのOrg-modeを使用するためこれをMarkdown形式に変換する必要があった．Org-modeのテキストを Markdownに変換する方法として代表的なものは，

- emacs Org-mode標準組み込みのMarkdown変換機能，
- 多様なフォーマット間の変換を行うツールの'pandoc'[12][13]，

などがある．しかし，これらの方法でMarkdownへの変換を行うと標準の規格に準拠したMarkdownが生成される．これではQiitaに投稿するためのMarkdownとしては少し不十分であった．例えば，

- Qiitaにはサイト上で目次が生成されるため，Markdownテキスト上に目次を生成する必要がない，
- コードの表示部分の文頭にスペースが置かれることで，Qiita上でコードのシンタックスハイライトが効かない，

などの問題があり，Qiitaで綺麗な記事として表示できない．そこで今回はQiita投稿用のMarkdown変換を主目的としているox-qmdを使用することとした．

2.5.2 使用方法

0x60df氏のGithub[11]にox-qmd.elというプログラムが公開されている．このプログラムを取得し使用する．emacs上で使用できるようにするには以下のことを行う．

- ox-qmd.elをemacsの設定ファイル(.emacsや.emacs.dなど)ディレクトリに用意する
- init.elなどの設定プログラムにox-qmdのパスを通す

例えば.emacs.dの中にinit.elなどがある場合は，ox-qmdといったディレクトリを作成し，そこにox-qmd.elを用意する．そしてinit.elにパスを通す．init.el内に

```
(setq load-path (cons "~/.emacs.d/ox-qmd" load-path))
(require 'ox-qmd)
```

と記載する．これらを行うことで，Org-mode 標準の export 用メニューから実行できるようになる．コマンドは'Ctrl-c, Ctrl-e, 9, 9' とすれば，Org-mode のテキストから Markdown が生成される．

本研究ではこのように手動で実行するのではなく，batch で外部から ox-qmd.el を呼び出すことで，Org-mode のテキストを変換し使用する．

第3章　各コマンドの設計と考察

3.1　post

post コマンドは作成した Org-mode のテキストを Qiita に投稿するためのコマンドである．コマンドを実行することで，作成したテキストを Qiita 投稿用の Markdown に変換し，投稿する．また投稿先を公開投稿や限定投稿，QiitaTeam から選択しそこへ投稿することが目的である．

Terminal でコマンドを実行してから Qiita へ投稿する流れは図 3.1に示す．

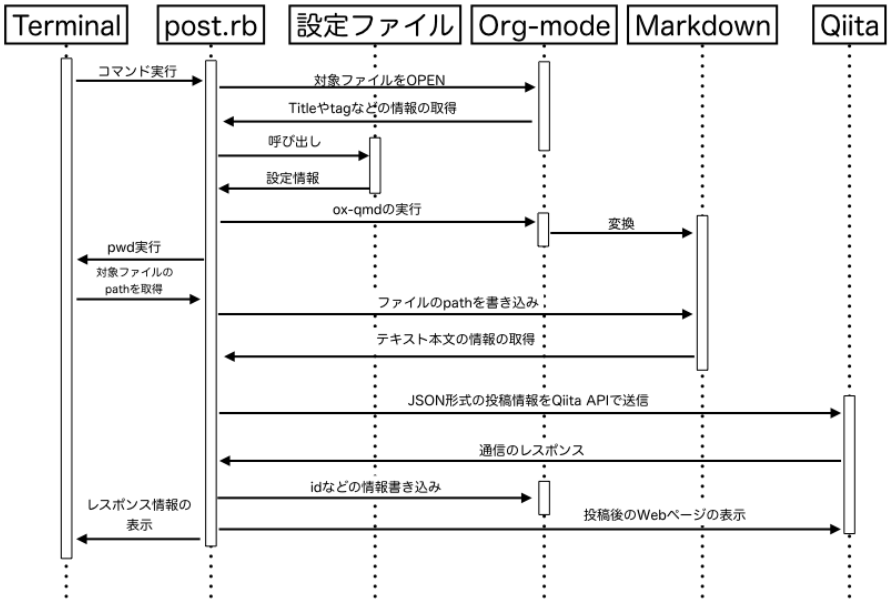

図 3.1: 投稿の流れ.

それぞれ，ターミナル，設定ファイル，Org-mode，Markdown，Qiita が役割を持ち，その間を post.rb が橋渡しをしている．ターミナルでコマンドを呼び出し，post.rb が実行されるとまず記事の元データである Org-mode 内のヘッダーから Title や tag などの情報を，設定ファイルからは，Qiita へアクセスするための設定情報を取得する．次に ox-qmd を実行し Org-mode テキストを Markdown 形式に変換を行う．生成された Markdown テキストに'pwd' をターミナルで実行することで取得した，カレントディレクトリの PATH を書き込み，その後本文情報として post.rb 内に取得する．これまでに取得した情報を JSON 形式にまとめ Qiita API を用いて Qiita との通信を実行．通信のレスポンスを受け取り，その中から記事 ID は Org-mode テキストに書き込み，URL 情報を使い投稿後の Web ページを表示する．その後ターミナルにレスポンス情報を表示して終了する．

以下に詳しいプログラムの内容を示す．

3.1.1 コマンドの引数

コマンドの引数には，投稿する Org-mode のテキストファイル名と投稿先を指定するオプションを入力する．ファイル名を'src'，投稿先を'option' として post コマンドのプログラムで使用する．

また，thor で受け取ったターミナルからの入力を整理し，拡張子'.org' でファイルを個別に識別，順次 post コマンドを実行する関数'get_multiple_files' を用意することで，アスタリスクなどを用いた複数のファイルの投稿にも対応できるようにしてある．

3.1.2 ファイルの読み込み

Org-mode テキストの読み込みには，Ruby に用意されている'File' クラスを用いる．ファイルを読み出すには

```
conts = File.read(src)
```

とし，'conts' に Org-mode テキストの内容を保存している．

3.1.3 Title と Tag の取得

Qiita への投稿にはタイトルとタグが必要である．これがなければ API を使い投稿した際に'Bad request' や'Forbidden' などのエラーが起き投稿できなくなる．

タイトルとタグの情報取得には正規表現を用いる．'qiita_org' のテンプレートとしてヘッダーにタイトルとタグを表記する場所を用意しているのでその文章を正規表現でプログラムで読み出す．

```
conts.match(/\#\+(TITLE|title|Title): (.+)/)
conts.match(/\#\+(TAG|tag|Tag|tags|TAGS|Tags): (.+)/)
```

'(.+)' の部分に Org テキストに記入した情報が取得でき，これを整理し，変数'title', 'tags' と保存する．

3.1.4 Markdown への変換

Markdown への変換には ox-qmd を用いる．ox-qmd を batch で外部から呼び出し，Org テキストを Markdown テキストに変換する．ox-qmd 呼び出しのコマンド実行には'command_line' という gem を用いて実行，実行時のレスポンスを受け取るようにした．

```
command = "emacs #{src} --batch -l #{ox_qmd_load_path} -f
  org-qmd-export-to-markdown --kill"
res = command_line command
```

'src' にはファイル名，'ox_qmd_load_path' には設定ファイルから読み取った ox-qmd.el の場所が入る．

3.1.5 Markdown テキストの編集

どこの Org テキストから投稿したのかは，時間が経てば忘れてしまうことがある．そこで，Qiita の記事上でどのディレクトリに元テキストがあるかを確認出来るように，Markdown テキストの最後の行に元ファイルの PATH を記述する．プログラムは

```
lines = File.readlines(src.gsub(".org", ".md"))
path = Dir.pwd.gsub(ENV["HOME"], "~")
lines << "\n\n------\n - **source** #{path}/#{src}\n"
```

としており，保存した元ファイル名の拡張子を'.md' とし，Markdown テキストを行ごとに配列で読み込む．そしてその配列の一番後ろに元ファイルの PATH を追加している．

3.1.6 投稿方法の確認

Qiita API の記事送信方法には新規記事作成と更新の 2 種類がある．ここでは Org テキストから ID 情報があるかどうかを確認し，新規作成と更新を設定している．

Org テキスト内に記事の ID の記載があるかを正規表現で確認し，ある場合は更新を選択する．

```
m = []
if m = conts.match(/\#\+qiita_#{option}: (.+)/)
  qiita_id = m[1]
  patch = true
else
  qiita_id = ""
end
```

patch の真か偽を設定することで更新か新規投稿かを設定する．

また，Qiita サイト上で限定共有投稿の記事を公開投稿記事に変更する場合がある．このとき'qiita_org' を使い，投稿すると Org テキスト上には限定共有記事の ID しか記載されていないため，公開記事は無いとエラーでコマンドが終了するか，同じ内容の記事が新しく投稿されてしまう．この事例を防ぐため，入力された'option' が限定記事であるなら，一度 Qiita にアクセスし，公開記事に変更されていないかを確認するようにしている．

3.1.7 Qiita への送信

Qiita への送信には ruby の Net:HTTP ライブラリと Qiita API をを用い，今までに取得した投稿用の情報，Markdown テキストを本文とし送信する．

投稿のための URL は API ドキュメントから'https://qiita.com/api/v2/items' と設定し，設定ファイルから取得した Qiita アクセストークンを入れ投稿する．

3.1.8　記事 ID の書き込みと Markdown テキストの削除

Qiita の記事には個別に ID が付いている．これを Org テキスト編集後，投稿しても新規に記事を作成せず記事を更新できるように，Org テキストに記述する．ID は Qiita へ送信した後のレスポンス情報から抜き出し変数に保存している．記事の投稿が更新であればこの作業は行わない．

また，ox-qmd で作成した Markdown テキストが残っていると，Org テキストを編集し投稿した際に，文法が間違っているなどのエラーで，Markdown の生成が行われなかった場合でも投稿ができてしまう．これが起こらないように post プログラムの最後で Markdown テキストを削除する．

3.1.9　結果と改善案

このコマンドを使うことで，ターミナル上のみで，Qiita への投稿が可能になった．したがって，本研究の目的である，’Org-mode のテキストを Qiita へ投稿する’ は達成していると言えるだろう．

しかし，コマンド実行後ターミナル上に表示される投稿に関する情報が図 3.2 にあるように user 情報が少し見辛い形となっている．

```
       rendered_body brabrabra...
                body brabrabra...
            coediting false
      comments_count 0
          created_at 2020-11-12T15:31:49+09:00
               group
                  id 2e74fa93d03585847510
         likes_count 0
             private true
     reactions_count 0
                tags [{"name"=>"test", "versions"=>[]}]
               title test
          updated_at 2021-01-17T15:34:06+09:00
                 url https://qiita.com/yamatoken/private/2e74fa93d03585847510
                user {"description"=>nil, "facebook_id"=>nil, "followees_count"=
>0, "followers_count"=>0, "github_login_name"=>"yamatoken", "id"=>"yamatoken", "
items_count"=>4, "linkedin_id"=>nil, "location"=>nil, "name"=>"", "organization"
=>nil, "permanent_id"=>612049, "profile_image_url"=>"https://avatars0.githubuser
content.com/u/49702981?v=4", "team_only"=>false, "twitter_screen_name"=>"k0603ya
mamot", "website_url"=>nil}
```

図 3.2: post コマンドの実行レスポンス．

今後の課題としては，この user 情報を整理して表示，例えば’permanent_id’ などの通常使用者が気にしない情報は表示しないなどがある．

3.2 upload

Qiita 上に図等の画像を挿入するためには，サイト上にてドラック&ドロップを行い URL を発行する必要がある．Org-mode の記法で図を挿入するだけでは post コマンドを実行した際に URL は発行されず，Qiita サイト上に表示されない．upload コマンドではこの問題を解決するべく，post とは別のコマンドとして，サイトに画像情報を送信し，サイト上に図の表示を，Org-mode テキスト上にはサイトから URL を取得し記載を行えるようにすることが目的である．

目的を達成するため私は 2 種類の方法を検討した．

1. Selenium を使い完全自動で画像を投稿．
2. 画像を Qiita にドラッグ&ドロップする部分だけ手動にする．

結論としては，2 案の’ 手動を組み合わせた手法’ を採用した．採用理由としては総実行時間を比較したところ，2 案の方が短いためである．以下にそれぞれの手法の詳細と比較を記す．

3.2.1 Selenium

Selenium とは Web アプリケーションのテストを自動化するために開発されたツールである [14][15]. 現在はテスト以外にもタスクの自動化や Web サイトのクローリングなどの様々な用途で利用されている．

Selenium を用いると，人が Web ブラウザ上で行う作業のほとんどをプログラムで指定することで実行することができる．今回は一つ目の手法として Selenium を使い Google Chrome を動かすことで Qiita へ画像を投稿を行なった．

Selenium を用いた自動で画像を投稿する流れは図 3.3に示す．

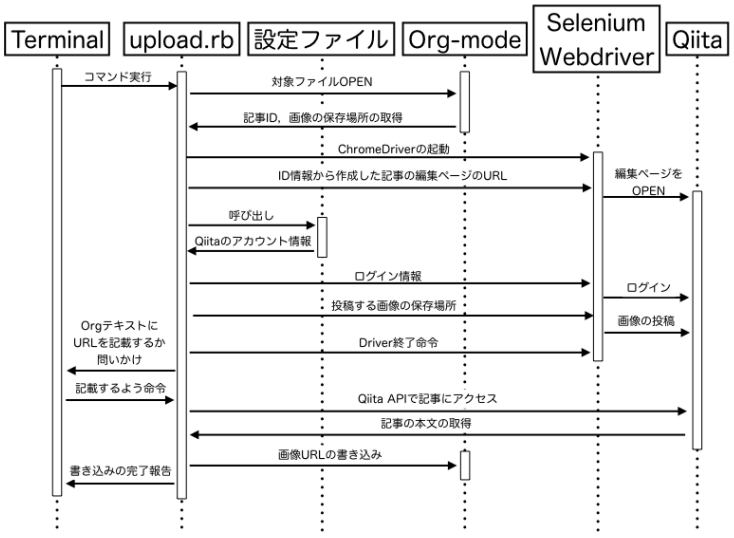

図 3.3: Selenium を用いた画像投稿の流れ．

Selenium では Web ブラウザのボタンの位置や要素の属性を指定することで目的の行動を行うことができる．ボタンの位置は Chrome の開発者ツールを開き (Mac では Ctrl+option+i で開く)，HTML を見ることで確認できる．以下は Selenium を用いて Qiita へ画像を挿入するプログラムの例である．

```
require "selenium-webdriver"

#Chrome 用のドライバの設定
driver = Selenium::WebDriver.fo :chrome
#Qiita の記事編集画面にアクセス
driver.get "https://qiita.com/drafts/REPORT_ID/edit"
sleep 3
#Qiita のアカウントに Github アカウントでログイン
element = driver.find_element(:class, "fa-github")
element.click
sleep 3
#アカウント情報の送信, ログインボタンのクリック
element = driver.find_element(:name, "login")
element.send_keys "your github name"

element = driver.find_element(:name, "password")
element.send_keys "your github password"

element = driver.find_element(:name, "commit")
element.submit
sleep 3

#画像の PATH の送信
element = driver.find_element(:xpath, "/html/body/div[1]/div[3]/div/div[3]/
       div/div[1]/div[1]/div[2]/div[1]/div/input")
element.send_keys "画像の PATH"
sleep 5

#更新ボタンのクリック
element = driver.find_element(:class, "e9bkwgs1")
element.click

sleep 15 #15 秒後 Driver を終了.
```

このプログラムは Qiita 編集画面を開き，Qiita へ Github アカウントを用いてログイン．その後画像の PATH を送信することで画像を投稿，URL の発行を行なっている．

upload コマンドとして利用するためには，上記のようなプログラムで URL を発行，Qiita API を用いて本文情報を取得しその中から URL を抜き出し Org テキストに記述．その後，post コマンドを用いることで意図した位置に画像を挿入するようにした．

3.2.2 cui と gui の組み合わせ

次に多少の手作業が発生してもドラッグ&ドロップ程度の作業であれば苦にならないのではないかと考えた．そのため二つ目の手法として，プログラムで画像の保存してあるフォルダと Qiita の記事ページを表示して，画像をドラッグ&ドロップする部分だけは手作業で行うこととした．

図 3.4にドラッグ&ドロップ作業を手動で行う画像投稿の流れを示す．

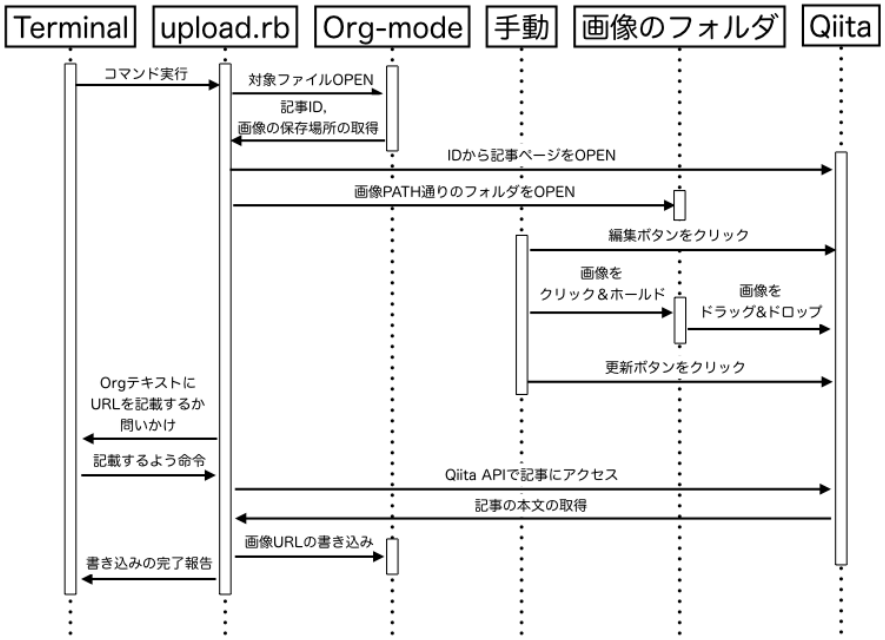

図 3.4: 手動部分を含む画像投稿の流れ．

プログラムの流れとしては，目的の Org テキストから記事の ID 情報と画像の PATH を取得．この二つの情報からターミナルの'open' コマンドを実行．デスクトップ上に画像のフォルダと Qiita の Web ページを表示する．ここで一旦手作業の部分を行い，再度ターミナルに戻った際に続きを開始するようにしている．

3.2.3 upload 手法の比較

今回とった手法のそれぞれの利点と欠点は表 3.1に示す．表 3.1にあるように，Selenium を用いると自動で画面が遷移するのを眺める時間が発生し，実行時間が長くなり大幅に作業効率が悪くなる．手動を組み合わせた手法の方は画面を眺める時間の中で作業を終わらせることができる．

表 3.1: 二手法の利点と欠点.

	Selenium	cui+gui
利点	完全自動でテキストの文法があっていればミスがない.	一部手動で行うことで作業時間が速い.
欠点	Web の待機時間を指定しているため完了までに時間がかかる.	ドラッグするファイルを間違えるなどのミスが発生する場合がある.

Selenium は完全に自動で行えることは大きな利点ではあるが，コマンドの総実行時間を考慮し，今回は cui と gui を組み合わせた手法を upload コマンドとして採用した.

今後 Selenium のように完全自動で行える，かつ実行時間が高速な手段を模索し実装することが課題となるだろう.

3.3 config

設定ファイルを使用者が各自手動で作成すると，手間もかかる上予期せぬエラーが起こることがある．そこで config コマンドでは'qiita_org' を使用するのに必要な設定ファイルを比較的簡単にコマンドを実行することで設定できるようにすることを目的とした.

3.3.1 設定ファイルの雛形と処理方法

人にも見やすくかつプログラムで呼び出しやすい形にするために，JSON 形式で設定を書き込むこととした.

設定ファイルの雛形は以下に示す.

```
{
  "name": "",
  "email": "",
  "access_token": "",
  "teams_url": "",
  "display": "open",
}
```

Ruby 標準ライブラリの JSON モジュールを用い設定ファイルへの書き込みや読み出しを行なった.

```
require 'json'

items = JSON.load("設定ファイルの PATH")
```

上記のように JSON 形式のファイルパスを使いファイルを読み出す．これにより items に入った情報の key を指定することでそれに対応した値を変更することができる．例えば，’items[”name”]’ と指定すれば,name の値を変更できる．

これを用いてコマンド呼び出しの際に，key を指定し設定ファイルの変更を行なった．

変更後のファイル書き出しについては下記に示す．

```
conts = JSON.pretty_generate(items)
File.write("設定ファイルの PATH", conts)
```

items を’pretty_generate’ を用いることで，改行やインデント整え，その後ファイルへ書き込みを行なっている．

3.4 template

’qiita_org’ では Qiita に投稿するために様々なヘッダーを使用している．例えば

- 記事のタイトル
- 記事の著者
- Qiita に表示するタグ情報
- Twitter に投稿をするかどうか

などである．このヘッダーの書き方をを全て覚えて記事を作成するたびに自ら書き込むのは困難である．そこでこの template コマンドでは，これらを記載したテンプレートテキストを容易に作成することを目的とした．

3.4.1 用意されるテンプレートの内容

ヘッダーの雛形は以下に示す．

```
#+OPTIONS: ^:{}
#+STARTUP: indent nolineimages
#+TITLE: title
#+AUTHOR: Your Name
#+EMAIL:     (concat "hogehoge@example.com")
#+LANGUAGE:  jp
#+OPTIONS:   toc:nil
#+TAG: hoge, hoge2
#+TWITTER: off
```

この雛形が書かれたファイル'template.org' を'qiita_org' 内に用意し，コマンドが実行されるとこのファイルを呼び出されたディレクトリにコピーする．

また，Qiita の記事には自分の作業している環境を書くことが多いため，雛形をコピーする際に環境を書き込むこととした．

環境の表示には'Shields IO'[16][17] を使用する．'Shields IO' を使うことでバッジの画像が取得できる．Qiita 記事上で使用するには，

```
![<LABEL>-<MESSAGE>](https://img.shields.io/badge/<LABEL>-<MESSAGE>-<COLOR>)
```

と表記する．本研究では'LABEL' に環境の名前，'MESSAGE' にはバージョン情報を表示させ使用した．サイト上には図 3.5のように表示される．

図 3.5: Qiita 上での環境表示の例．

環境情報は，ターミナルでコマンドを実行することで取得する．例えば MacOS であれば，

```
system "sw_vers > ver_info.txt"
```

とプログラムで実行することでテキストファイルにバージョン情報を記載し，その中から正規表現で目的の情報を取得している．

3.4.2　環境記載についての改善案

現在テンプレート取得時に書き込むことのできる環境情報は，'MacOS', 'Ubuntu', 'Ruby' のバージョンだけである．西谷研では Ruby を主に用いるため現状はこの形になっているが，Qiita に書き込む記事の内容をこのバージョンだけでカバーはできない．今後は多様な内容の記事に対応できるように記載できる環境情報を増やしていく必要があるだろう．しかし，各環境ごとに関数を作成していてはプログラムが大きくなってしまう．そのため単に内容を増やすだけでなく，プログラムの方法も模索する必要がある．

3.5　list と get

list コマンドは自分の投稿した記事のタイトルや URL などの情報をターミナルに表示させることを，get コマンドは Qiita 上の記事をディレクトリ内に Org テキストとして保

存することを目的とした．Qiita に投稿した元の Org テキストの場所を忘れた際や，Qiita 上の記事を Org テキストとして保存したい場合に，役立つであろう．

3.5.1 記事取得の手法

list コマンドは Qiita にアクセスすることで，最新から 100 件までの記事の情報を取得しターミナル上に表示する．表示される内容は，タイトルと記事の URL，元 Org ファイルの PATH である．

また，get コマンドには以下の 2 種類の動作がある．

- 自分の記事一覧から選択したものの取得
- 指定した ID の記事のみを取得

まず，自分の記事一覧から取得する記事を選択し取得する流れを図 3.6に示す．

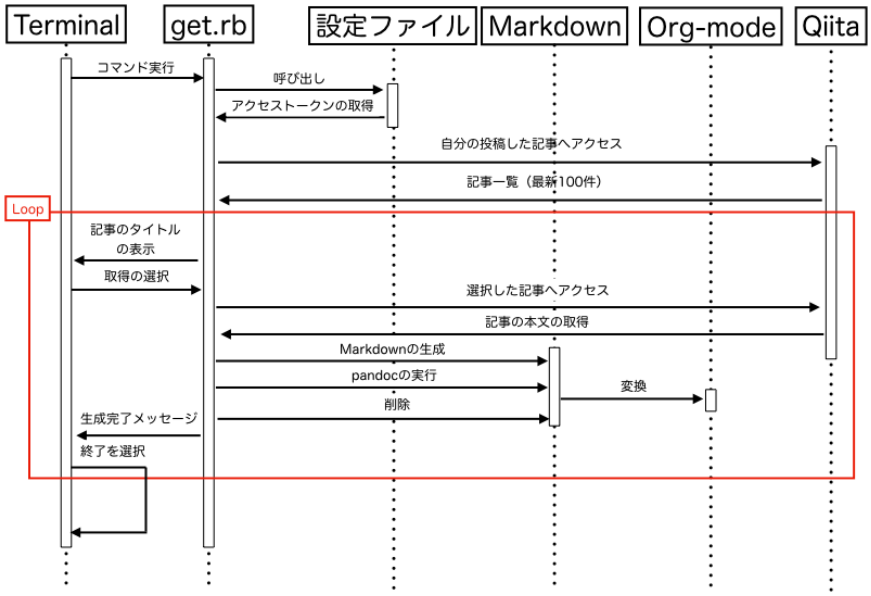

図 3.6: 複数の記事の中から取得する記事を選択する流れ．

コマンドを実行すると，get.rb が設定ファイルより Qiita へのアクセスに必要な情報を取得する．その後，自分の記事の一覧情報を Qiita から Qiita API で受け取り，最新よりタイトルをひとつずつ表示し，ディレクトリ内に保存するかどうかをターミナルから入力する．保存する場合はその記事にアクセスし，取得した本文情報から Markdown の生成を行う．その後 pandoc で Markdown を Org-mode 形式に変換したのち，不要になった Markdown テキストの削除を行い，生成完了メッセージをターミナルに表示している．この後，ターミナルで終了指示を行うまで取得した全ての記事について，同じ動作を繰り返す．

この取得記事を選択する場合と list コマンドの動きは図 3.6のループ前までが共通しており，ループに入る前にターミナル上に一覧を表示をし終了するか，一覧の表示を行わず，ループに入り先述の手順を行うかの違いである．

また，ID を指定した場合は，一覧を取得せずに ID の情報から直接記事にアクセスし，その記事を取得している．

3.5.2　コマンドの利点と改善案

このコマンドを Qiita::Team の記事に対して実行すると自分の記事だけでなく，参加しているメンバーの記事も一覧として取得できる．ターミナル上で記事の概要の確認を行えることや，Org-mode のテキストとして保存できるようになり，記事の確認を行いやすくなり，より情報共有が円滑になった．

しかし，Github の issue で指摘を受けたように，Qiita API の仕様上記事の取得は最新から 100 件しか取得できない．つまり，自分の記事の探索をする際に目的の記事が 101 件目であれば，ターミナル上で見つけ出すことができない．

探したい記事があればその記事のタイトルなどは断片的になら覚えているだろう．そこで今後は’–title’ や’–tag’ などのオプションをつけ，単語を指定することで，より早く目的の記事を表示できるようにすることが課題である．

3.6　all

投稿した記事が複数ある場合それをひとつずつ投稿するのは，手間もかかり時間の無駄だ．そこで all コマンドはディレクトリ内全ての Org テキストを一括で投稿するために作成した．

3.6.1　プログラムと仕様について

all コマンドのプログラムは以下のようである．

```
files = Dir.glob(".org")
p files
files.each do |file|
  unless mode
    puts file.blue
    if File.read(file).match(/#\+qiita_(.+)/)
      system ("qiitapost #{file} open") if File.read(file).match
       (/#\+(.+)_public/)
      system ("qiita post #{file} teams") if File.read(file).match
   (/#\+(.+)_teams/)
      system ("qiita post #{file} private") if File.read(file).match
(/#\+(.+)_private/)
    else
      system ("qiita post #{file}")
```

```
    end
  else
    puts "qiita post #{file} #{mode}".blue
    system "qiita post #{file} #{mode}"
  end
```

all コマンドでは投稿先を指定すれば指定先だけに，指定がなければ，Org テキストから正規表現で ID 情報を読み出し記事を更新，ID の記載がない新規投稿であれば限定記事として投稿する形をとった．

また，投稿したくないファイルがディレクトリ内に存在する場合を考慮し，’–exclude’ とオプションをつけファイル名を指定するとそのファイルは除外して投稿するようにした．

3.6.2 all 実行時の問題

ディレクトリ内のすべてのファイルが正しく書けているとは限らない．現状では，all コマンド実行時に 1 つでもエラーが出るとそこでエラー文を表示しコマンドが終了してしまい，それ以降のファイルが投稿されない．今後はエラーが出た場合コマンドが終了するのではなく，最後まで実行し一番最後にエラーの出たファイルについての情報をターミナルに表示するようにプログラムを変更することが課題となるだろう．

第4章　Githubによるissueおよびpull request

Github にはユーザフィードバックの収集やバグの報告を行うことのできる issue や pull request という機能がある [18][19]．本研究では作成した qiita_org を西谷の大学院の授業の受講生に試用してもらい，issue や pull request を収集し，qiita_org の改善を行なった．

4.1　issue

qiita_org への issue は合計で 14 個あった．以下は実際にきた issue の一部である．

- バグの報告.
- Qiita の Twitter 連携をコマンドでも使用できるようにしてほしい.
- qiita_org 内の gems ディレクトリの必要性.
- all コマンドの仕様変更案とオプション追加案.

上記のような意見を受け取り，機能の修正や追加を行った一部を以下の節に示す.

4.1.1　TwitterとQiitaを紐づけし共有する

Qiita では Twitter アカウントと連携しておくことで，記事を投稿すると Tweet として記事をアップする機能がある．この機能を生かすために qiita_org を使い投稿した内容も Twitter で表示できるようにしてほしいと issue を受けた [20].

機能の追加のために行ったことは以下である.

- 記事テンプレートのヘッダーに Twitter に関する記載を追加
- post 時に送信する JSON 情報に Twitter に関するオプションの追加

まず，記事テンプレートに'#+Twitter' と情報を記載した．記載することでこの文をプログラムで処理し実際に Qiita へ送信する情報を決めることにした．Qiita API ドキュメントによると Twitter に投稿するかは API で送信する JSON 情報に'tweet' 項目を追加することでできるとあった．Twitter に投稿するには

```
"tweet": true
```

と JSON 情報を追加すればできる．post 内での処理は以下のような関数を作成し行った．

```
def select_twitter(conts, option)
  m = []
  twitter = false
  if option == "public"
    m = conts.match(/\#\+(twitter|Twitter|TWITTER): (.+)/)
    if m[2] == 'on' || m[2] == 'On' || m[2] == 'ON'
      twitter = true
    else
      twitter = false
    end
  end
  twitter
  return twitter
end
```

この関数では，Org-mode テキストのヘッダーから'#+Twitter' の記載を正規表現で取得し，内容が'On' であれば true に，'Off' なら false としている．送信する JSON に識別した情報を記載することで，この issue で受けた，Twitter との連携を行えるようにした．

4.1.2 qiita_org 内の gems ディレクトリについて

qiita_org の中には gems ディレクトリが存在していた．このディレクトリは qiita_org で関連している他の gem についての記載がされているが，このディレクトリがあることで以下のような問題が起こる可能性があると指摘を受けた [21]．

- 開発している他の gem と開発環境が混同する
- 関連する gem が変更された場合変更が追えない
- 作成している gem のデータが想定以上に大きくなる

大抵の場合この gems ディレクトリは意図せず追加され上記のような問題が起こる．複数の gem を開発している人であればなお起こりやすい．

一般的には開発環境の中で使用する gem は，rbenv や bundler で管理しており，関連する gem はここに PATH を通し使う．qiita_org ではファイル群の中に'.bundle' というものがあり，この config に

```
BUNDLE_PATH: "gems"
```

と記載されていたため，gems ディレクトリから関連する gem を使用していたので，結果的に gems ディレクトリが必要となっていた．

本研究ではこの gems ディレクトリは意図せず追加されていたディレクトリであったため，これを削除することにした．

しかし，単純に問題になっている gems ディレクトリを削除するだけでは'.bundle' で PATH を指定しているにも関わらず，gems ディレクトリがなくなるので，関連 gem が使えず，コマンドが動かなくなる．そのため，'.bundle' を削除した後，gems ディレクトリを削除することでこの問題を解決した．

4.1.3 未解決の issue について．

解決していない issue としては

- post 実行後更新であっても新しくブラウザのタブが開かれる [22]
- list コマンドで記事を検索できるようにする [23]

などがある．

新しくブラウザのタブが開かれる問題は，MacOS 以外の OS でこの問題が起きている．MacOS の場合 Safari で Web ページがもともと表示されていればページの更新を行うだけで新規にタブが開かれることはない．しかし Windows や Linux だとデフォルトのブラウザが違うためこの問題が起きていた．この問題が起きていると確認できているブラウザは Firefox，Chrome，Microsoft Edge である．設定ファイルに投稿後，ブラウザを開くかどうかを指定する欄を設けることで解決を図った．

```
> qiita config global display 'suppress'
```

と実行すると

```
"display": "suppress"
```

と設定ファイルに書き込まれ，投稿後の Web ページ表示を抑制できる．しかし，投稿後しっかり投稿できているか確認するためにも Web ページが表示されることは重要であると考える．今後はプログラムで新たなタブが開かないようにする方法を模索・実装する必要がある．

また，list コマンドは 3 章 5 節 2 項に示したように，'–tag' などのオプションを追加することで検索を可能にすることで解決することができるだろう．

4.2 pull request

qiita_org には 9 個の pull request があり，内 7 個は西谷からである．9 個全てを採用している．西谷以外の pull request は

- template コマンドの仕様変更案 [24]
- Twitter 投稿に関する関数のバグ修正 [25]

である．以下にこの 2 つの pull request について示す．

4.2.1 template コマンドの仕様変更

template コマンドは実行すると，実行したディレクトリ内に'template.org' という記事投稿用のテンプレートが用意される．もし実行時にディレクトリ内に'template.org' が存在していた場合，重複して作成は行われず終了するようになっていた．

コマンドを実行するたびにファイル名を変更するのは不便であると，コマンド実行時にファイル名を指定することができるように，変更した pull request があった．

この案を採用したが，拡張子'.org' を記載しない形でのみ実行できるようになっていた．しかし，'mv' などでファイル名を変更する際は拡張子を書くことから，拡張子があってもコマンドを実行できるようにするべきと考え，拡張子をつけてもつけなくても実行できるように変更した．

4.2.2 Twitter 投稿のバグについて

issue での意見を元に私が作成した Twitter 投稿用の情報を取得する関数では，Org-mode テキスト内に'#+twitter' が無いと異常終了するようになっていた．

この問題を解決した上で，関数を綺麗に整理した pull request だったのでこれを採用した．プログラムは

```
def select_twitter(conts, option)
  option == "public" && conts.match?(/^\#\+twitter:\s*on$/i)
end
```

となっており，元々12 行使い書いていた関数が 1 行で収められている上に，バグが修正されている．このようなコーディングの方法があると知ることができ，大きな学びとなった．

4.3 issue と pull request の重要性

本研究で私は issue と pull request により多くの学びを得た．issue では見つけることのできなかったバグや改善案をもらうことにより qiita_org の向上に，pull request ではプログラムに関する新たな知識やコーディングのテクニックを学ぶことができた．

今後はより多くのユーザに qiita_org を使用してもらうことで，意見を吸い上げ，改善することで qiita_org がより使いやすい gem に成長することができると考える．

第5章　総括

本研究で作成した'qiita_org' を研究室に取り入れ，記事の作成から投稿を日常的にターミナルを用いて行うことで，研究のメモを効率よく残せるようになっただけでなく，プログラマの必須技能であるターミナル操作能力の向上につなげることができた．そして，Qiita:Team を使用することにより研究室内の情報共有がより円滑になり，自らの，または後続の研究の手助けとなるシステムを作ることができたと考える．

今回の研究では，今まで時間がかかっていた細かな部分の情報共有を記事として文字媒体で残すことで効率化することができた．今後も研究室内でこのシステムを使い，研究の手順，成功や失敗の記憶を残していくことで，再利用性・汎用性の高い情報を集約できる環境を作ることができると考える．

このシステムは，日々得られる経験や技術を記事として周囲と共有することで，お互いの技術向上につながる環境作成に利用して頂きたい．そして，未来のより良い研究につながるよう多くの共同体に取り入れてもらえることを期待している．

謝辞

本論文の作成にあたり，多くの方々にご支援いただきました．
先ず，貴重なご指導とご助言を賜りました西谷滋人教授に心より深く感謝いたします．教授のご指導のおかげで，システムの仕様を定め研究を行うことができました．そして，研究の進行に伴い，様々な助力や助言，知識の供給を頂きました西谷研究室の同輩や，先輩方に深く感謝いたします．本当にありがとうございました．

参考文献

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

[2] Bundler とは - Qiita, https://qiita.com/io_fleming/items/14626a9cff44bc87e7db (accessd on 20 Jan 2021).

[3] すがわら まさのり, 寺田 玄太郎, 三村 益隆, 近藤 宇智郎, 橋立 友宏, 関口 亮一 - パーフェクト Ruby, 13 章 gem パッケージの作り方 (技術評論社, 2014).

[4] CLI Builder Gems - BestGems.org, http://52.198.30.45/categories/CLI+Builder (accessd on 20 Jan 2021).

[5] thor - Github, https://github.com/erikhuda/thor (accessd on 20 Jan 2021).

[6] Qiita とは - QiitaSupport, https://help.qiita.com/ja/articles/qiita (accessd on 20 Jan 2021).

[7] 生産を向上させる情報共有ツール-Qiita Team - teams.qiita.com, https://teams.qiita.com (accessd on 20 Jan 2021).

[8] Qiita Organization とは - Qiita Support, https://help.qiita.com/ja/articles/qiita-org-1 (accessd on 20 Jan 2021).

[9] Qiita API v2 documentation - Qiita:Developer, https://qiita.com/api/v2/docs (accessd on 20 Jan 2021).

[10] Org-mode から Qiita 準拠の Markdown を export するパッケージを作ってみました - Qiita, https://qiita.com/0x60df/items/3cde67967e3db30d9afe (accessd on 20 Jan 2021).

[11] ox-qmd - Github, https://github.com/0x60df/ox-qmd (accessd on 20 Jan 2021).

[12] Pandoc - Pandoc a universal document converter, https://pandoc.org (accessd on 20 Jan 2021).

[13] 多様なフォーマットに対応！ドキュメント変換ツール Pandoc を知ろう - Qiita, https://qiita.com/sky_y/items/80bcd0f353ef5b8980ee (accessd on 20 Jan 2021).

[14] Selenium History - selenium.dev, https://www.selenium.dev/history/ (accessd on 20 Jan 2021).

[15] 10分で理解する Selenium - Qiita, https://qiita.com/Chanmoro/items/9a3c86bb465c1cce738a (accessd on 20 Jan 2021).

[16] shields IO - https://shields.io (accessd on 20 Jan 2021).

[17] Qiita 記事にさりげなく（けど、わかりやすく）環境を記載する - Qiita, https://qiita.com/zizi4n5/items/c5f0013236fd055c3e02 (accessd on 20 Jan 2021).

[18] Issue について - Github Docs, https://docs.github.com/ja/github/managing-your-work-on-github/about-issues (accessd on 26 Jan 2021).

[19] プルリクエストについて - Github Docs,
https://docs.github.com/ja/github/collaborating-with-issues-and-pull-requests/about-pull-requests (accessd on 26 Jan 2021).

[20] 投稿した際に，Twitter と紐付けて Twitter で共有することができないか？ - Github issue #11, https://github.com/yamatoken/qiita_org/issues/11 (accessd on 26 Jan 2021).

[21] Is the vendor data needed? - Github issue #20, https://github.com/yamatoken/qiita_org/issues/20 (accessd on 26 Jan 2021).

[22] qiita post をする際に、ブラウザのタブが新しく開かれてしまう - Github issue #12, https://github.com/yamatoken/qiita_org/issues/12 (accessd on 26 Jan 2021).

[23] 記事の Title 検索 or Tag 検索 - Github issue #22, https://github.com/yamatoken/qiita_org/issues/22 (accessd on 26 Jan 2021).

[24] qiita template 時に任意のファイル名に - Github pull request, https://github.com/yamatoken/qiita_org/pull/16 (accessd on 26 Jan 2021).

[25] fix: twitter In-Buffer Settings - Github pull request, https://github.com/yamatoken/qiita_org/pull/14 (accessd on 26 Jan 2021).

[26] qiita_org manual - Qiita, https://qiita.com/yamatoken/items/2cefc503215afacc0b11 (accessd on 1 Feb 2021).

付録

以下に'qiita_org' のインストールから設定，コマンドの使い方を解説した Qiita の記事を添付しておく [26].

使用方法

インストール

terminal にて，gem を使うことでインストールすることができる.

```
> gem install qiita_org
```

インストール後，動作確認を行う.

```
    > qiita
Commands:
  qiita all [teams/public/private] [options]    # post all org files in the d...
  qiita config [global/local] [option] [input]  # set config
  qiita get [qiita/teams] [ITEM_ID]             # get qiita report
  qiita help [COMMAND]                          # Describe available commands...
  qiita list [qiita/teams]                      # view qiita report list
  qiita post [FILE] [private/public/teams]      # post to qiita from org
  qiita say_hello                               # say_hello
  qiita template                                # make template.org
  qiita upload [FILE] [teams/public/private]    # upload about image to qiita
  qiita version                                 # show version
```

このようにコマンド一覧が表示されていれば, インストール完了である.

設定方法

Qiita へ記事を投稿するには qiita_org のインストールだけでなく，コマンドで設定ファイルを作成し，以下のような設定をする必要がある.

- pandoc のインストール.
- 名前，email アドレスの登録.

- Qiita のアクセストークンを取得して登録.
- Qiita:Team の URL を登録（所属する Team がある場合）.

まず,pandoc という markdown のテキストを Org-mode のテキストに変換するパッケージを使用しているので，これをインストールする.

```
//mac の場合//
> brew install pandoc

//ubuntu の場合//
> sudo apt update
> sudo apt install pandoc
```

次に設定ファイルを HOME ディレクトリに作成する.

```
> qiita config global
```

もう一度同じように実行し，コピーされた設定ファイルの雛形を確認する.

```
> qiita config global
/Users/kentayamamoto/Github/yamatoken/.qiita.conf

"name": ""
"email": ""
"access_token": ""
"teams_url": ""
"display": "open"
"ox_qmd_load_path": "~/.emacs.d/site_lisp/ox-qmd"
```

次に，名前と email アドレスの登録を行う.

```
> qiita config global name 'Your Name'
> qiita config global name 'your_email@example.com'
```

アクセストークンは Qiita のサイトで発行する必要がある.
Qiita の web サイト上で図 1のように操作し，必要な「スコープ」にチェックを入れ発行する．ページを移動すると個人用のアクセストークン欄に発行されたアクセストークンが表示される．これをコピーし以下のように実行する.

```
> qiita config global access_token 'コピーしたアクセストークン'
```

最後に，Qiita:Taem に所属いているのではれば,

```
> qiita config global teams_url 'https://hoge.qiita.com/'
```

図 1: アクセストークンの発行方法.

と実行し登録する.

また，複数の Team に所属している場合や，ディレクトリごとにメールアドレスを設定したい場合があれば local の設定ファイルを指定できる．設定ファイルを作りたいディレクトリにて，

```
> qiita config local set
```

と実行し，その後は上記の設定と同様に書き込んでいくことで local の投稿設定ができる.

コマンドの使用方法

post

Qiita へ記事を投稿するためのコマンドである.

コマンドは'qiita post [投稿したい file] [option]'.

例えば，hoge.org というファイルを公開記事として投稿するには以下のように実行する.

```
> qiita post hoge.org public
```

option には'public','private','teams' があり，それぞれ公開記事，限定共有記事，Qiita:Team への投稿を表している.

また，すでに投稿済みの記事で Org-mode テキスト上に記事の ID が記載されているものに関しては，option を省略することができ，一番上に記載されている ID へ投稿される.

アスタリスクや xargs などを用いた複数のファイルの投稿にも対応しており，

```
> qiita post hoge*.org private
> ls -1 hoge*.org | xargs qiita post
```

などと実行すると hoge とついたファイルを一度に投稿することができる.

upload

Org-mode テキスト上に記載された図や写真を Qiita に表示させるためのコマンドである．
コマンドは’qiita upload [投稿したい file] [option]’
option については post コマンドと同様である．
コマンドを実行すると，投稿された Qiita の Web ページと図や写真が保存されているフォルダが図 2のように表示される．

図 2: upload 実行時にの様子．

Web ページ上に表示されている’編集する’をクリックし，編集画面に移動する．
移動後，図 3のように記事上の目的の場所にフォルダからファイルをドラッグ&ドロップする．URL が発行できたら．元々あった image を表示するための行を消し，ページ下部の更新ボタンを押し記事を更新する．

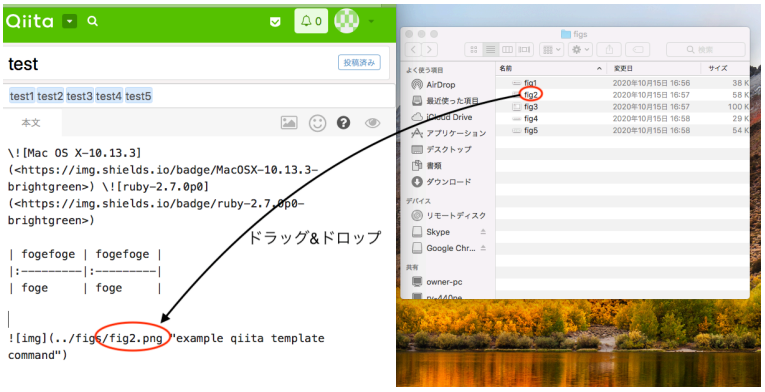

図 3: 画像の URL を発行する方法．

上記の作業を終えた後，ターミナル上に

```
Overwrite file URL's on test2.org? (y/n)
```

と表示されているので，’y’ を押すと Org-mode テキスト上に URL が記載され，これ以降記事を投稿した際 upload を行わなくても表示するようにできる．

template

Qiita 投稿用のヘッダーが記載された Org-mode テキストテンプレートを取得するためのコマンドである．

コマンドは’qiita template [ファイル名]’.

実行すると，OS 情報や Ruby のバージョンを記載するか聞かれる．’y’ を入力で記載，’n’ で記載しないことを指定できる．

ファイル名を指定すれば，そのファイル名通りの Org-mode テキストが作成される．指定しなければ’template.org’ が作成される．

get

コマンドを実行したディレクトリ内に Qiita 上の記事を Org-mode テキストとして保存するためのコマンドである．

コマンドは’qiita get [qiita/teams] [記事の ID]’.

option は qiita か teams で，それぞれ Qiita 上の記事，Qiita:Team 上の記事の取得を指定できる．

記事の ID を指定すれば，その ID 通りの記事が取得できる．

ID とは Qiita の Web ページ URL の一番最後に記載されている文字列のことである．

例えば，’https://qiita.com/yamatoken/private/hogehogehoge’ の’hogehogehoge’ の部分のことである．

ID の指定がなければ，option で指定した方の記事のタイトルを順次表示し，取得するかしないかを選択できる．’y’ が取得，’n’ は次へ,’e’ は終了を表している．

取得した記事は’id.org’ としてディレクトリ内に作成される．

list

qiita 上の記事のタイトルや URL を表示するためのコマンドである．

コマンドは’qiita list [qiita/teams]’.

option を’qiita’ と指定すれば，Qiita 上の自分の記事を最新から 100 件表示し，teams とすれば，Qiita:Team 上の全ての記事を最新から 100 件表示する．

all

ディレクトリ内の全ての Org ファイルを Qiita へ投稿するためのコマンドである．

’qiita all’ と実行すると全ての Org ファイルをファイル内の ID の記載通りに投稿する．ID 記載の無いファイルは限定共有記事として新規投稿される．

option の指定も可能で，post,upload 同様 public,private,teams を選択でき，選択した場合指定の場所にしか投稿しない．

また，’–exclude’ と引数を使いファイルを指定するとそのファイルを除外してそれ以外を全て投稿する．例えば，数ある Org ファイルの中から’hoge.org’ だけは投稿したく無い場合は，

```
> qiita all --exclude hoge.org
```

と実行することで除外できる．アスタリスクなどを用いて複数のファイルを除外することも可能．

config

'qiita_org' を使用するための設定ファイルを設定するためのコマンド．設定方法は上記の' 設定方法' の通りである．

```
> qiita config [global/local]
```

と実行すると，現在設定されている内容を確認できる．