GitとGitHubによるプログラム管理の基礎

山ノ内　勇斗（名古屋大学, 理学研究科）

目次

参考文献

いちばんやさしい Git&GitHubの教本

https://amzn.to/3tvXF7E

Web制作者のための GitHubの教科書

https://amzn.to/4ajaiUt

GitとGitHubについて

・・・プログラムのバージョン管理を簡単にするツール

・ファイルがバージョンの分だけ増える

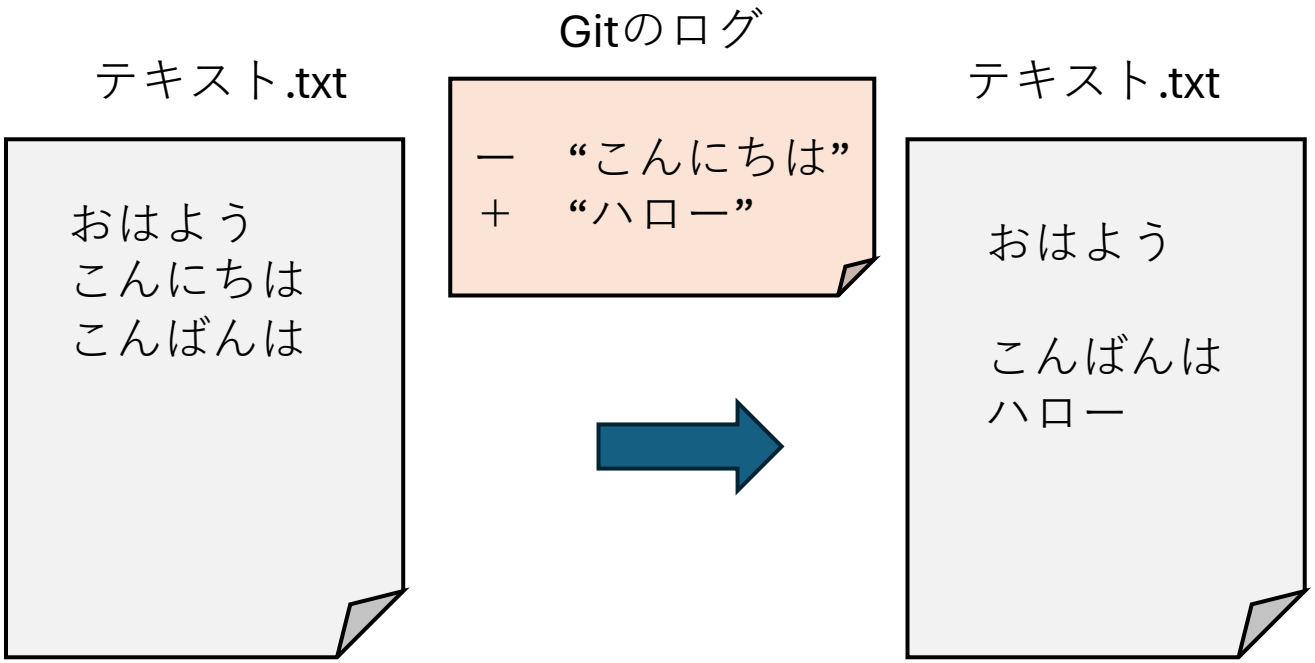
・変更履歴がわかりにくい

コラム：OneDriveでは変更履歴を保存しなくても、自動保存によって自動的にバージョン管理がなされている。

Gitのバージョン管理システムの概念

“各ファイルの差分のみを履歴に保存する”

前に戻りたい時には
追加した部分を消去すればいい！



GitとGitHubについて

git

https://git-scm.com

バージョン管理システム
ローカルでファイルの
バージョン管理を行う
(**ローカルレポジトリ**)

GitHub

https://github.com

オンラインでファイルの
バージョン管理を行う
(**リモートレポジトリ**)

GitHub Desktopについて

・・・本来CUIベース(ターミナルとか)で行うものを簡単に行えるようにしたツール！！

GitHub Desktop

https://desktop.github.com

GitやGitHubの操作をGUI上で行うことができる

GitHub のアカウントの種類

- **個人用アカウント**

  ・・・通常これ！！ひとりごとにひとつのアカウント。

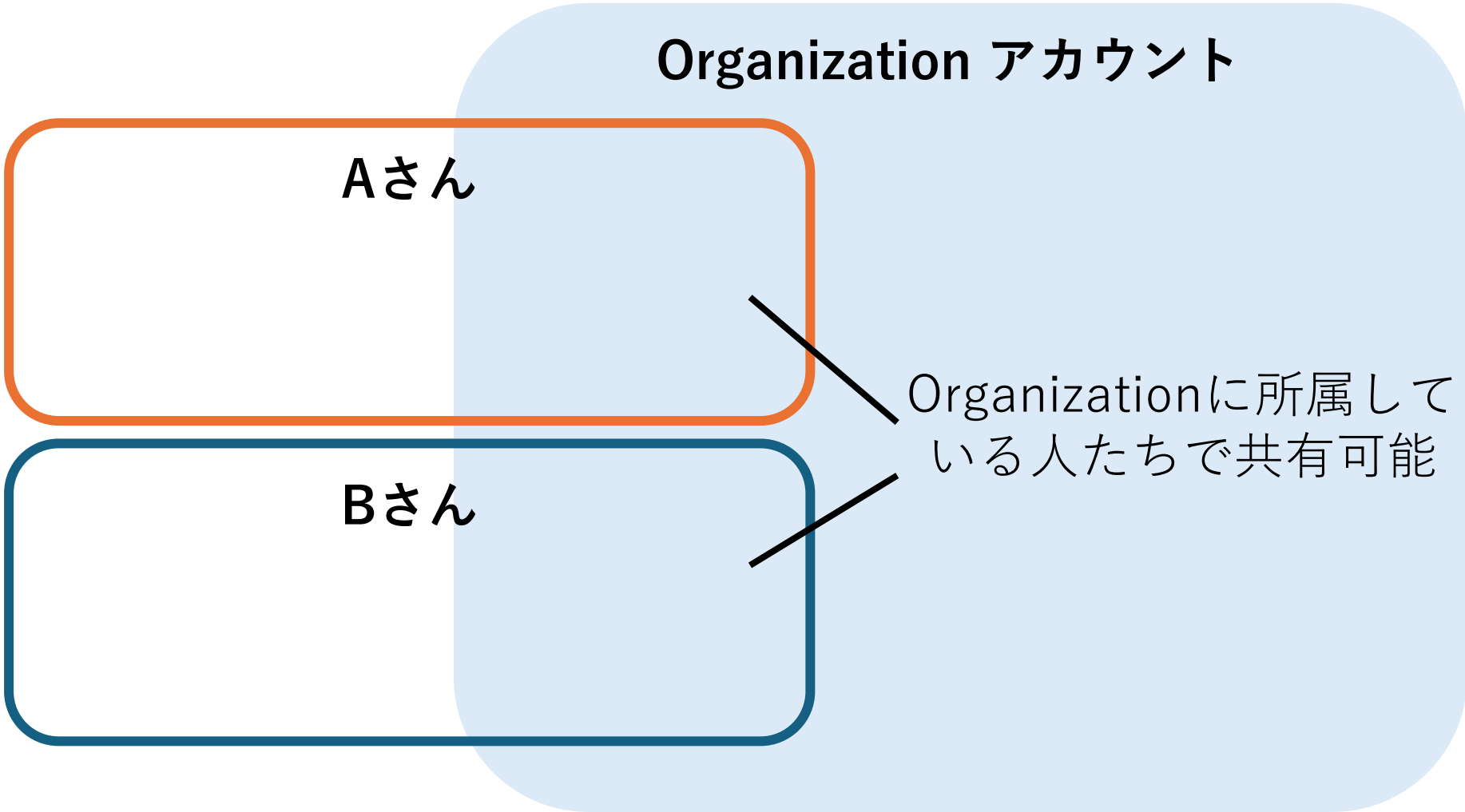
- **Organization アカウント**

  ・・・無制限の数の人が多くのプロジェクトで同時に
  コラボレーションできる共有アカウント。
  組織ごとにひとつのアカウント。

- **Enterprise アカウント**

  ・・・企業用アカウント。
  よくわからないが、基本使わない。

アカウントの所属関係について



共同制作について

1. レポジトリのcollaboratorに追加する

・Privateで公開しているレポジトリに、アクセス権限を割り振ることができる

・アクセス権限には、観覧のみ、編集可能、管理者など種類がある

2. Organizationに入っている人たちで共有する

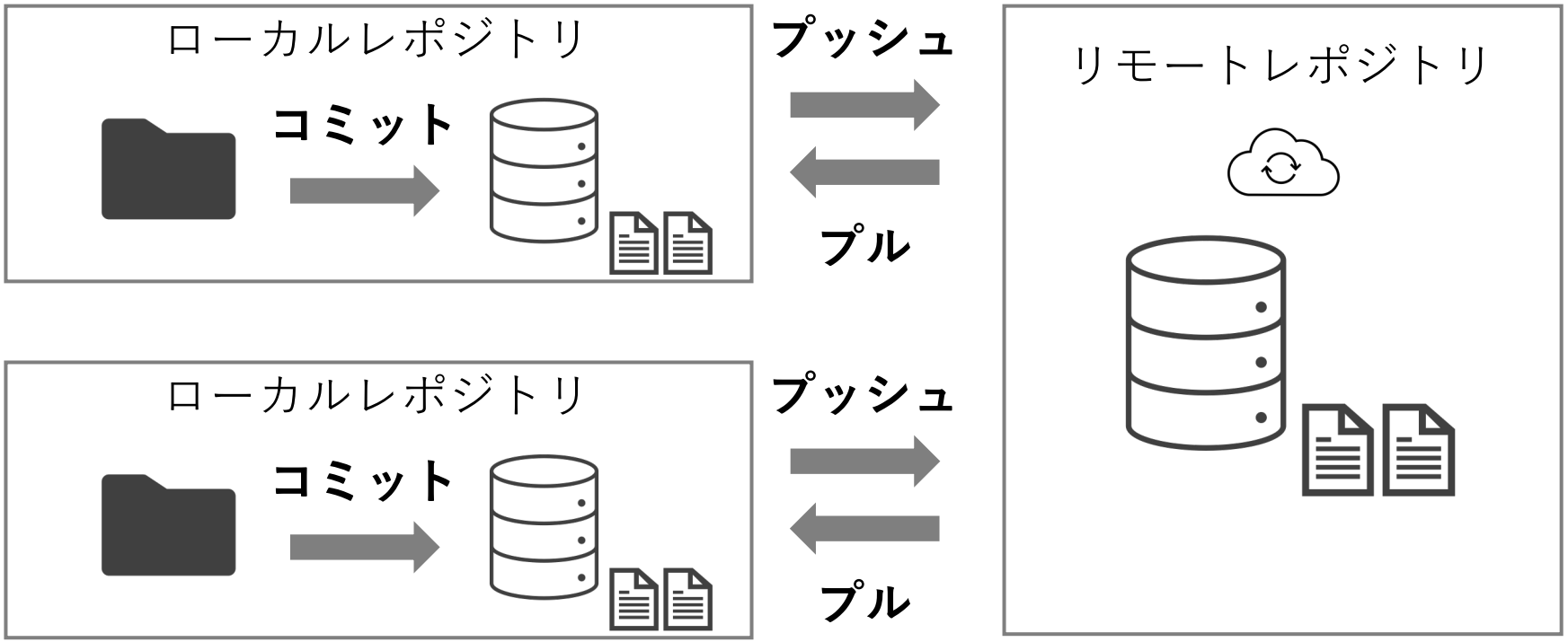
・Organizationに所属している人たちにレポジトリの管理者権限がある

・特に特別な操作は必要ない

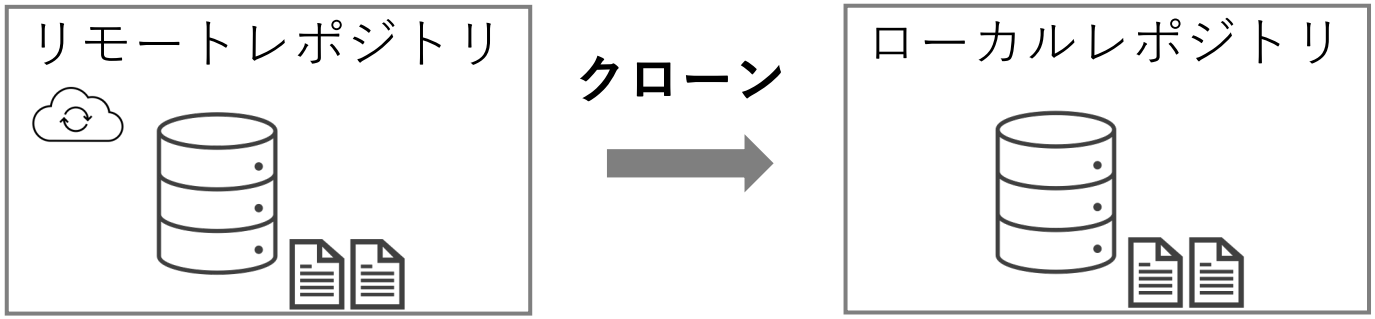
用語説明

- **レポジトリ**・・・プログラム等の”保管場所”
- **ローカルレポジトリ**・・・自分のパソコンにおけるレポジトリ
- **リモートレポジトリ**・・・ネット上にあるレポジトリ
- **ディレクトリ**・・・フォルダのこと
  狭義には、自身が作業しているフォルダ
- **コミット**・・・ディレクトリ→ローカルレポジトリ
- **プッシュ**・・・ローカルレポジトリ→リモートレポジトリ
- **プル**・・・リモートレポジトリ→ローカルレポジトリ

概要図



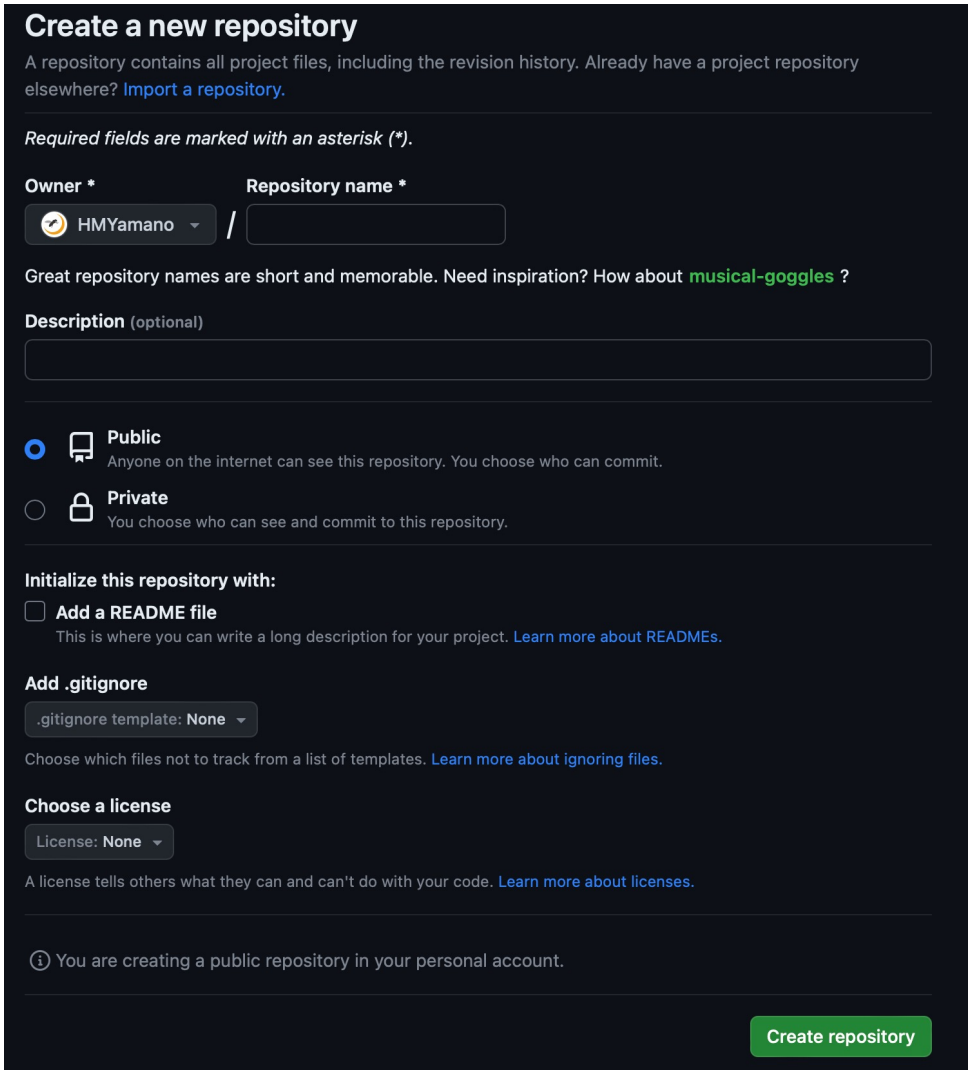
*初回のみ



実際の作業手順 - 新しいレポジトリを作る

@GitHub

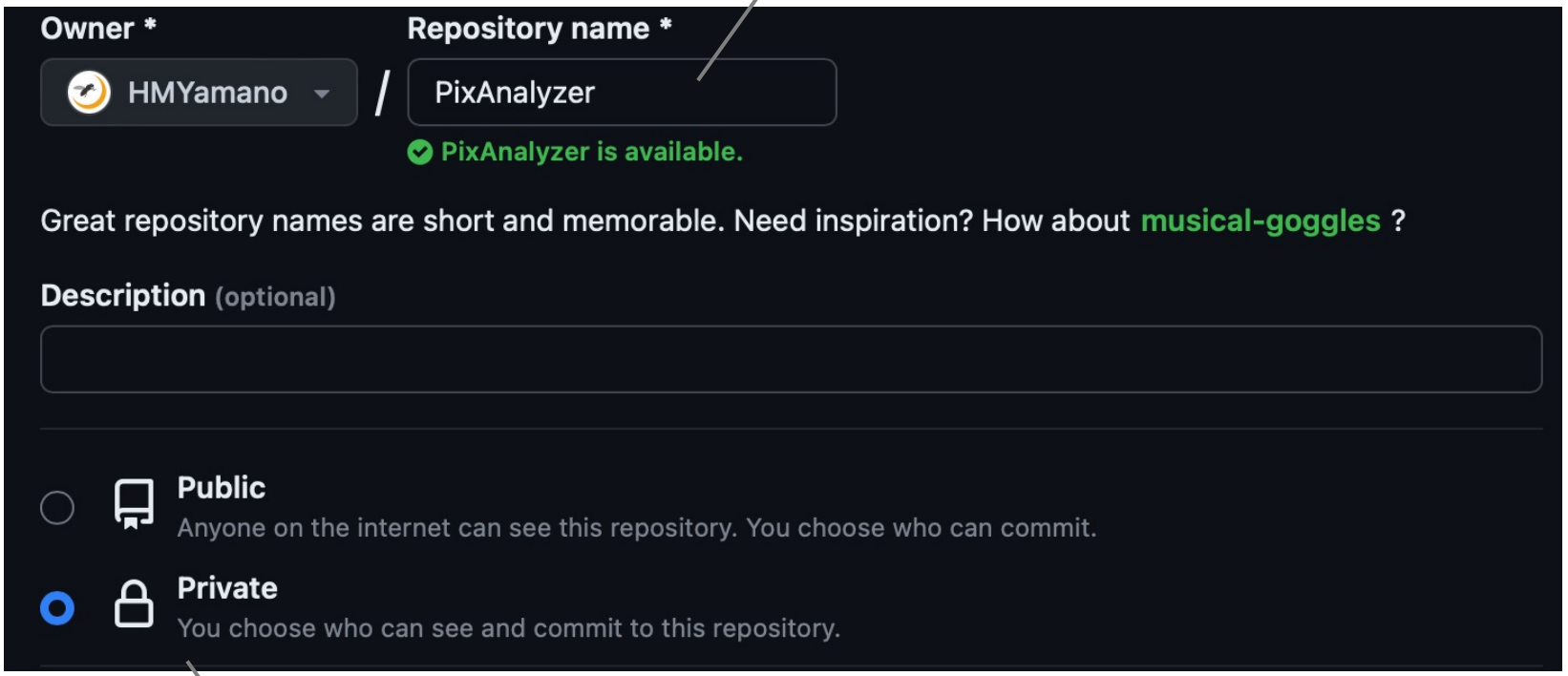
・・・GitHubでCreate a new repositoryを押す



実際の作業手順 - 新しいレポジトリを作る

@GitHub

レポジトリの名前 = プロジェクトの名前



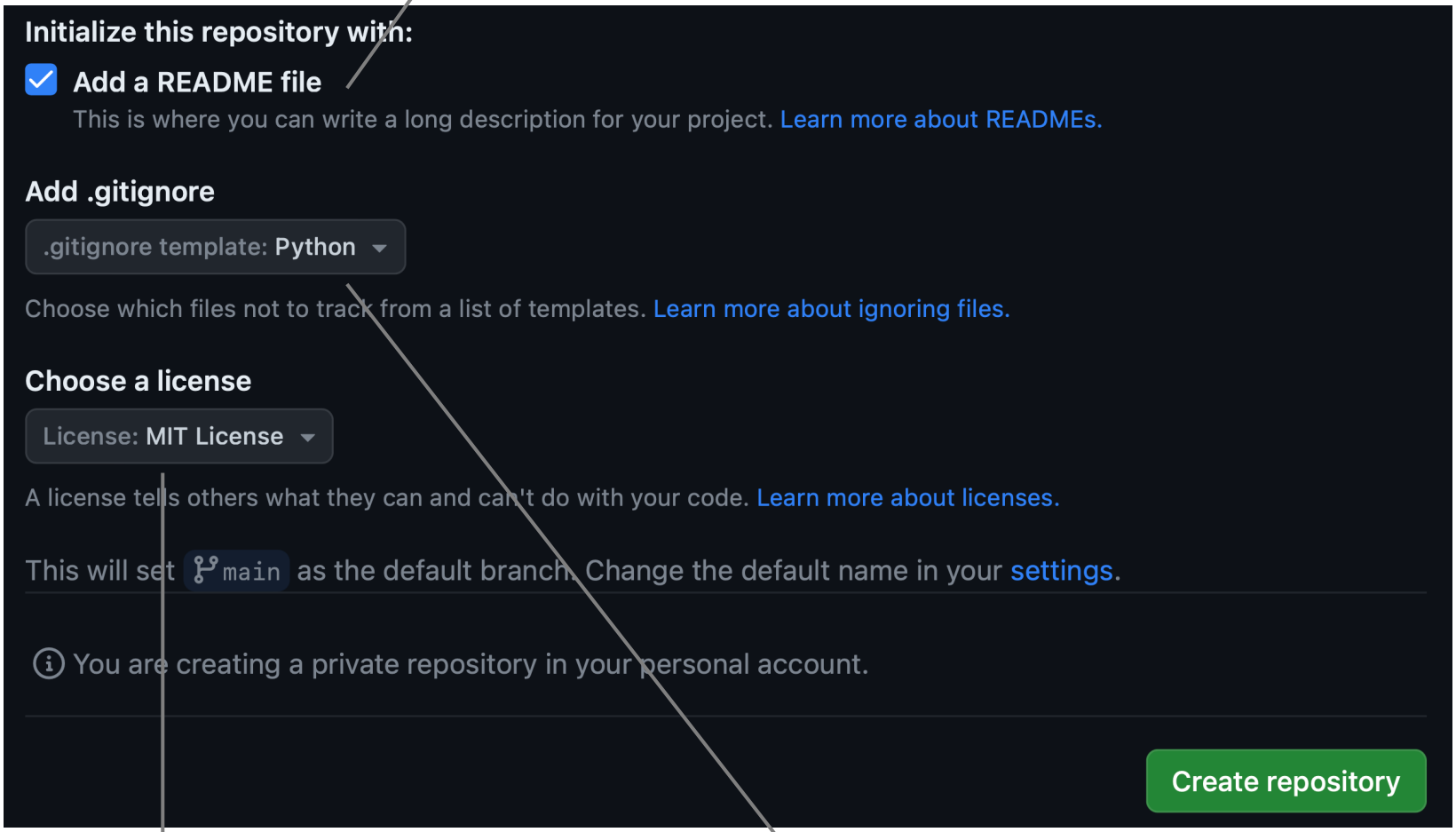
レポジトリの公開範囲

- Public：世の中の人全員
- Private ：自分と共有を許可した人たちのみ

実際の作業手順 - 新しいレポジトリを作る

@GitHub

READMEファイルを追加するか否か

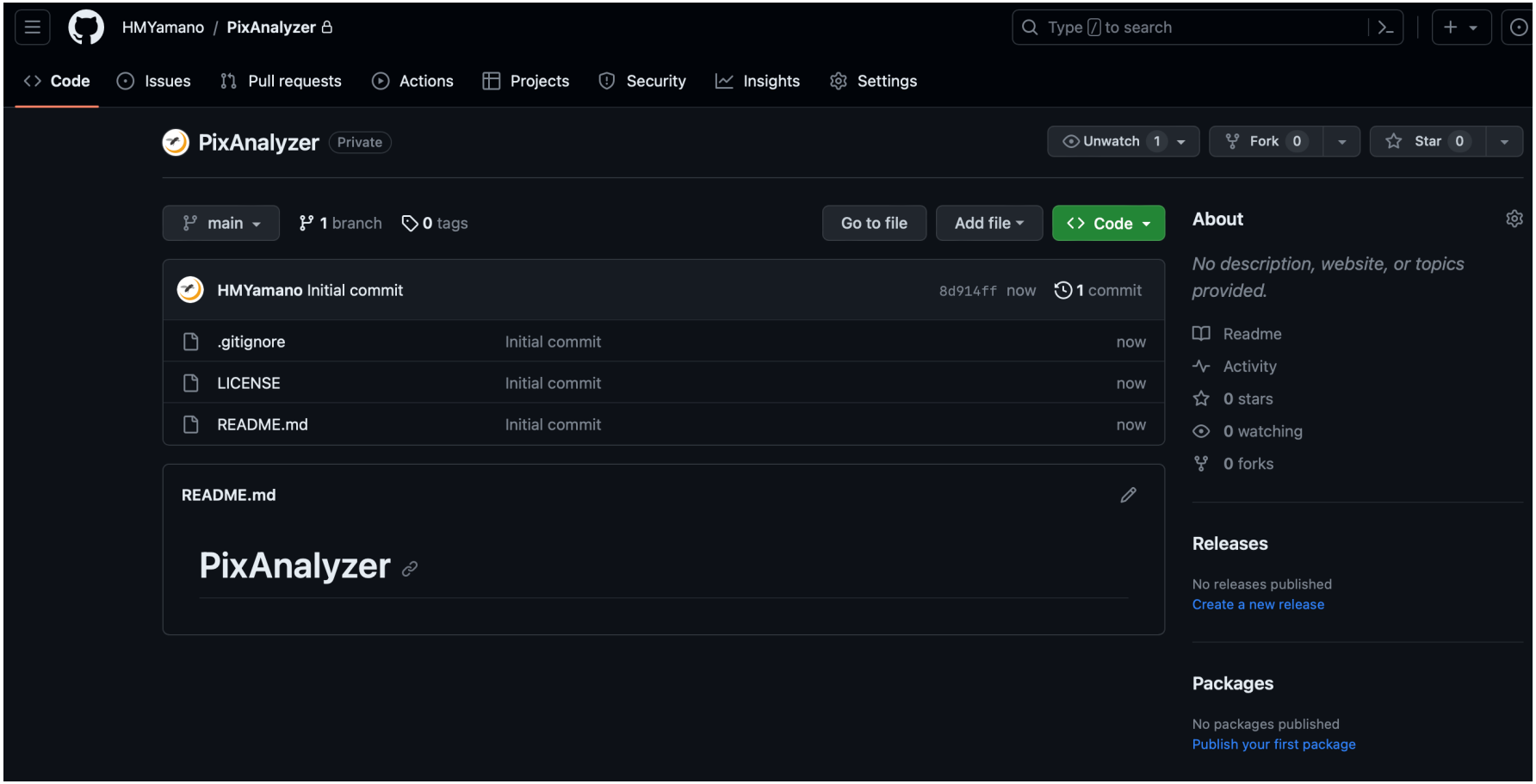


ライセンスについて

.gitignoreファイルについて

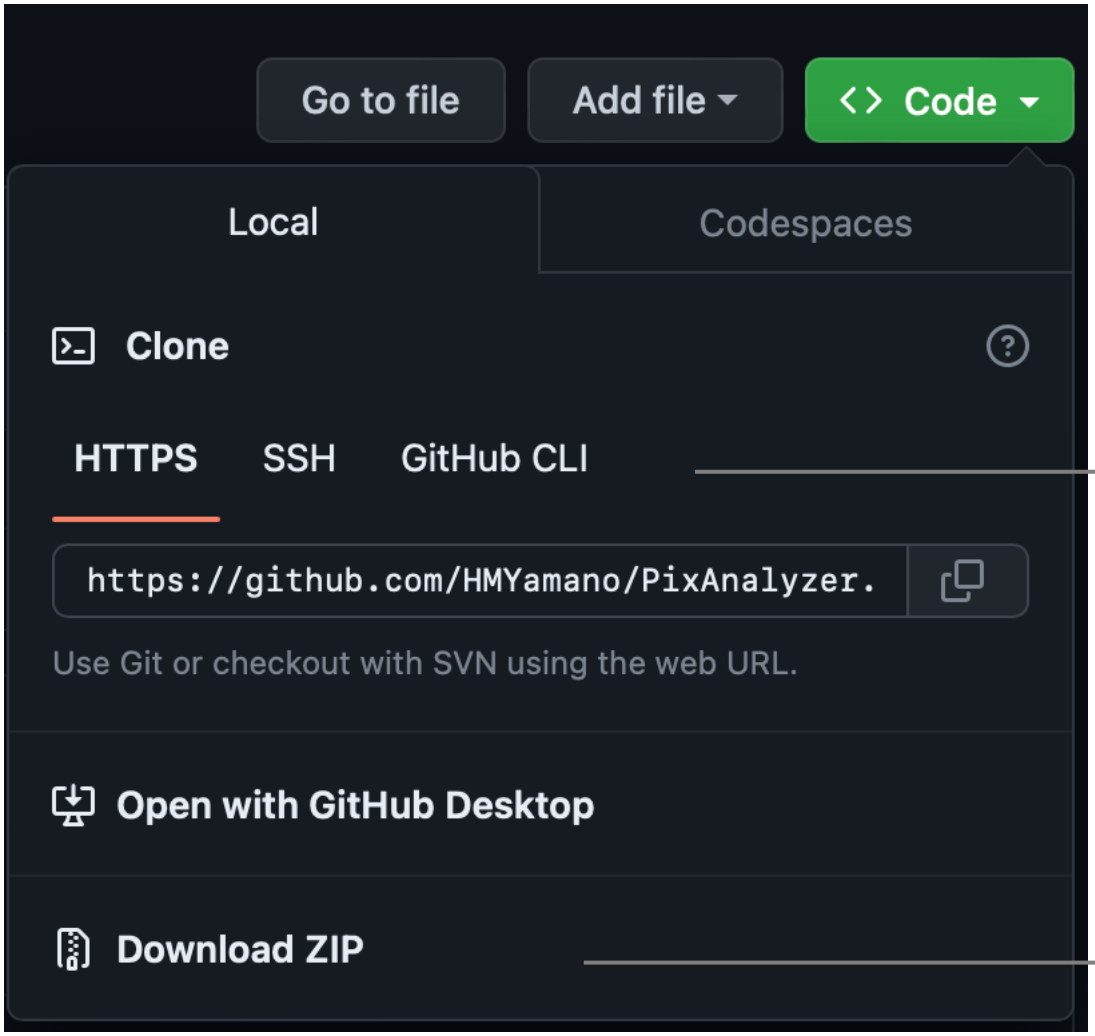
実際の作業手順 - 新しいレポジトリを作る

@GitHub



実際の作業手順 – クローンする

@GitHub



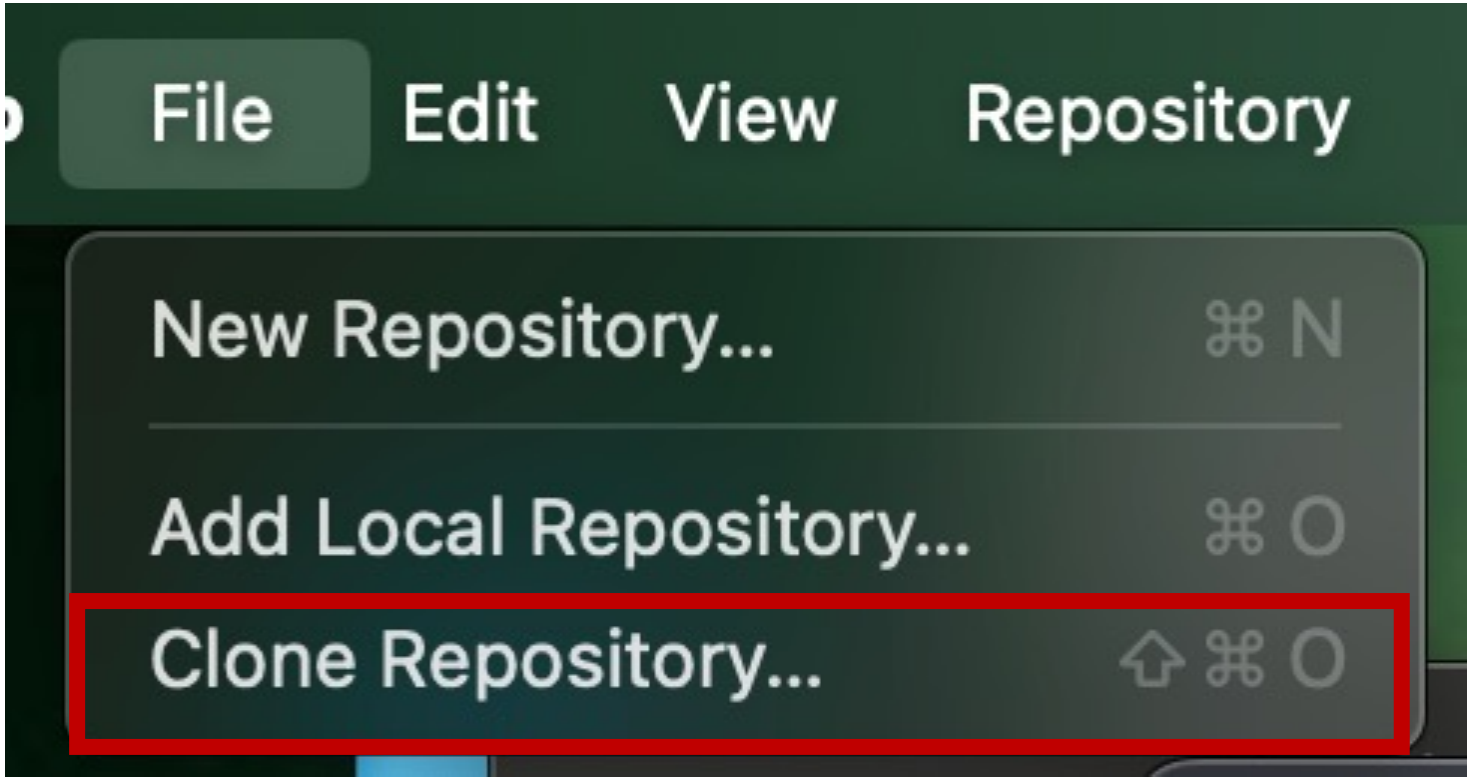
ターミナルでクローンするときのコード

ZIPファイルでダウンロード

実際の作業手順 – クローンする

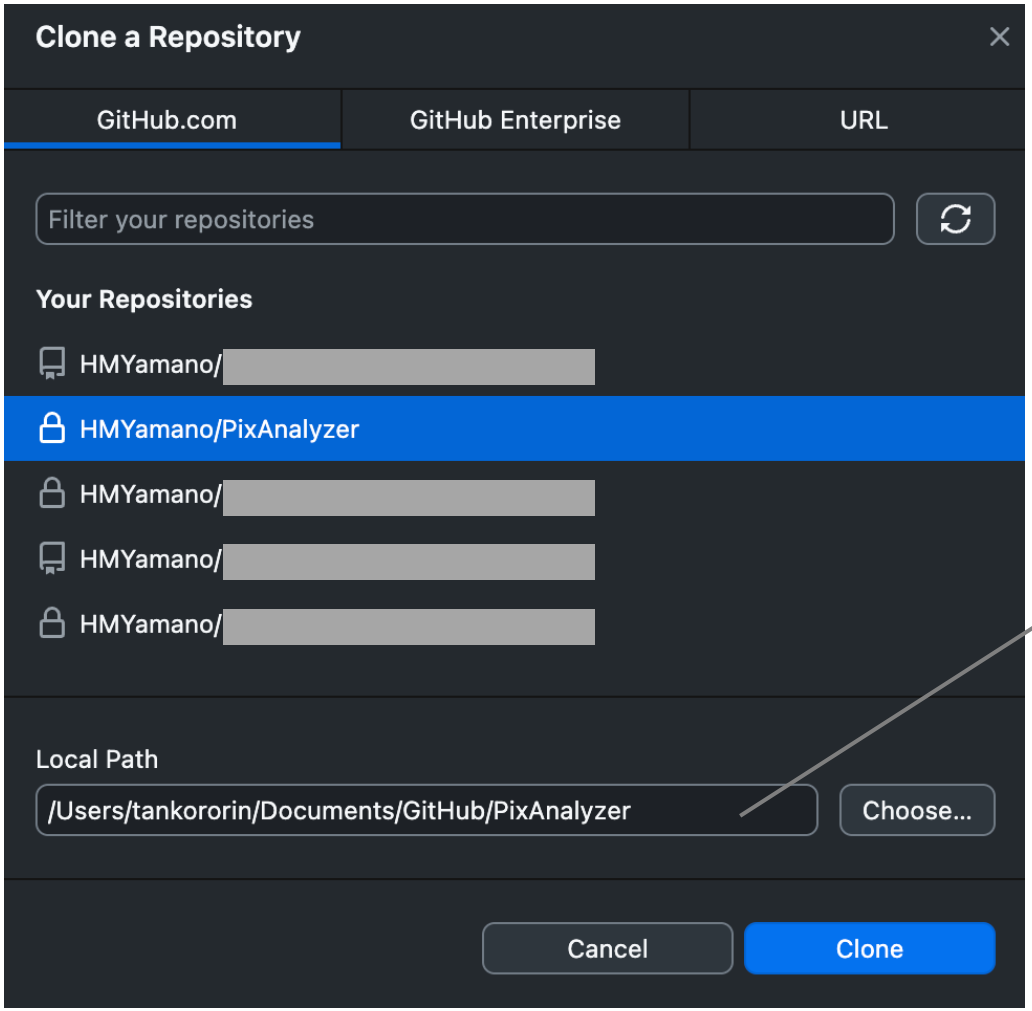
@GitHub Desktop

・・・GitHub Desktopを使ってクローンもできる！



実際の作業手順 – クローンする

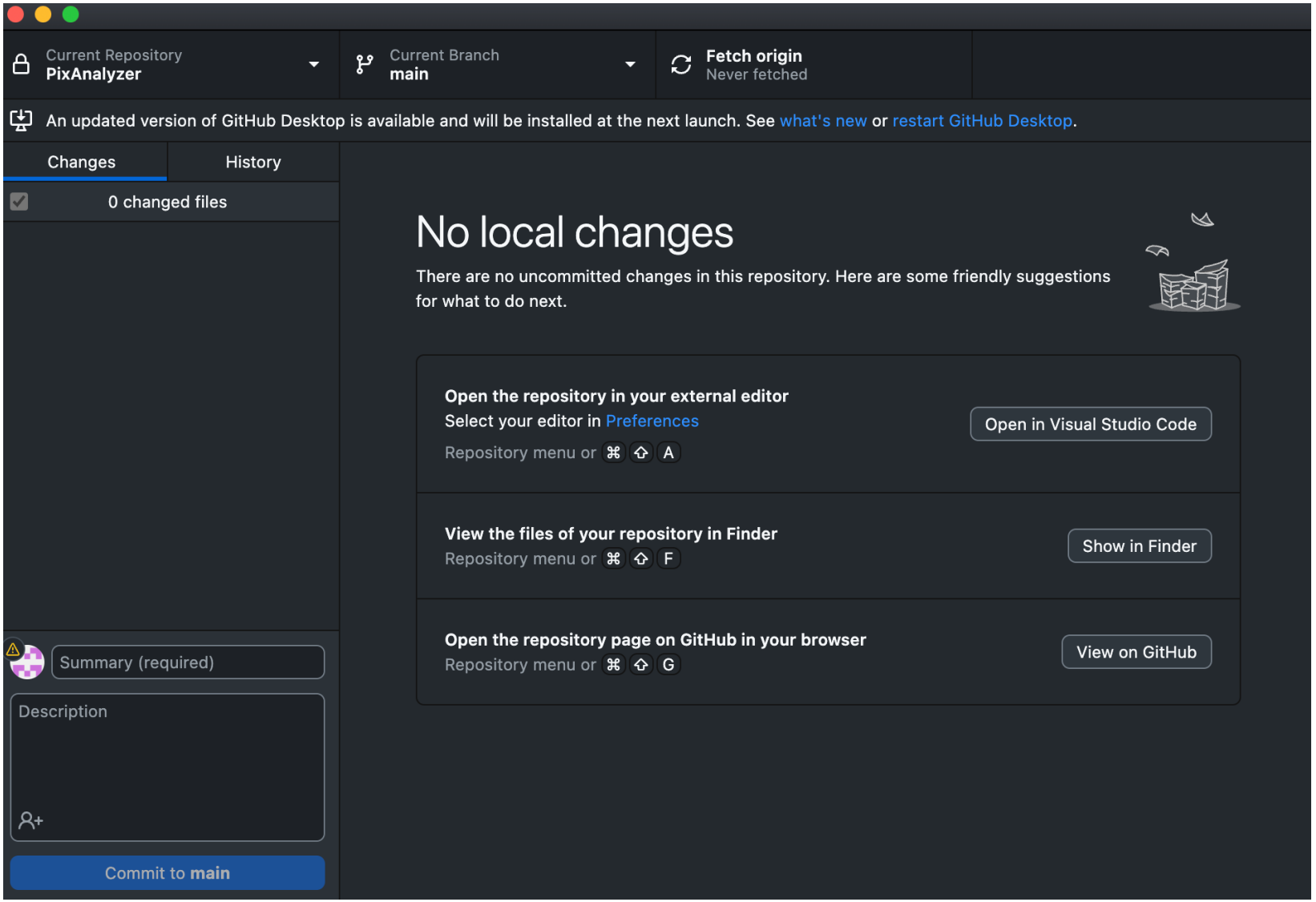
@GitHub Desktop



このパスに
クローンされる

実際の作業手順 – クローンする

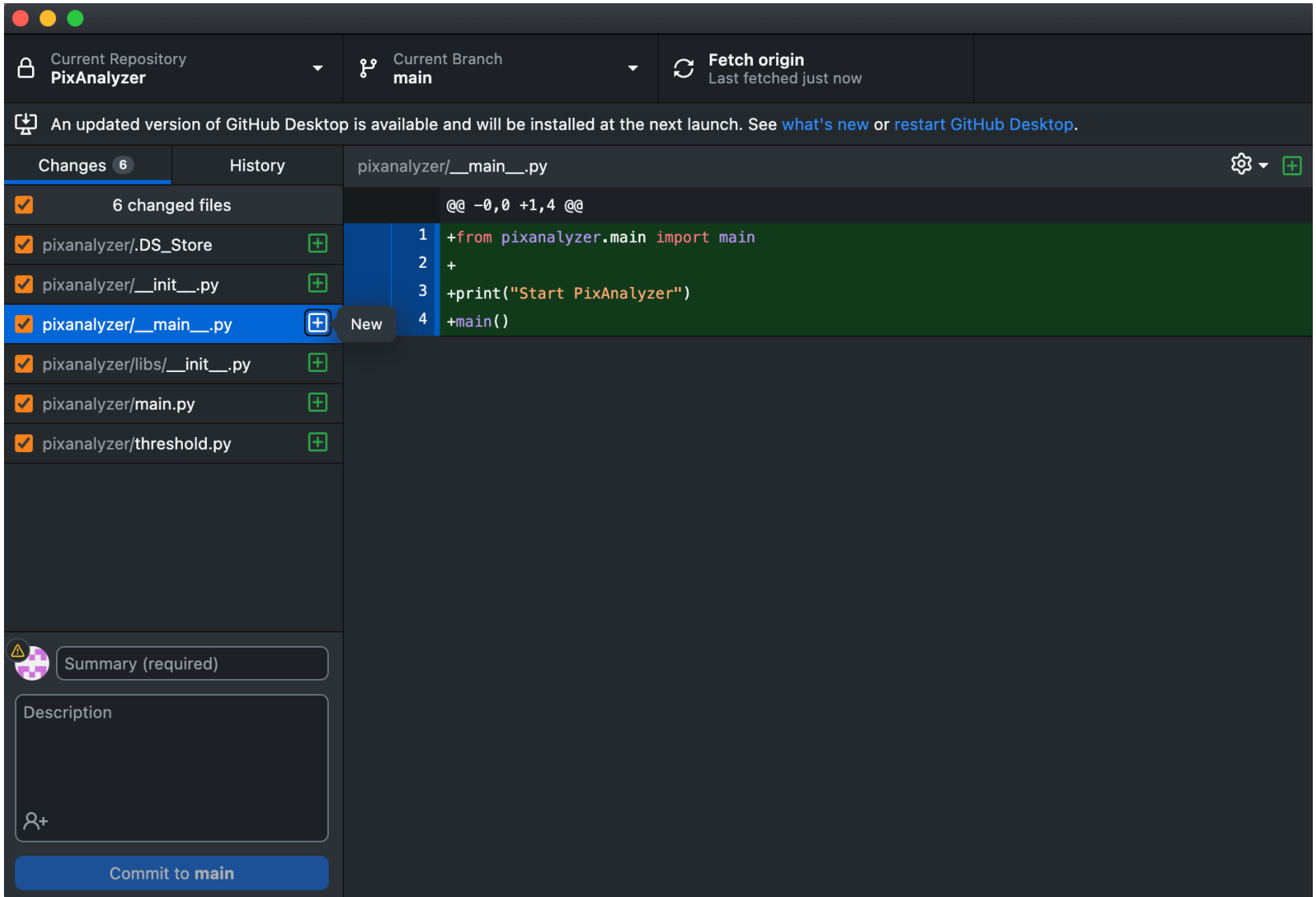
@GitHub Desktop



実際の作業手順 – コミット

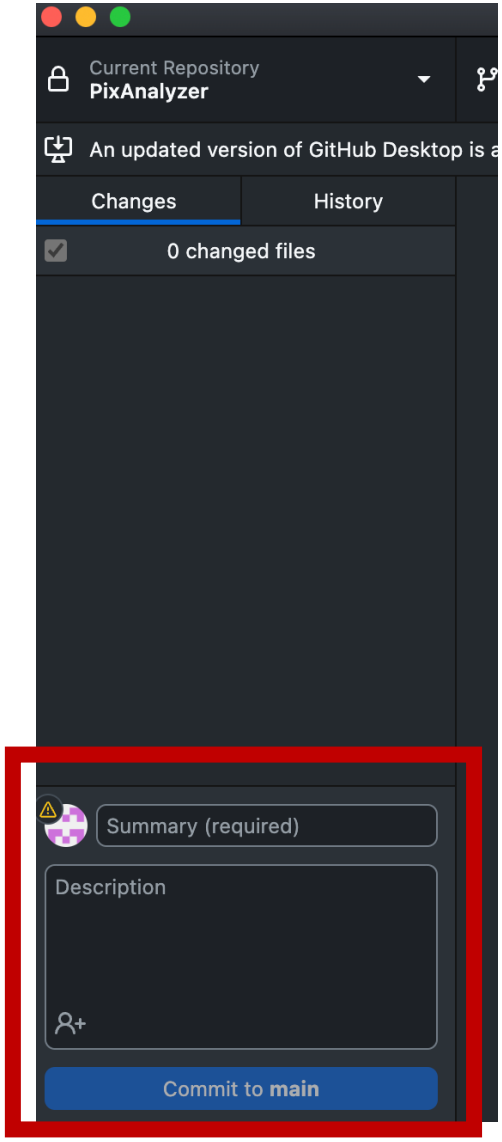
@GitHub Desktop

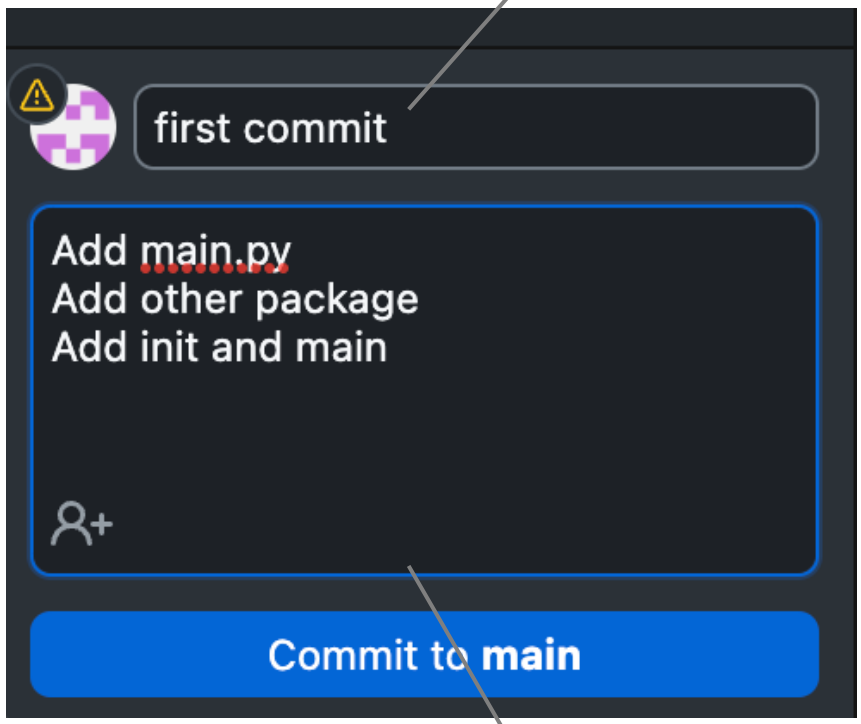
・・・プログラムに変更を加えると。。。



実際の作業手順 – コミット

@GitHub Desktop





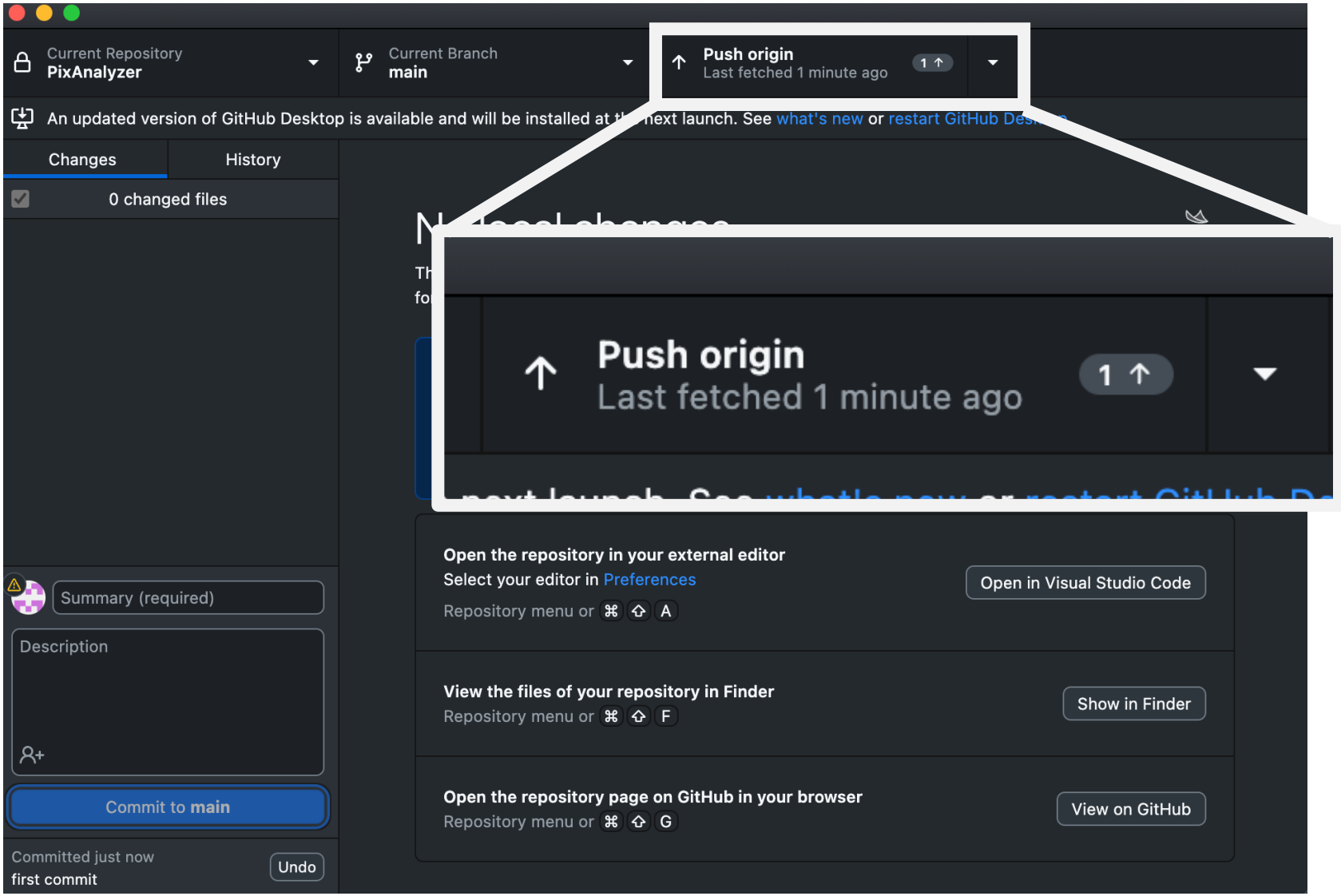
簡単なコミットの説明

詳しいコミットの説明

*慣例的に最初のコミットは
“first commit”とすることが多い

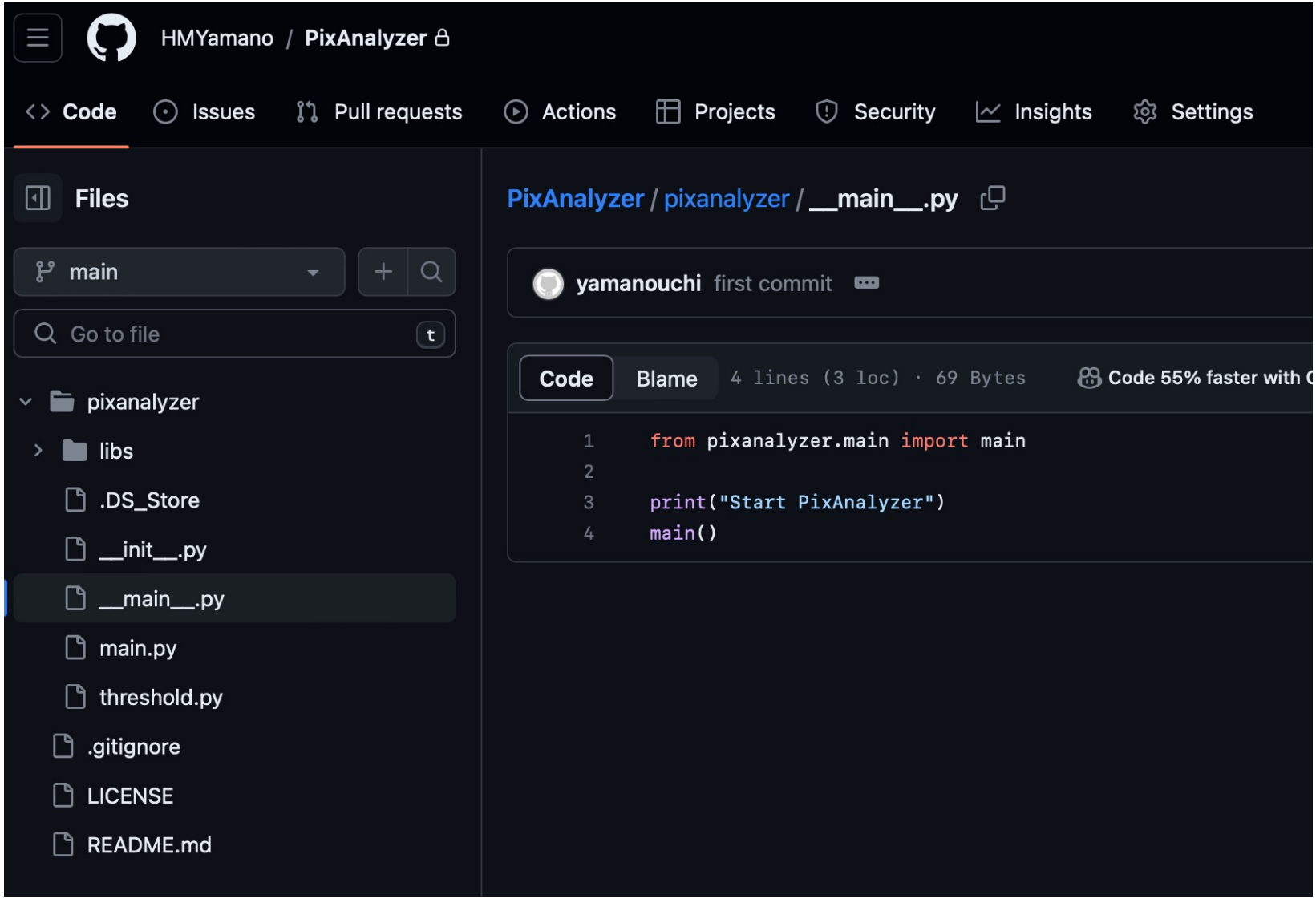
実際の作業手順 – プッシュ

@GitHub Desktop



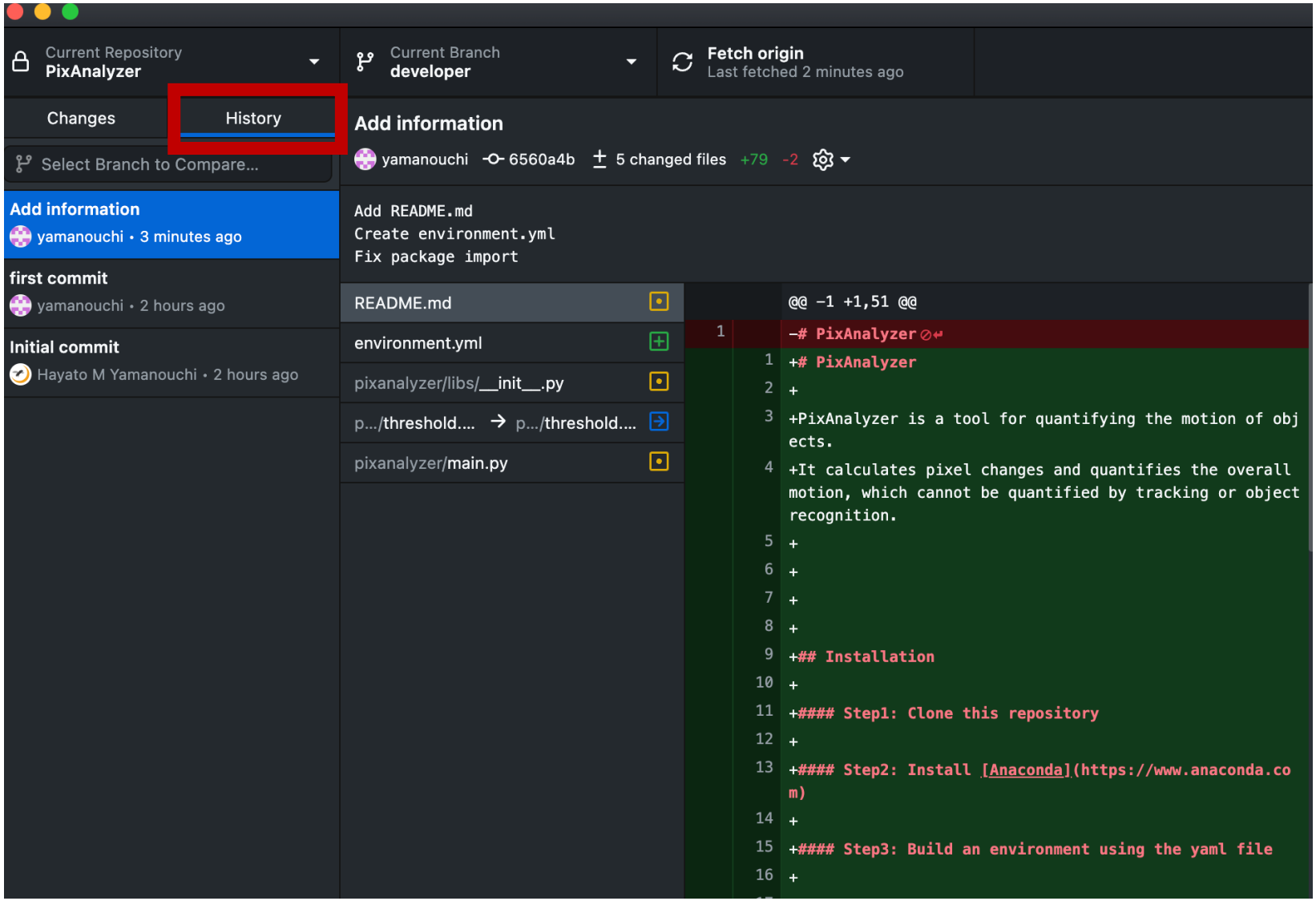
実際の作業手順 – プッシュ

@GitHub



実際の作業手順 – 履歴を見る

@GitHub Desktop



適切なコミット/プッシュのタイミング

・・・履歴を戻るときに自分がわかりやすい単位で

・**コミット**・・・1タスク単位で行う

Ex. 一つのバグの修正
ある統計解析のコードの追加・修正

・**プッシュ**・・・タスクのまとまり単位

Ex. 一連の統計解析の追加
大きな機能の追加

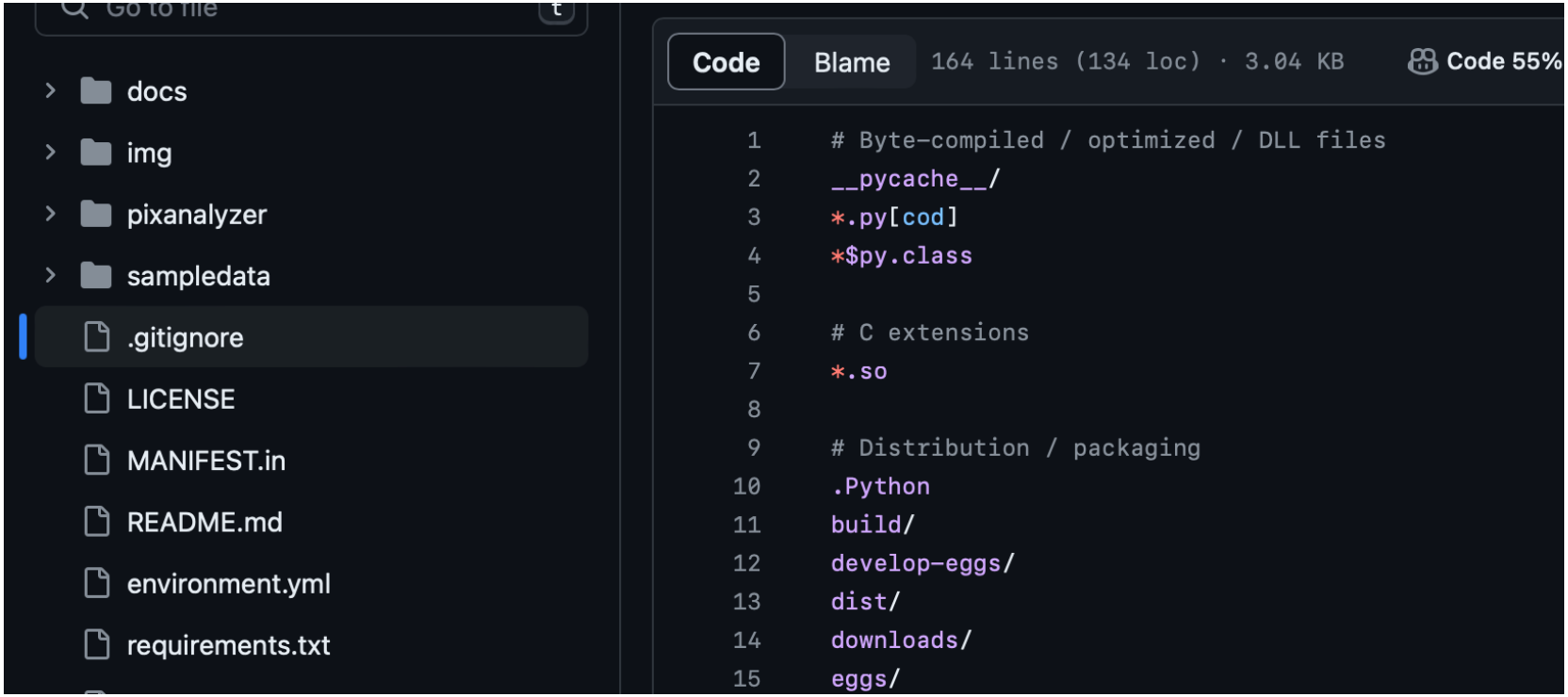
* コミットにはそのコミットが何であるかのコメントと共に行う必要がある

* チームで行う場合には、チームで共有しておくこと

.gitignoreファイルについて

・・・レポジトリで共有しないファイルの指定

レポジトリに預けられるファイルの大きさには限界がある
→動画とかは入れたくない



.gitignoreファイルについて

・・・レポジトリで共有しないファイルの指定

・特定のファイルを入れたくない
　　ファイル名全部を書く　(Ex.　sample.txt)

・特定のファイル形式を入れたくない
　　アスタリスクと拡張子　(Ex.　*.mp4)

・特定のフォルダを入れたくない
　　フォルダ名とスラッシュ　(Ex.　Sample/)

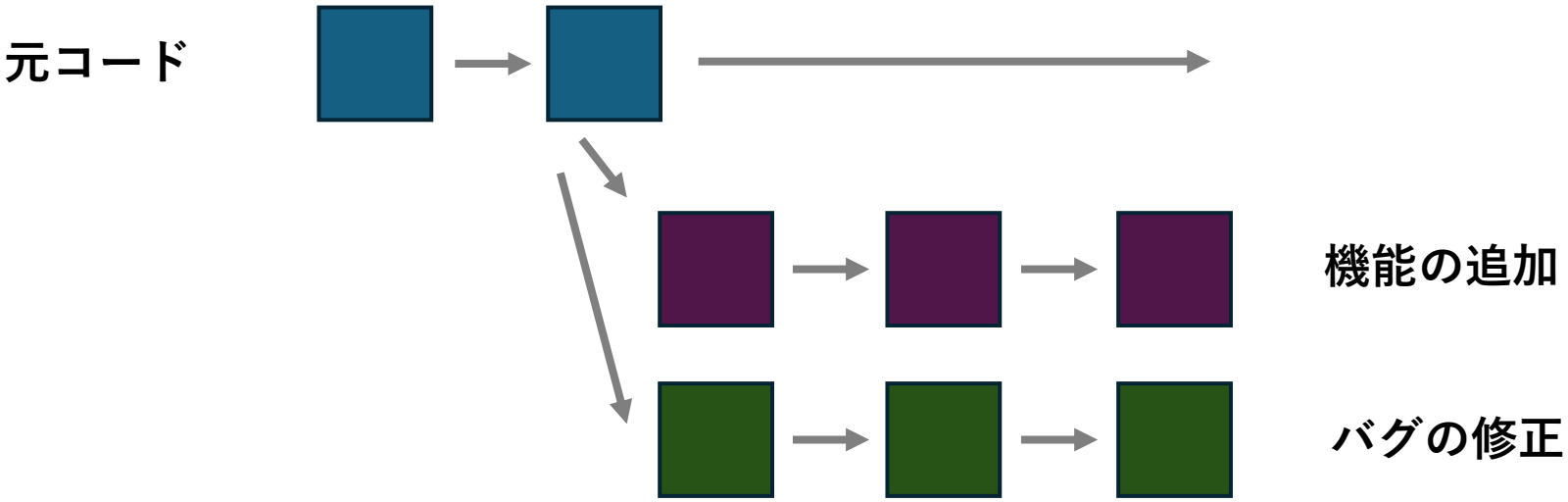
あとはただ羅列するだけ

ブランチ

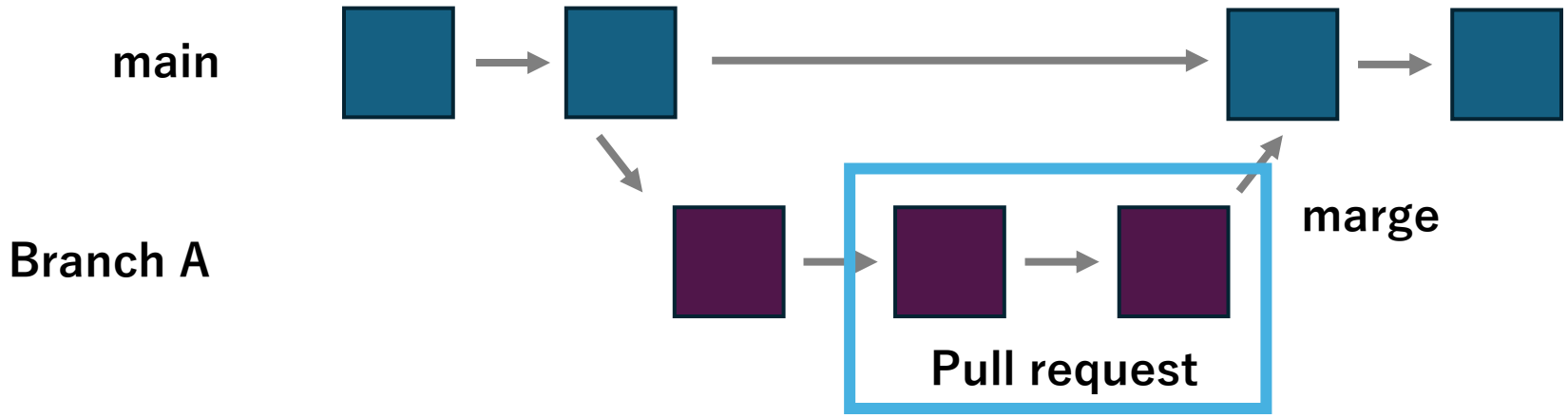
・・・履歴の流れを分岐して記録していく

プログラムの統計の検討もしたいし、
新たなデータの追加もしたいし、
やりたいことがいっぱい・・・・

ブランチを作る(切る)ことで
それぞれに特化して作業を進められる



ブランチをマージする

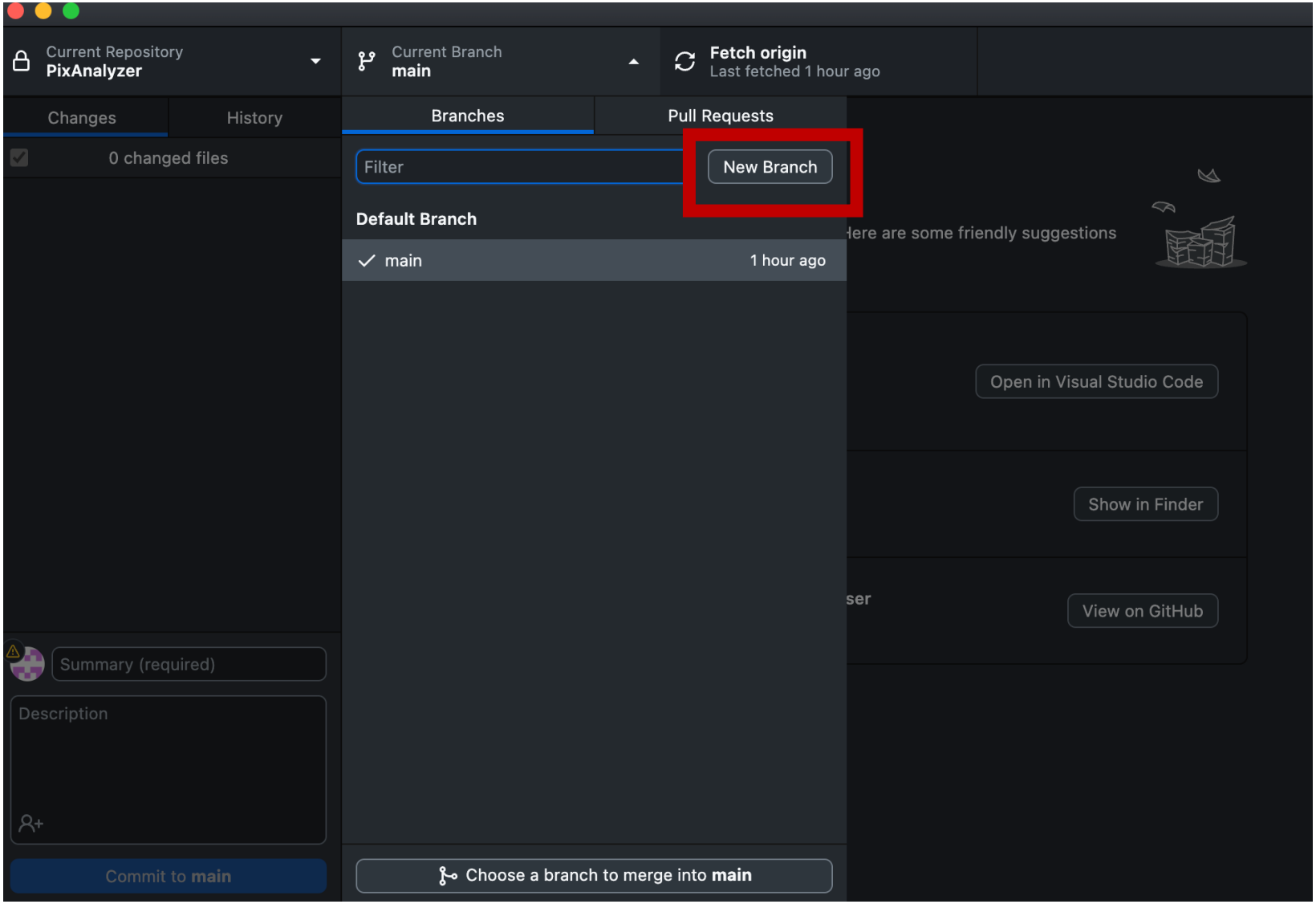


ブランチのマージにはプルリクスト(Pull request)を行なって、承認作業(自分or共同開発者)が必要となる

他の変更と変更内容が異なったりする場合には、プルリクエストでどれを反映させるかを選択、修正を行う

実際の作業手順 – ブランチを切る

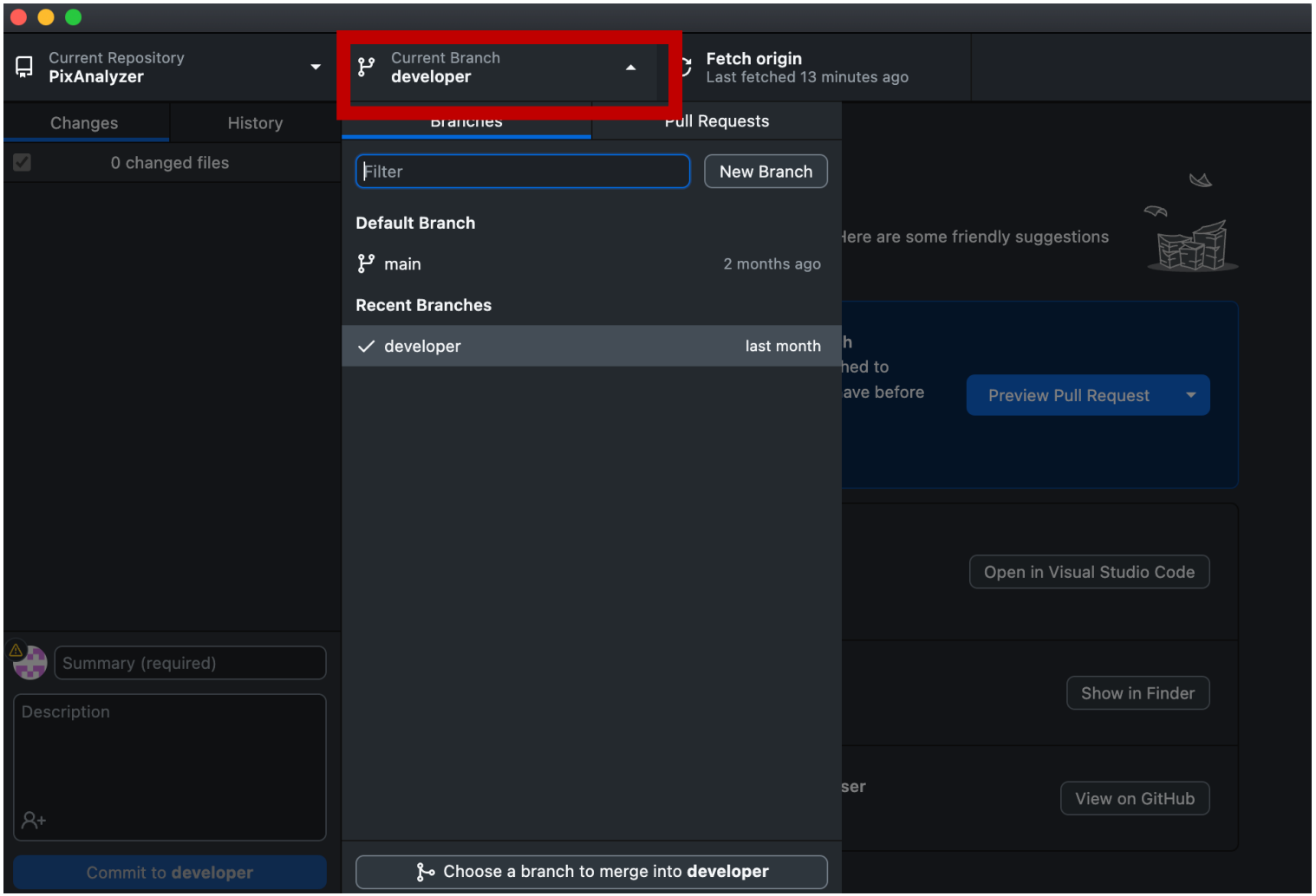
@GitHub Desktop



実際の作業手順 – ブランチを切る

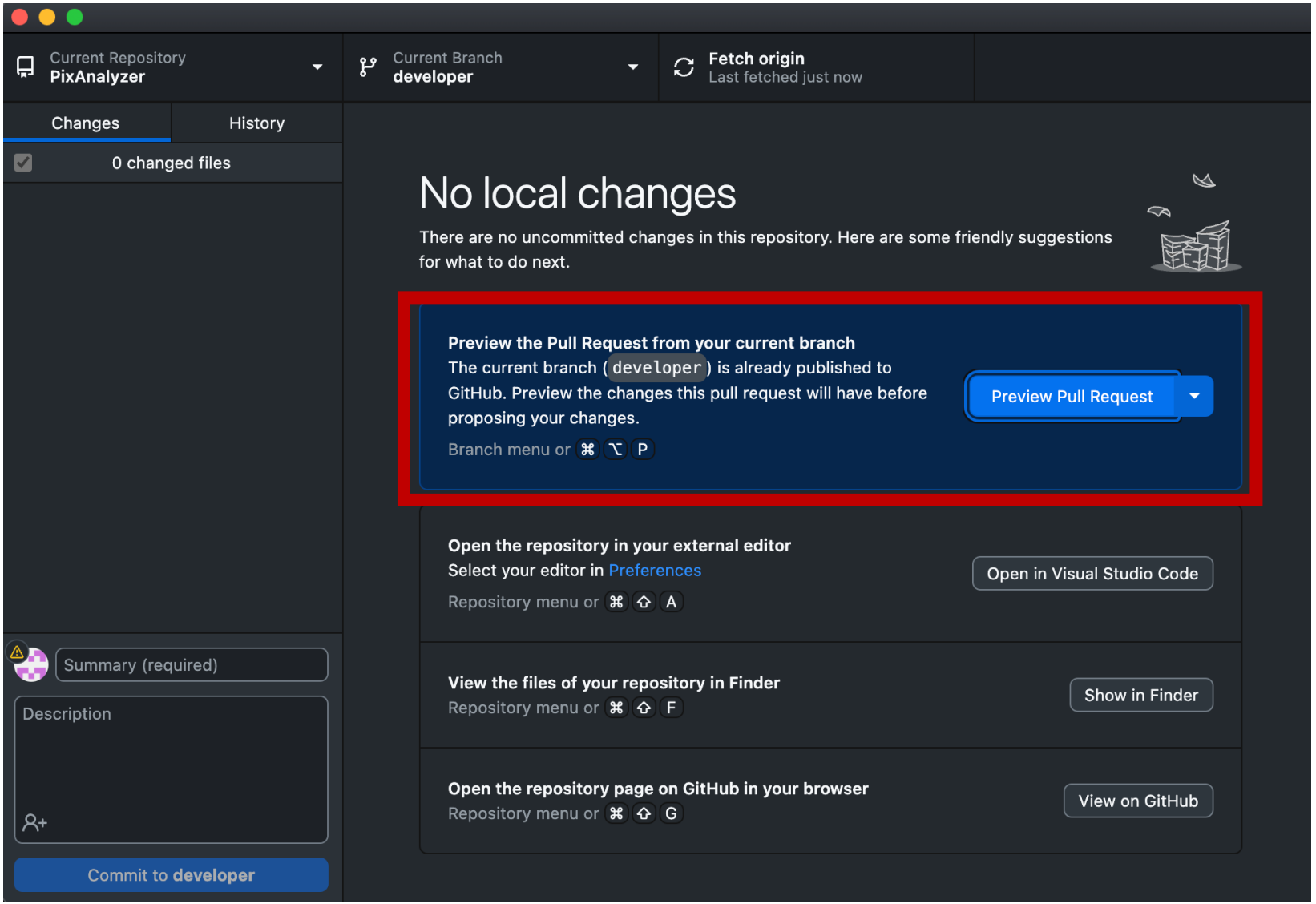
@GitHub Desktop

・・・Current Branchを選択すると、
ローカルレポジトリで勝手にブランチが切り替わる



実際の作業手順 – Pull request

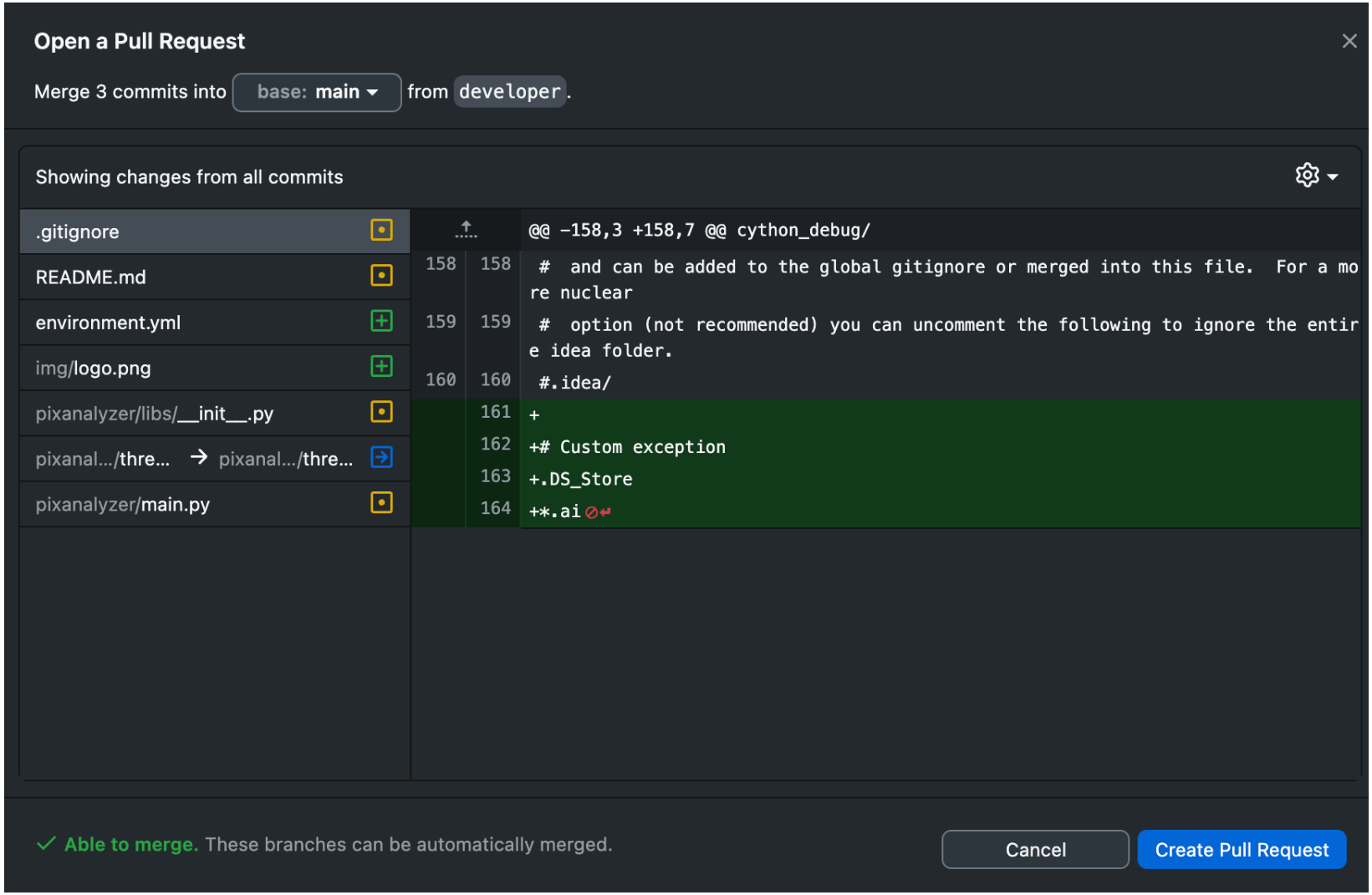
@GitHub Desktop



実際の作業手順 – Pull request

@GitHub Desktop

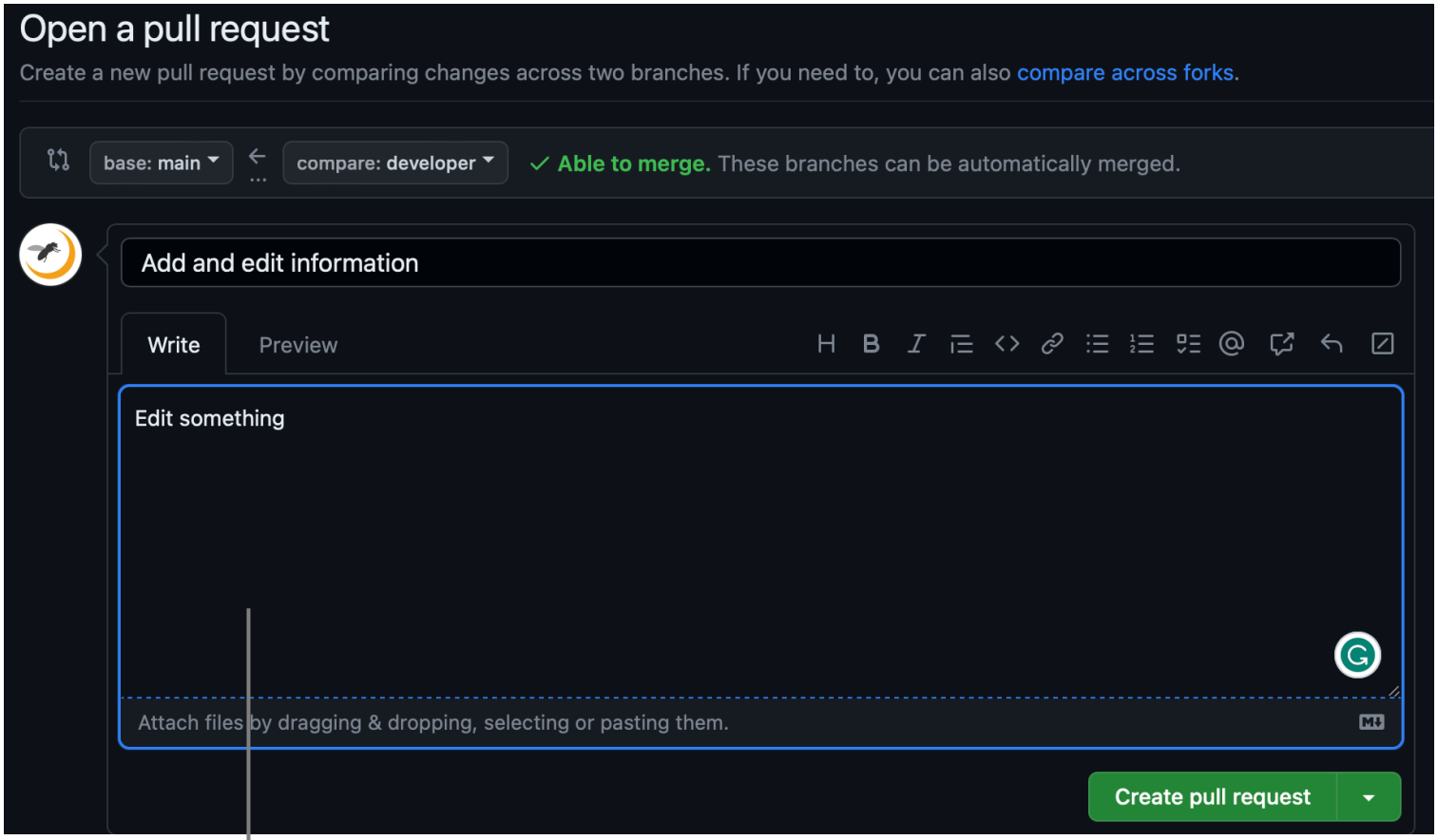
・・・ブランチ間の変更を見る！



やってみよう

@GitHub

・・・Create pull requestを押すとGitHubに飛ばされる

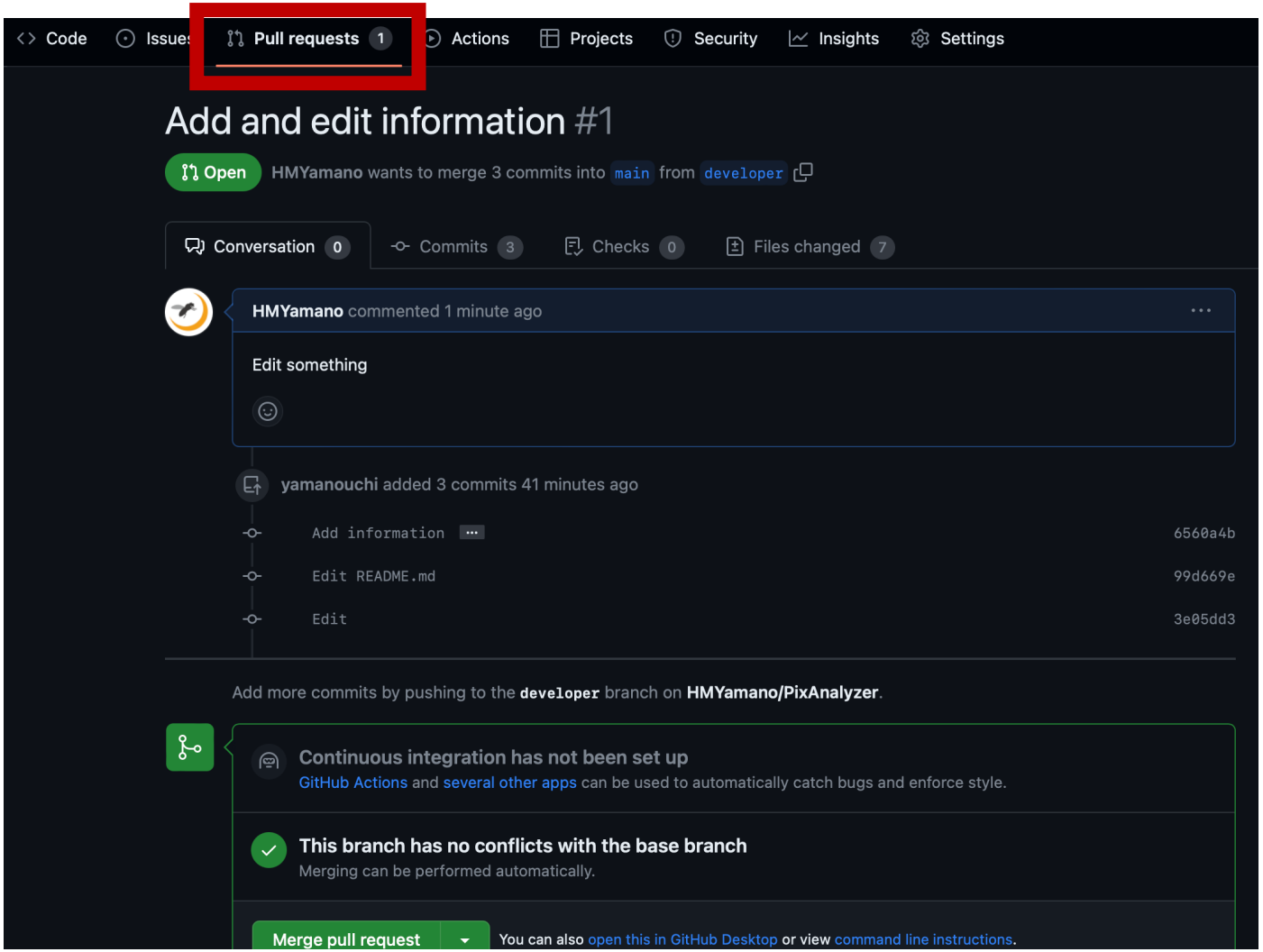


どんな修正を行なったのかなどの、
Pull requestの内容を記述する

やってみよう

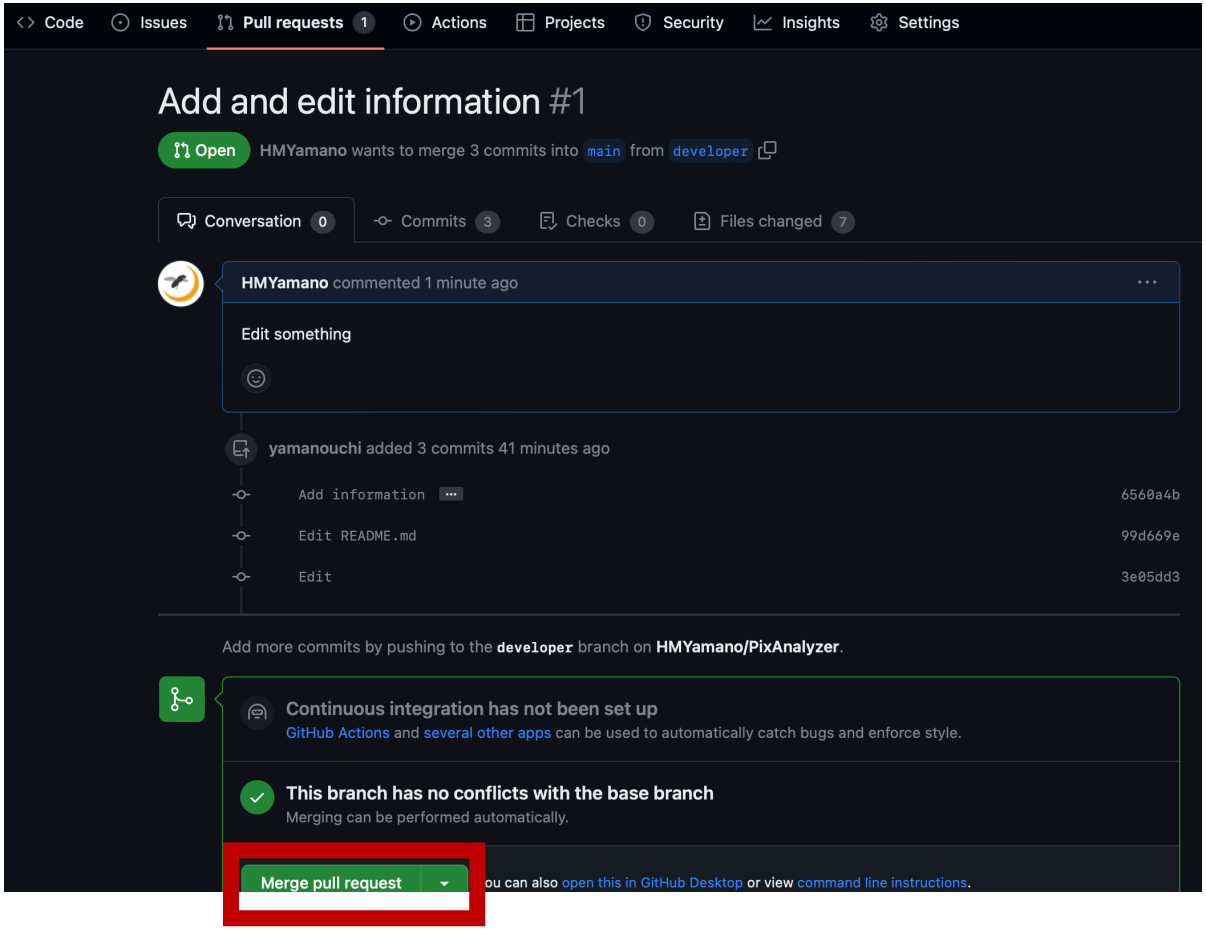
@GitHub

・・・Pull requestに表示される。



やってみよう

@GitHub

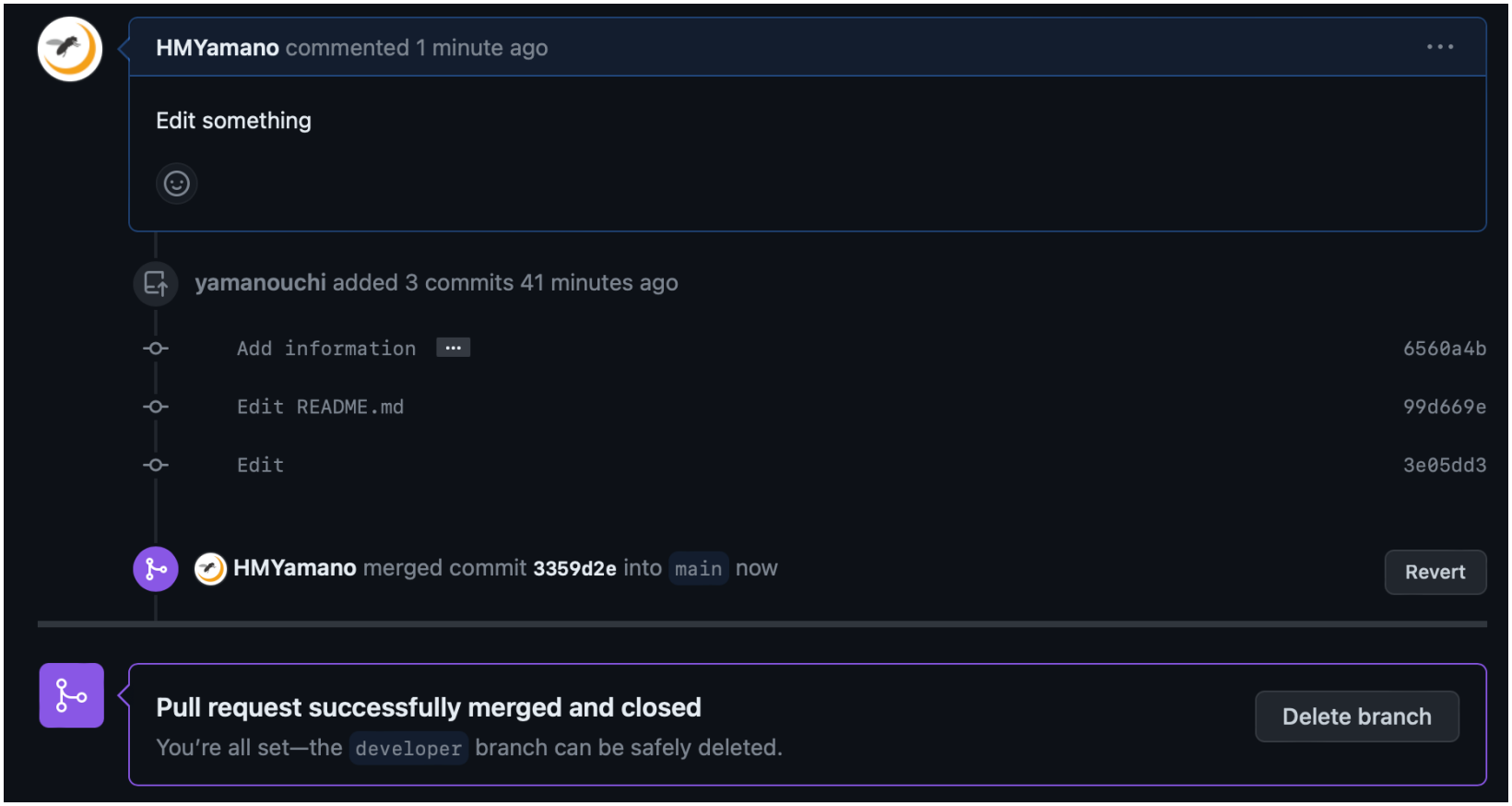


OKであれば、“Marge pull request”を、
Noであれば、修正依頼や議論、拒否を行う

やってみよう

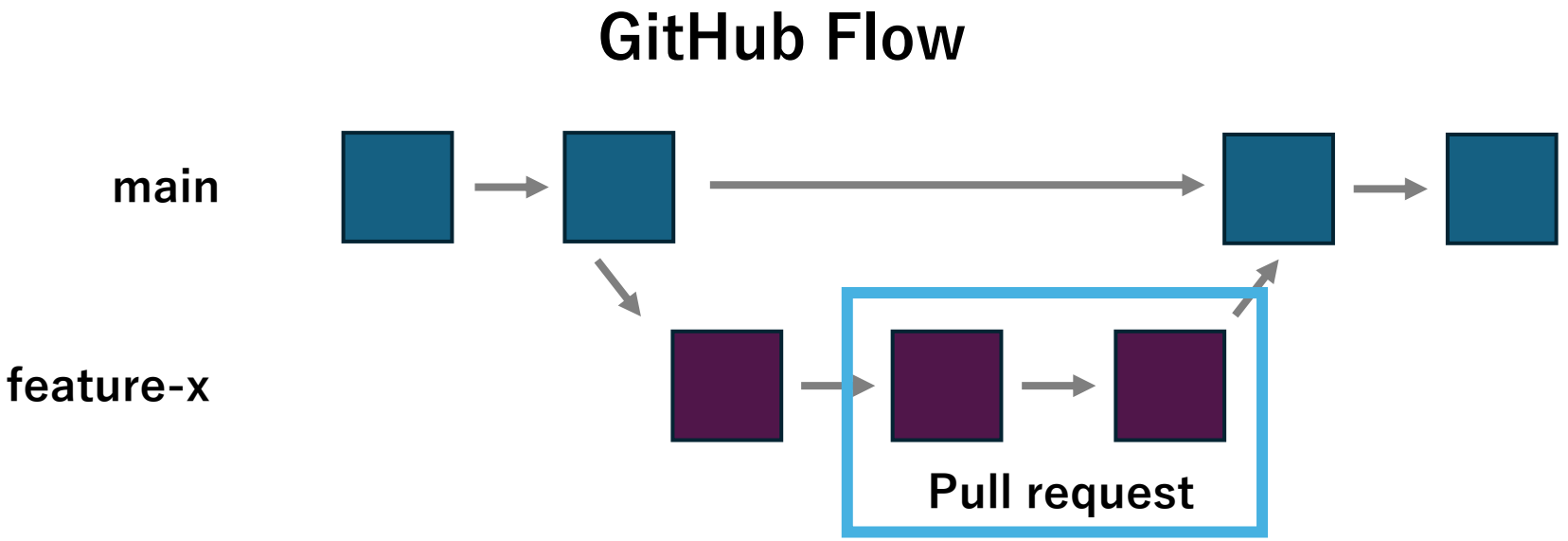
@GitHub

・・・Margeすると、pull requestが閉じられる



GitHubのブランチモデル

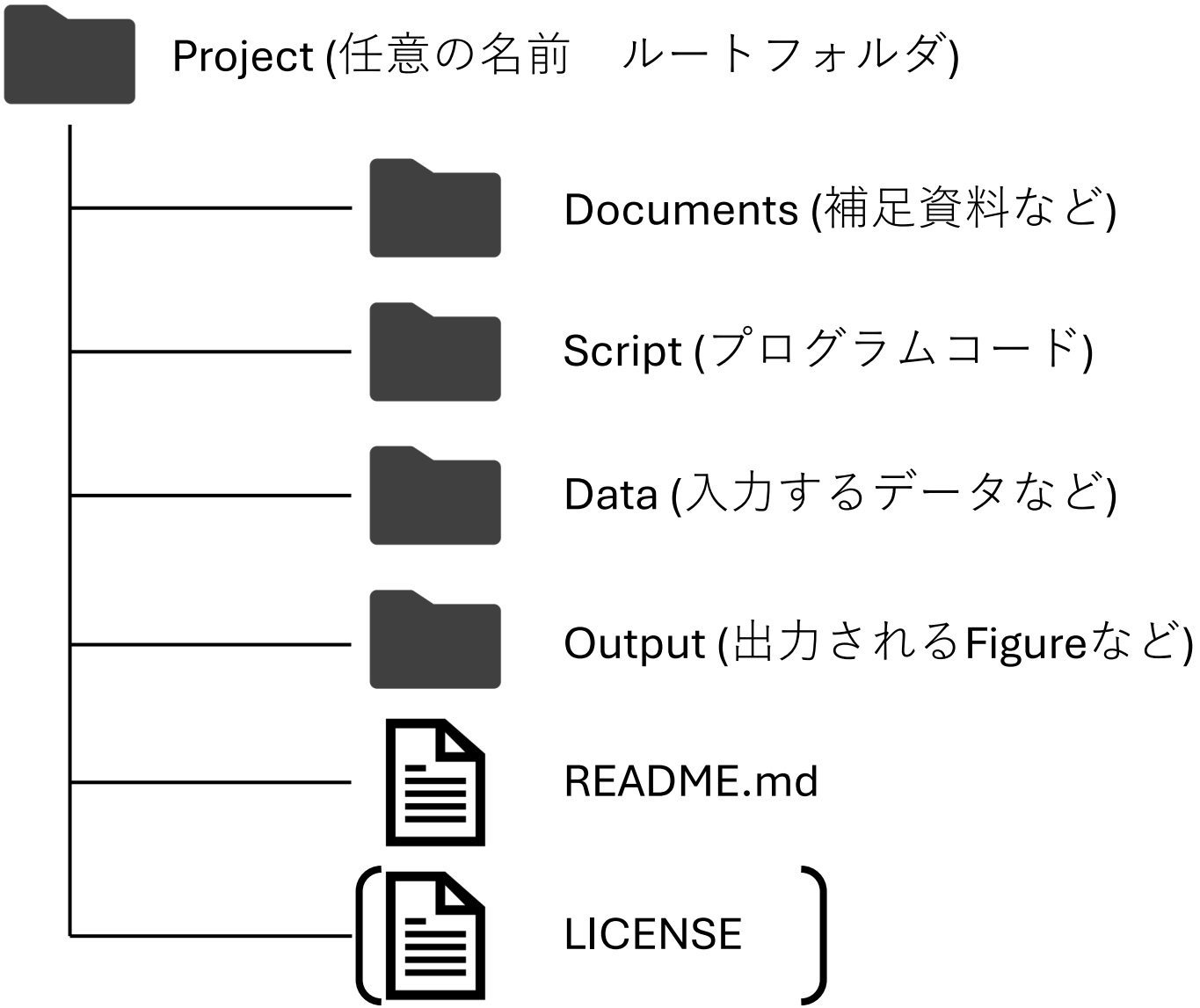
・・・プログラムのバージョン管理方法のモデルフロー
共同制作するときにどのようにするか



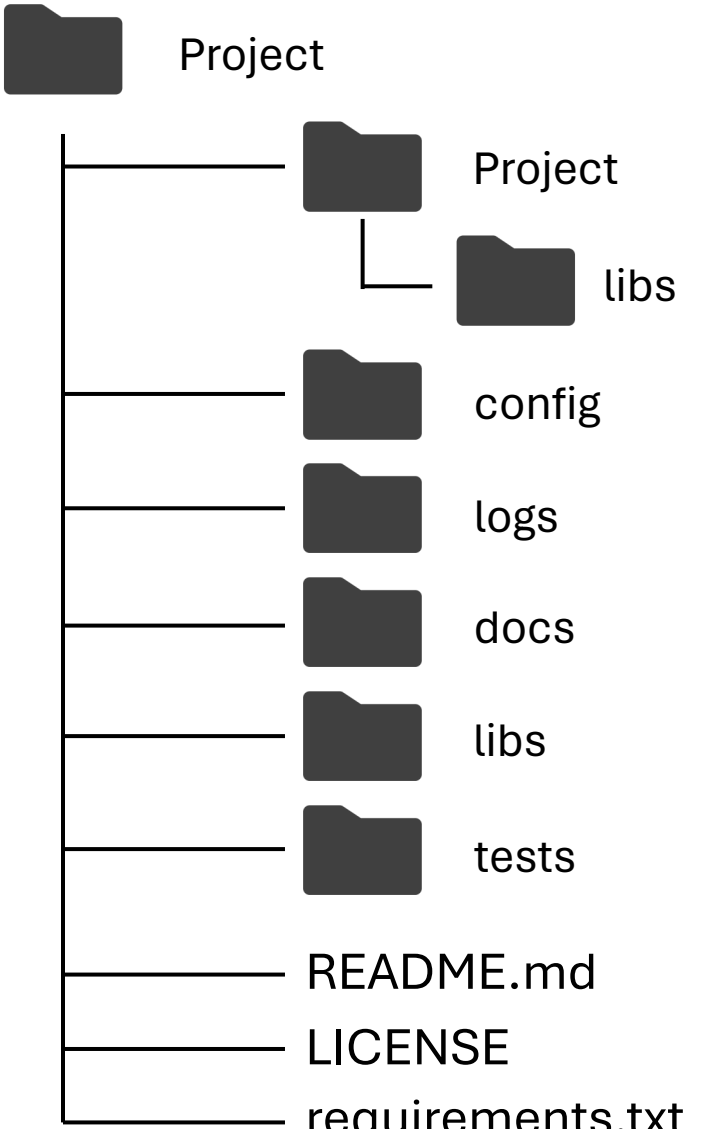
ほかにもgit-flowモデルなどがあり、プログラムの規模やソフトウェアの種類によって変わってくる。

**大切なのは、プログラムの共同管理の方法を
チーム内で共有すること**

プロジェクトのファイル構成の参考（解析コード）



プロジェクトのファイル構成の参考 (OSS)



ルートフォルダ

ソースコード

プロジェクト内でのライブラリの保存

各種設定ファイルの保存

ログファイルの保存

ドキュメント類

他のプロジェクトと共通のパッケージなど

テストコードやテストデータなど

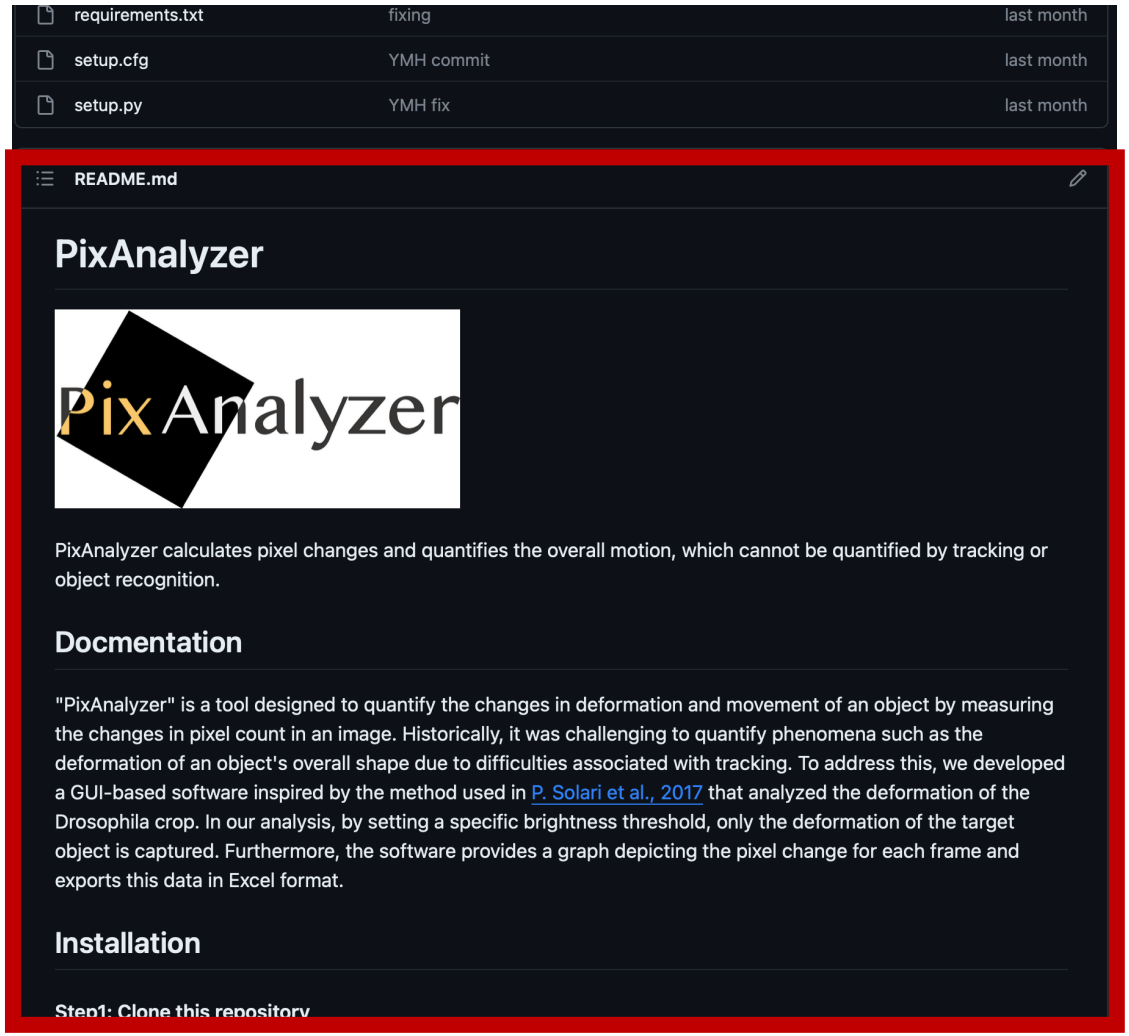
パッケージの説明など

ライセンス情報

パッケージ情報など　yamlなどもある

README.mdについて

・・・プログラムの説明をするもの



GitHub上でプログラムの下に表示される部分

ぱっと見でどのようなプログラムかを理解するために重要！！！

README.mdについて

“マークダウン記法”で書かれる

````
# PixAnalyzer
<!-- ![Logo](img/logo.png) -->
<img src="img/logo.png" width="40%">

PixAnalyzer calculates pixel changes and quantifies the overall mo

## Docmentation
"PixAnalyzer" is a tool designed to quantify the changes in deform

## Installation

#### Step1: Clone this repository

#### Step2: Install [Anaconda](https://www.anaconda.com)

#### Step3: Build an environment using the yaml file

In Anacnda prompt
```
  conda env create -f <path/to/environment.yml>
```
````

マークダウン記法 チートシート

見出し	# 見出しh1 ## 見出しh2	見出しh1 見出しh2
リスト	* テキスト1 　　* テキスト2 　　* テキスト3	• テキスト 　　・テキスト 　　・テキスト
番号付きリスト	1. テキスト1 2. テキスト2 3. テキスト3	1. テキスト1 2. テキスト2 3. テキスト3
空行・改行	行1 _ _(←半角スペース２つ) 行2 行３	行1 行2 行３
リンクの挿入	[タイトル](URL)	タイトル

参考：https://qiita.com/kamorits/items/6f342da395ad57468ae3

マークダウン記法 チートシート

コードの挿入	```言語:タイトル コード ```	
引用	> テキスト1 >> テキスト2	テキスト テキスト
太字	**テキスト**	**テキスト**
斜体	*テキスト*	*テキスト*
水平線	***	

参考：https://qiita.com/kamorits/items/6f342da395ad57468ae3

ライセンスについて

・・・プログラムコードを利用する人へのルール

オープンソース(OS)のソフトウェアの公開が
推奨されているが、自由勝手にプログラムを
自分のもののように公開されるのを防ぎたい！！

ユーザーのプログラム使用の範囲を限定する

Ex. コードの引用元を付けさせる
商用利用不可　など

解析コードに関しても同様に考えなければならい

ライセンスの種類について

MIT, BSD, Apache License 2.0, GPL v2.0 など。。。

自分が適応したい範囲に合わせて
選択する必要がある

一番利用者側の自由度が高いのが**MITライセンス**

・コピー利用、配布、変更の追加、変更を加えたもの再配布、商用利用、有料販売など可

・一切の保証、責任は製作者は取らない

ライセンスの適応の仕方

GitHubのフォルダーの中にLICENCEというのを入れる

適応したいライセンスを選択すると、
GitHubが自動的に作ってくれる

```
MIT License

Copyright (c) 2023 Hayato M Yamanouchi

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.
```

コードを論文に載せるため

・・・**Zenodo**でGitHubと連携して、レポジトリにDOIを発行する

https://upload.wikimedia.org/wikipedia/commons/e/e8/Zenodo-gradient-square.svg

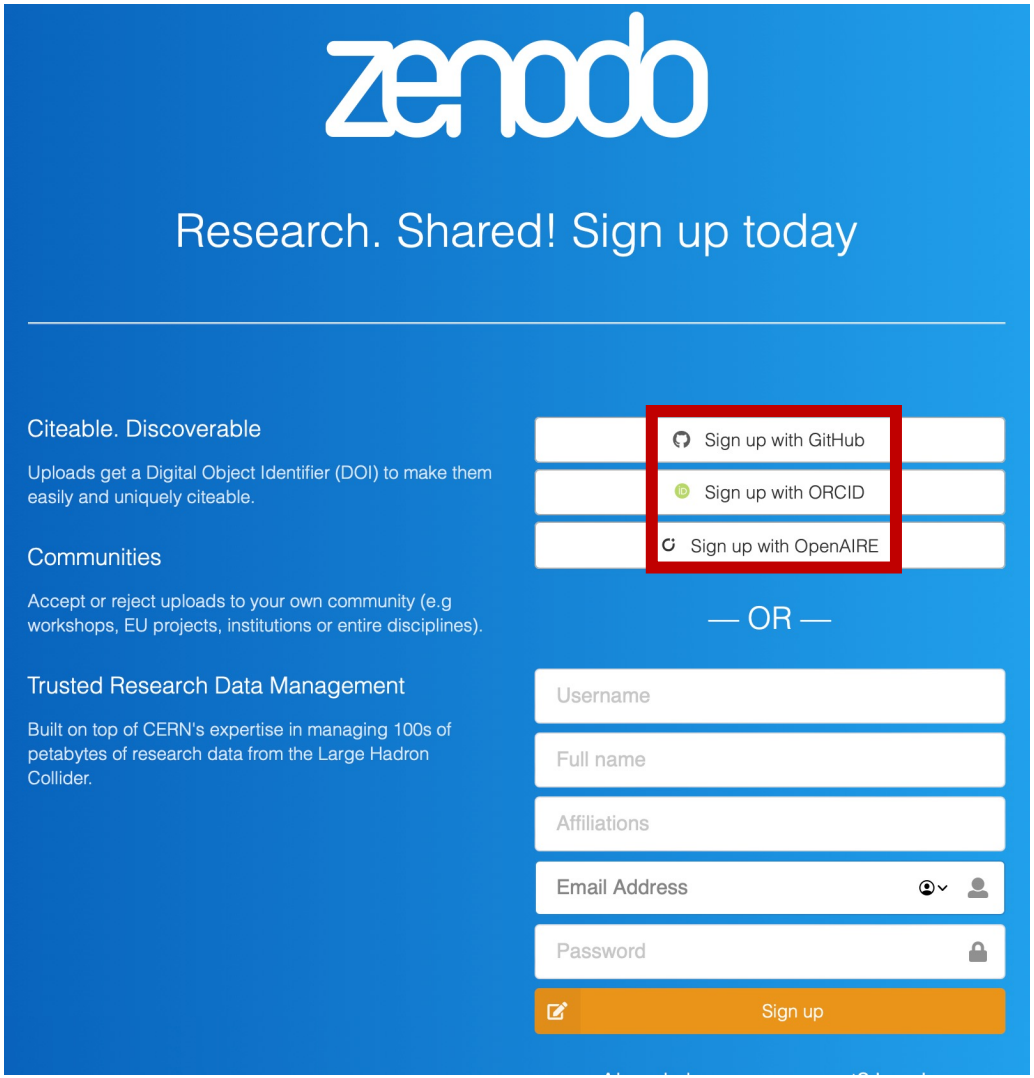
“Zenodoは研究データリポジトリである。研究者がデータセットを置くための場所を提供するためにOpenAIREとCERNによって作られた。2013年に始められ、いかなる研究分野の研究者も50 GBまでファイルをアップロードできるようになった。"

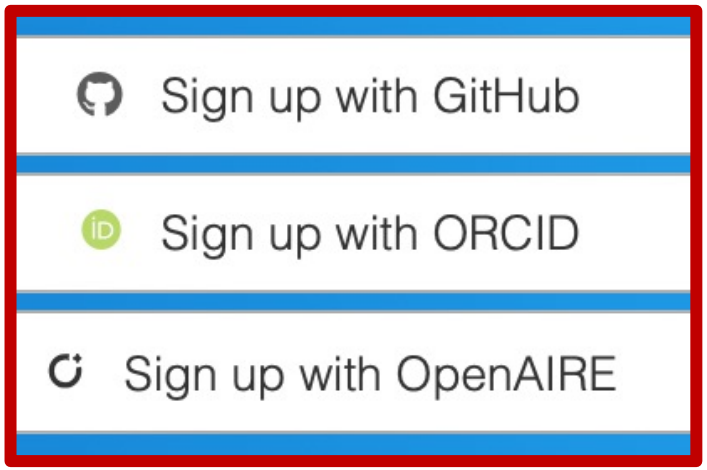
from Wikipedia

もちろんGitHubと連携せず、Zenodo単体でも
データセットの保存、DOI発行は可能

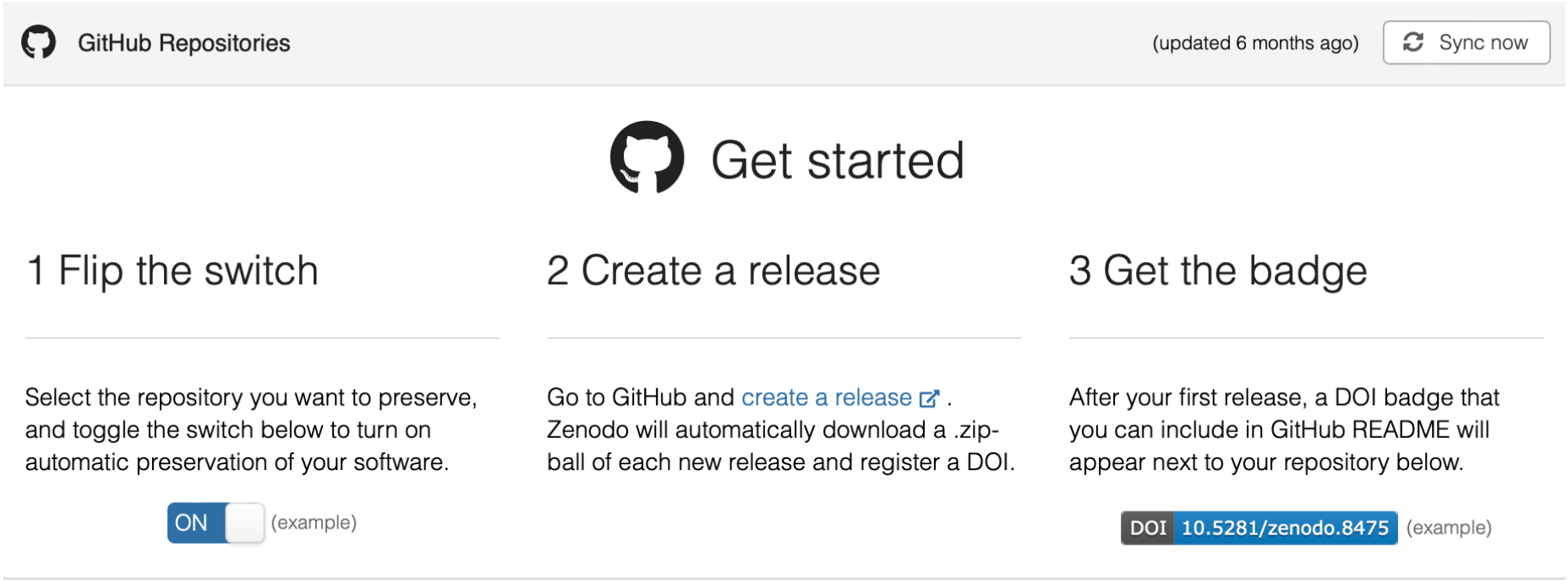
ZenodoとGitHubを連携させる

・・・ZenodoにGitHubアカウントでSign upするだけ！





GitHubのレポジトリにDOIを発行する

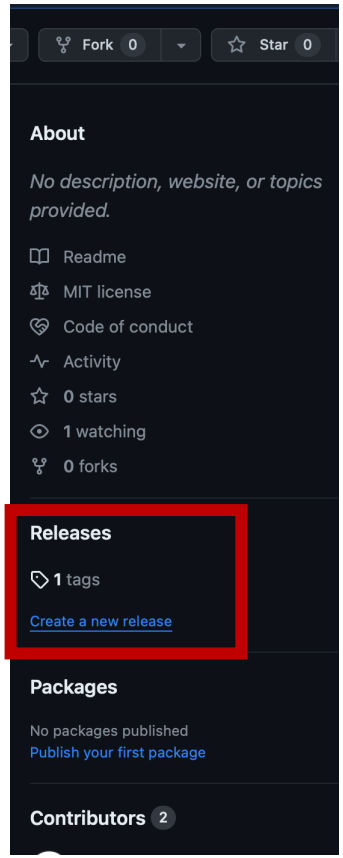


0. レポジトリをPublic にする
1. Zenodoで選択する
2. GitHubで”release”を作成する
3. レポジトリのREADME.mdにDOIの表記を入れる

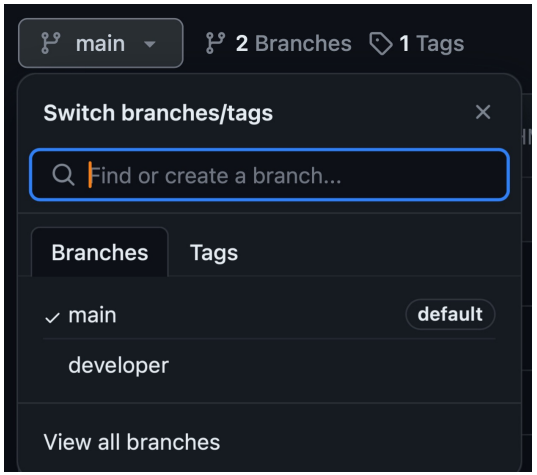
レポジトリのReleasesを作る

・・・コードのバージョンの発行 (ver 1.0.0みたいな)

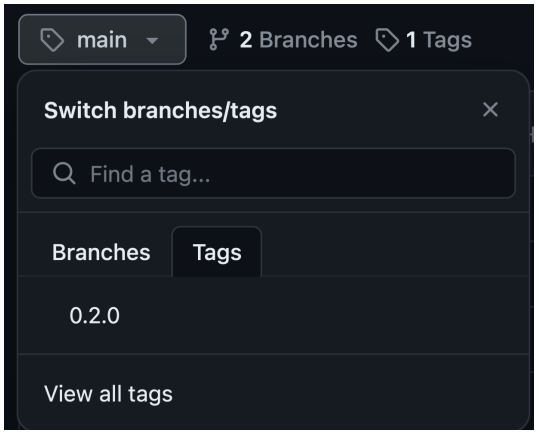
発行することで、その時のバージョンの
コードが保存される



Branches

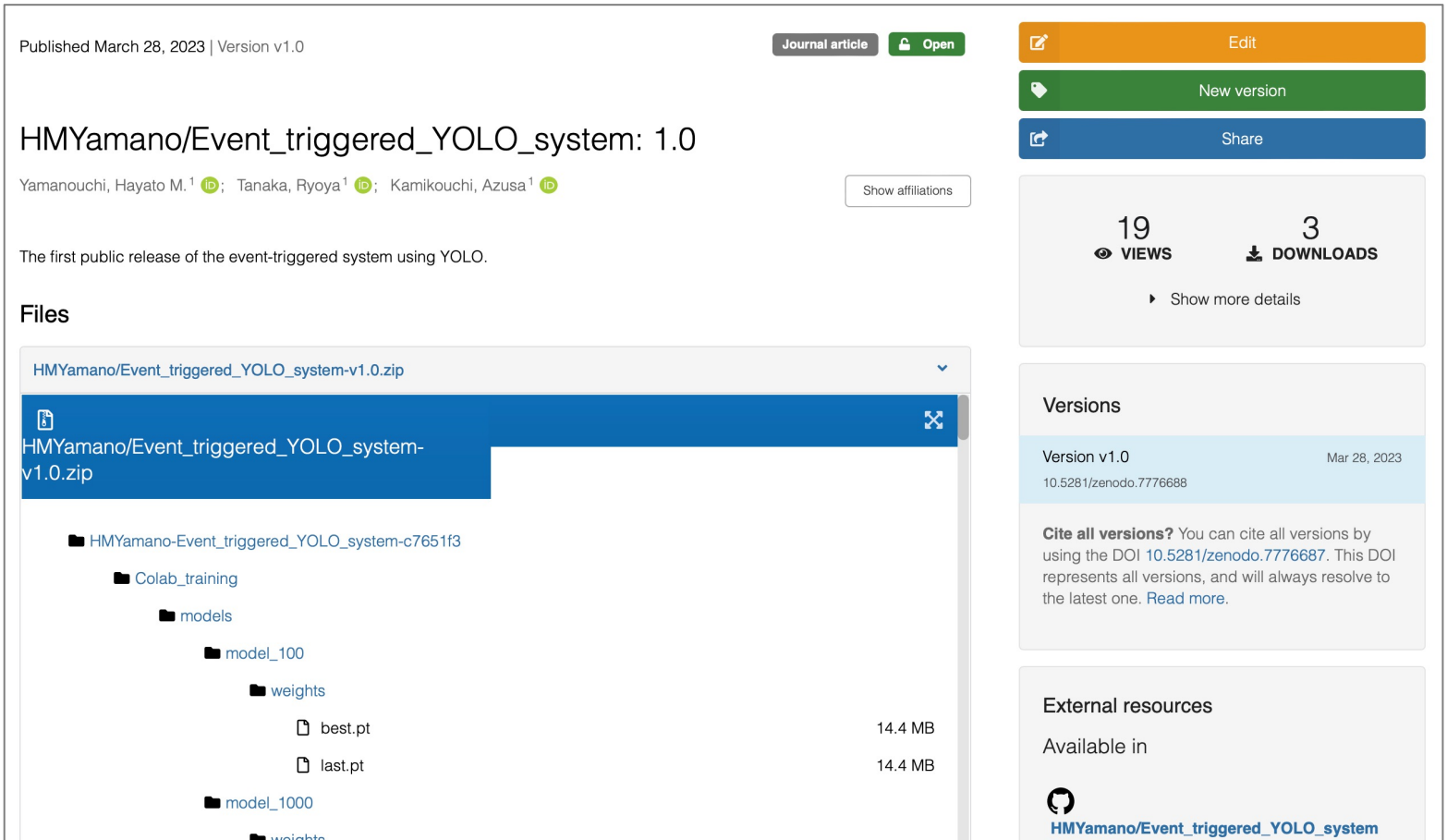


Tags (version)



GitHubのレポジトリにDOIを発行する

Zenodoには、DOIを発行した時のバージョンのフォルダが格納されている。



GitHubやプログラミングを快適に行うために

・・・Visual Studio Code (略VS Code)を使いましょう！！
(Visual Studioと混合しないように)

・WindowsやMacで使えるコードエディタ

・拡張機能でGitHubへのプッシュ/プルなどが可能に

・.yamlや.mdファイルなどの様々な拡張子のファイルの
表示、編集ができる

・使いやすい、知見もいっぱいある

https://upload.wikimedia.org/wikipedia/commons/9/9a/Visual_Studio_Code_1.35_icon.svg

+@要素

・・・ディスカッションやプロジェクトの進行管理など
多様な機能がある

- **Issues**・・・プログラムで問題があったときなどに
Issuesを作成することで、その問題の
対応などを管理できる

- **Actions**・・・GitHubの拡張パッケージ
Ex. プッシュ時に自動的にコードを整える

- **Insights**・・・プロジェクトの全体の動きを見れる