

GitHubを使ったチーム制作

- GitHubを使ってチーム制作を円滑に進める
- GitHubにはコラボレーション機能があり、ひとつのリポジトリを複数人で更新することができるようになっている
- リポジトリ内のファイルを各自で追加や編集をして、結果を統合(マージ)することができる

GitHubを使ったチーム制作

- GitHubで共同開発する手順

- ① 代表者がGitHub上にリポジトリを作成
- ② 代表者が作成したリポジトリにチームメンバーを招待
- ③ チームメンバー宛にメールが届くので、招待を受けるボタンをクリック
- ④ チームメンバーがリポジトリをクローン(複製)する

GitHubを使ったチーム制作

- 代表者としてリーダーもしくはメインプログラマのいずれかが、まずは共同作業をするリポジトリをGitHub Desktopから作成する
- [File]メニューから[New repository...]を選択する
- 表示された画面でレポジトリ名を入力

GitHubを使ったチーム制作

- [Name]には共同開発に使用するリポジトリ名を入力
- [Local path]はC:¥GitHubのままでOK
- [Git ignore]は
VisualStudioを選択

↑
超重要！

Create a new repository

Name
TeamCpp

Description

Local path
C:¥GitHub Choose...

☐ Initialize this repository with a README

Git ignore
VisualStudio

License
None

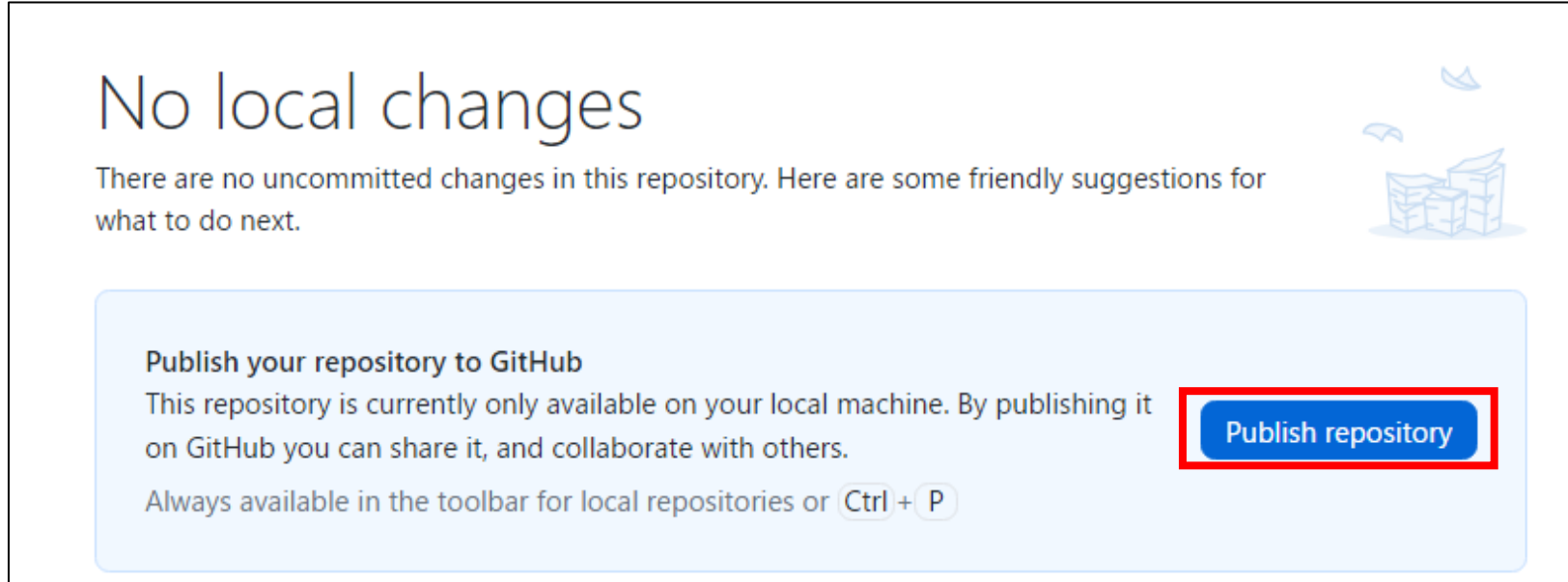
The repository will be created at C:\GitHub\TeamCpp.

Create repository Cancel

↑
入力を確認してクリック

GitHubを使ったチーム制作

- [Publish repository]ボタンをクリックする



GitHubを使ったチーム制作

- [Keep this code private]にチェックを入れてから、[Publish repository]をクリック

The screenshot shows the 'Publish repository' dialog box in GitHub. The 'Name' field is filled with 'testrepd'. The 'Keep this code private' checkbox is checked and highlighted with a red box. The 'Publish repository' button is also highlighted with a red box.

Publish repository

GitHub.com GitHub Enterprise

Name
testrepd

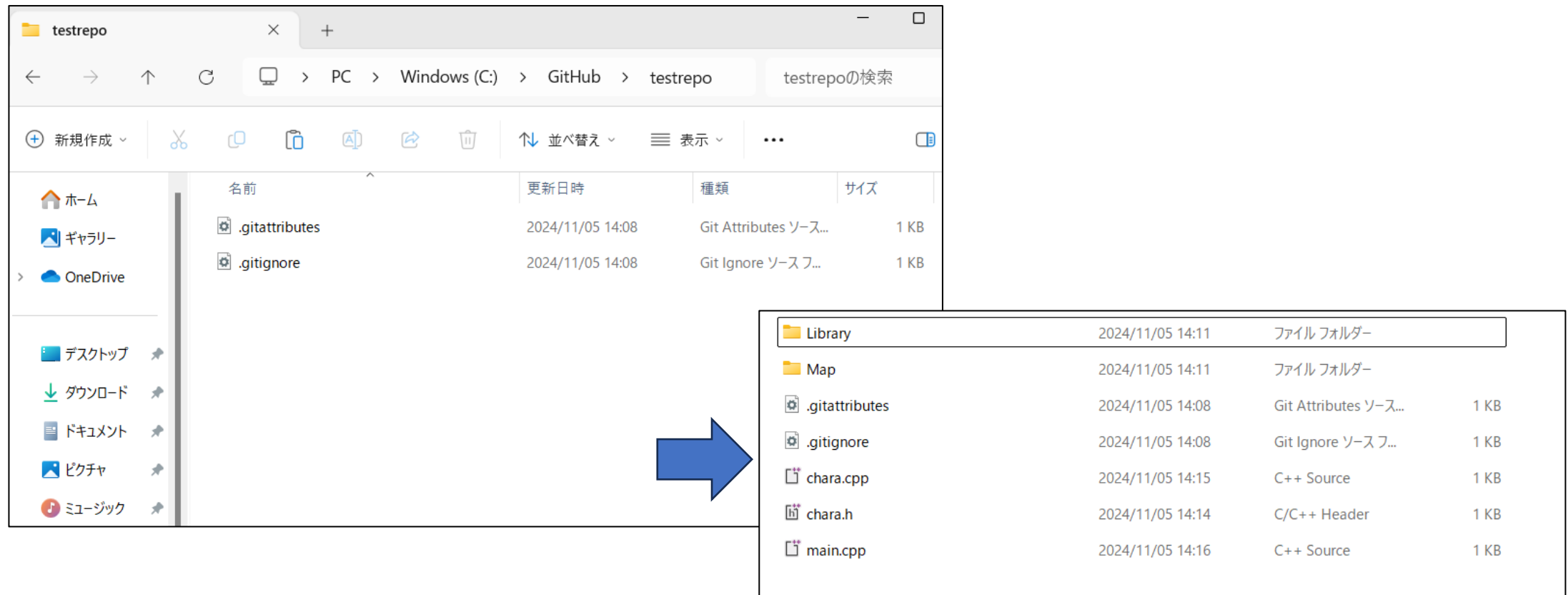
Description

☒ Keep this code private

Publish repository Cancel

GitHubを使ったチーム制作

- C:¥GitHub¥リポジトリ名のフォルダ内に開発に必要なものをすべてコピー

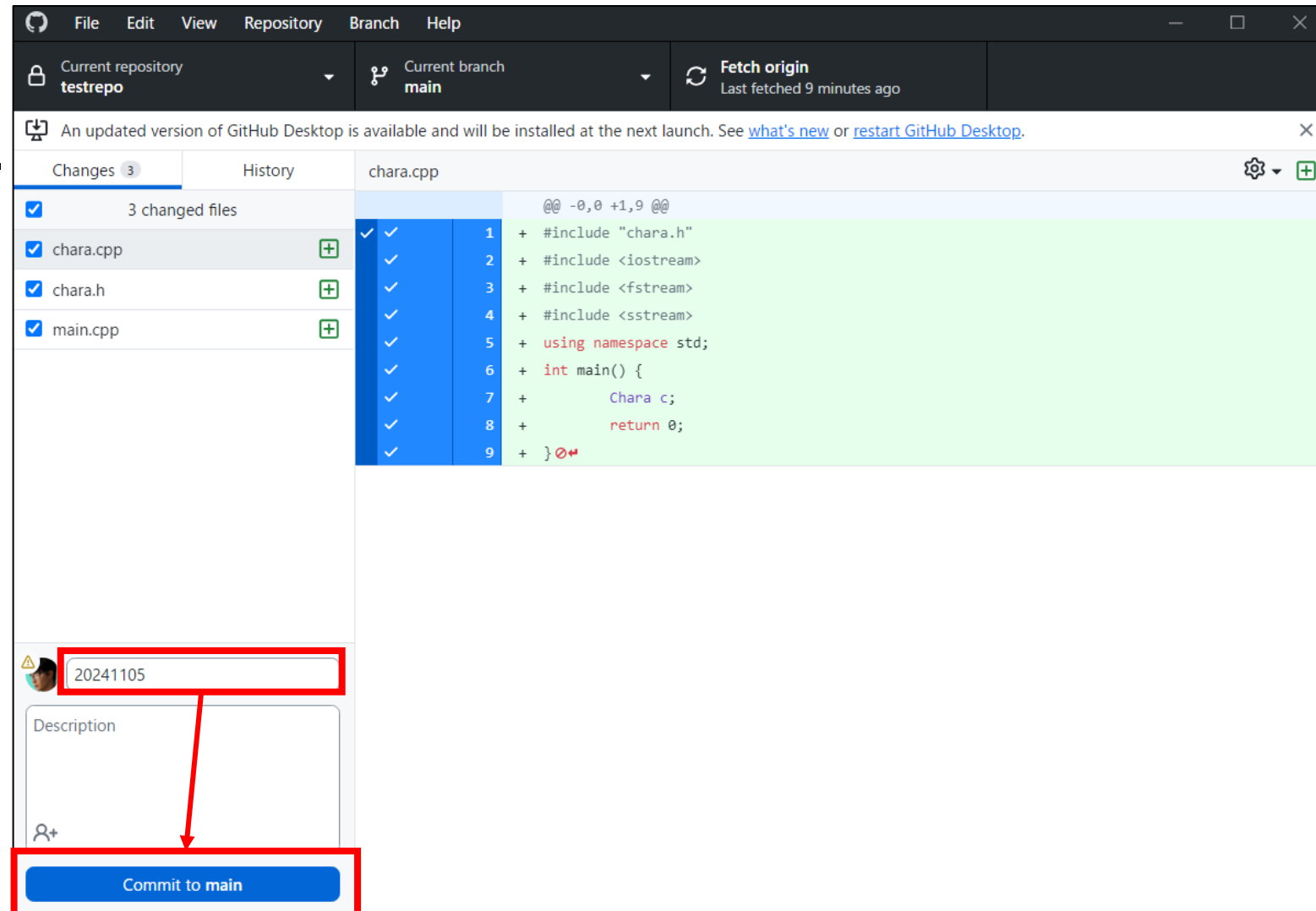


GitHubを使ったチーム制作

- フォルダに全
ファイルをコピー
した後、コメント
を入力して

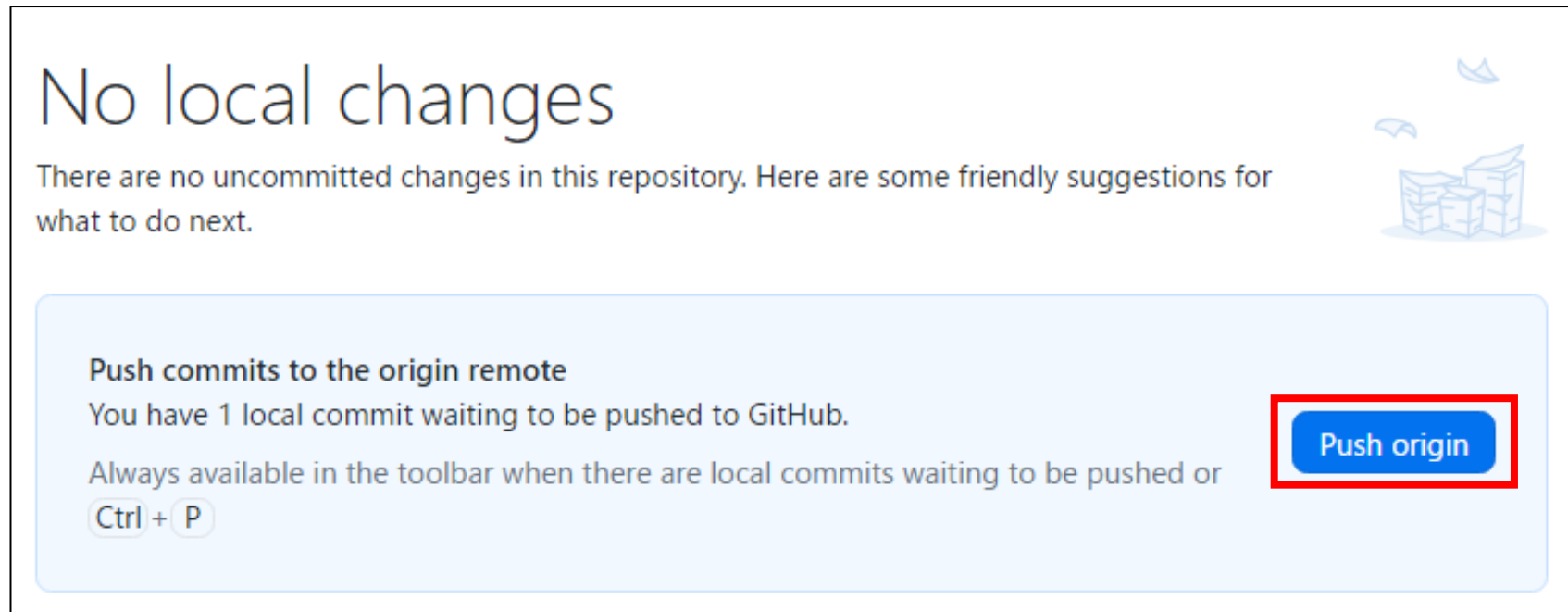
**[Commit
to main]**

をクリック



GitHubを使ったチーム制作

[Push origin]をクリックして、GitHubへアップロードを行う



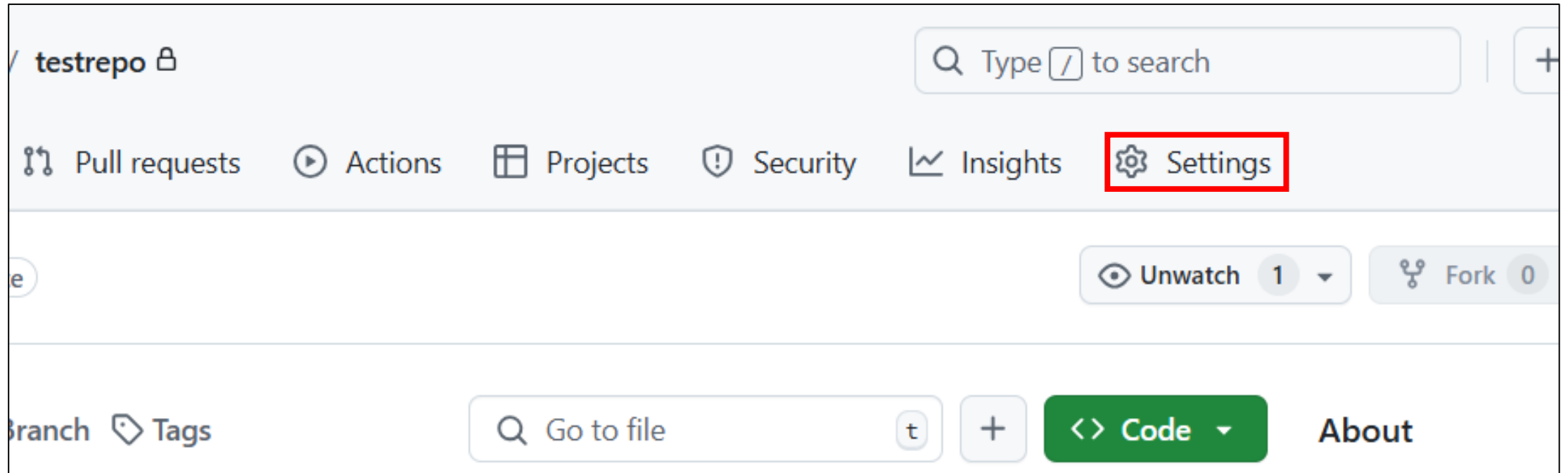
GitHubを使ったチーム制作

- `github.com`にアクセスして、リポジトリ一覧の中から共同作業用のリポジトリを選択



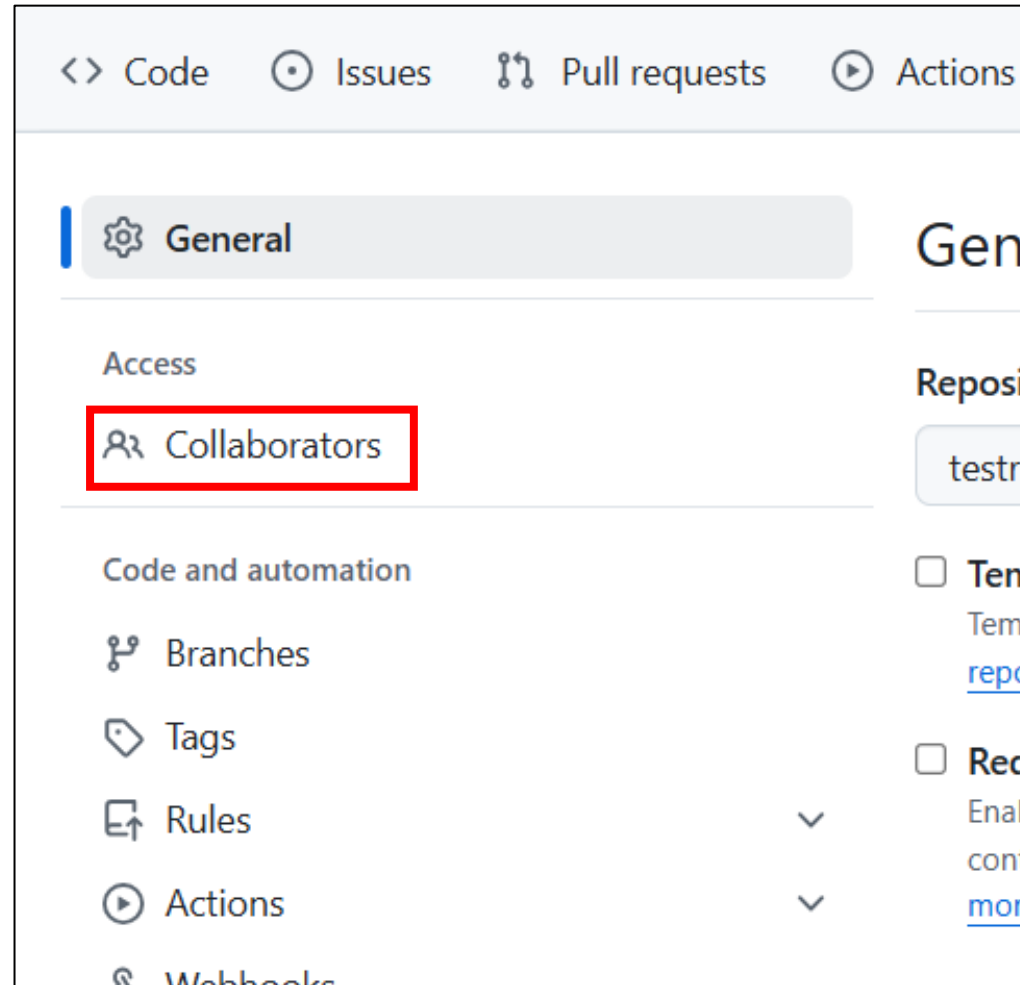
GitHubを使ったチーム制作

- [Settings]をクリック



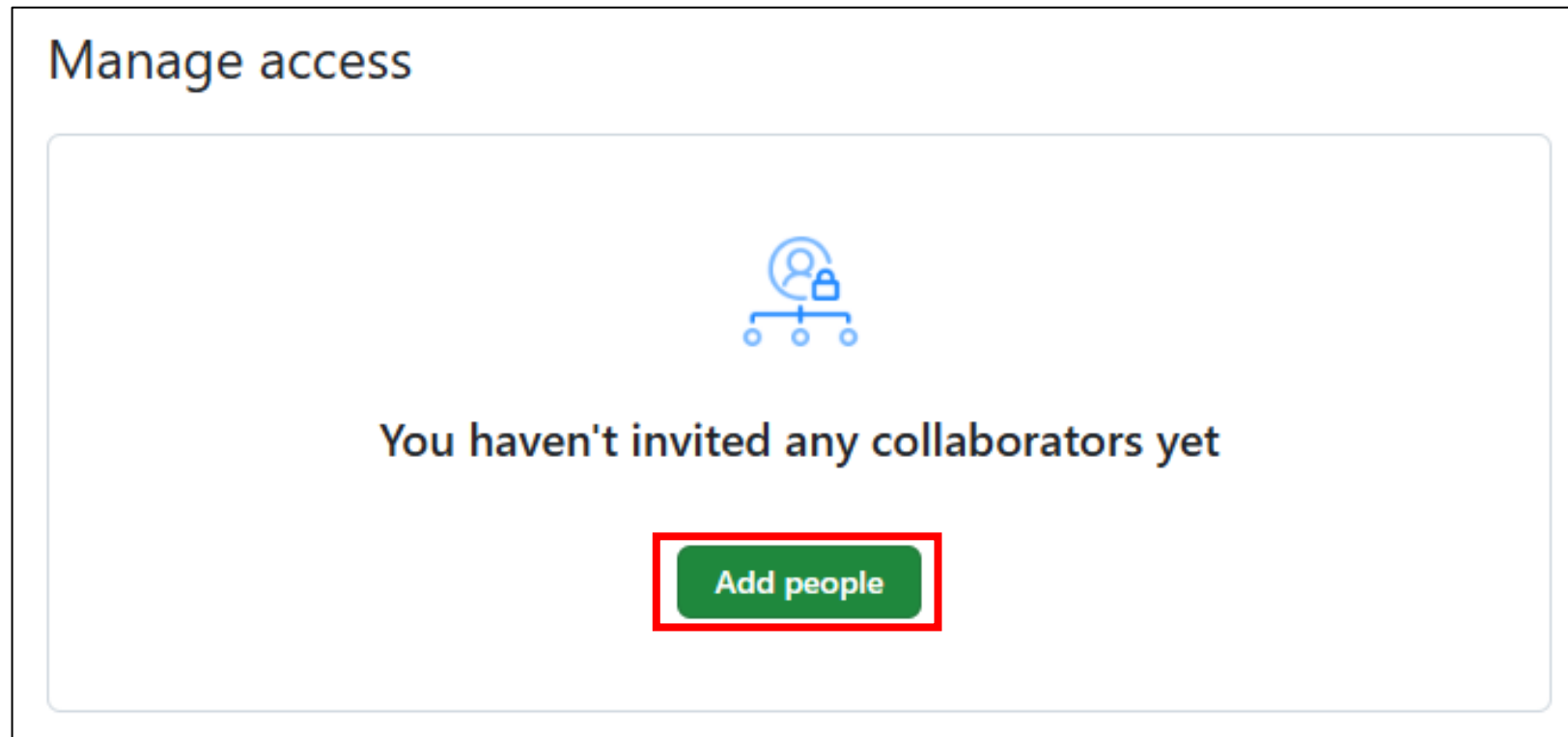
GitHubを使ったチーム制作

- メニューから[Collaborators]をクリック



GitHubを使ったチーム制作

[Add people]をクリック



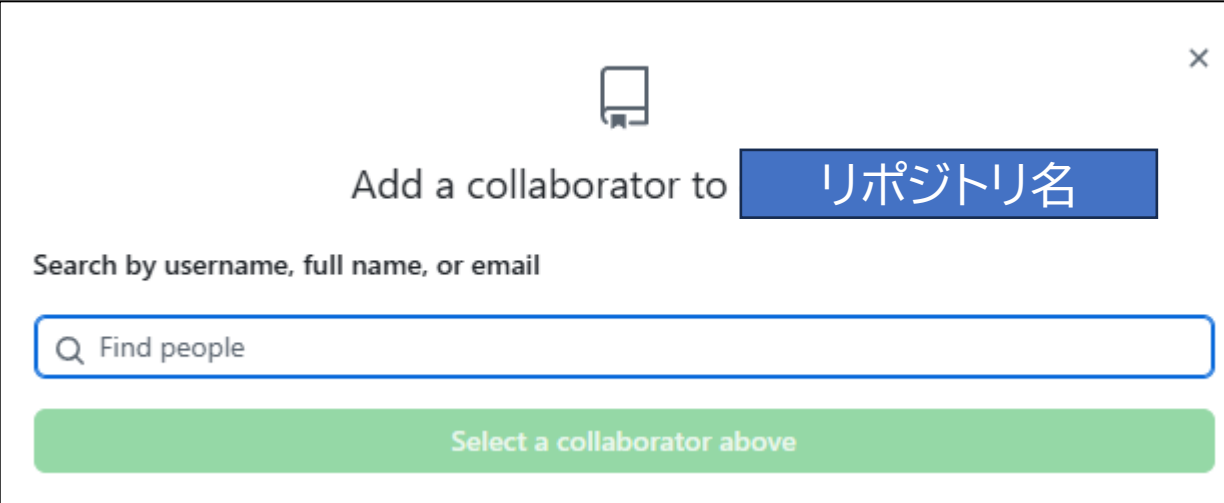
GitHubを使ったチーム制作

- この画面が表示されたら、チームメンバーのメールアドレスを入力すると一覧が表示されるので

[Select a collaborator above]

をメンバーを選択してからクリックする

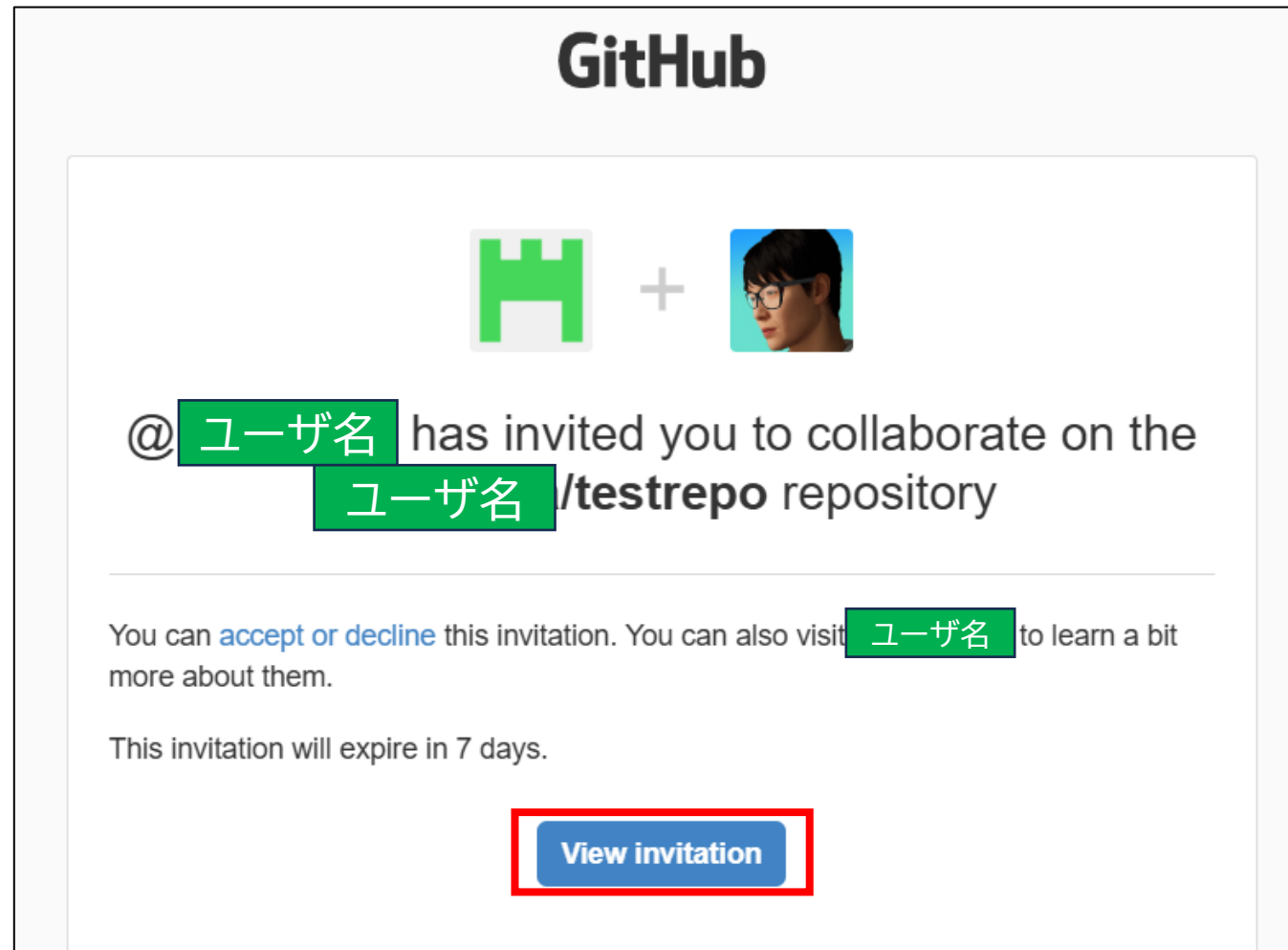
※チームメンバーがGitHubに登録済みでないだとダメ



The screenshot shows a modal dialog box for adding a collaborator to a repository. At the top, there is a close button (X) and a mobile device icon. Below the icon, the text 'Add a collaborator to' is followed by a blue button labeled 'リポジトリ名' (Repository Name). Underneath, it says 'Search by username, full name, or email'. There is a search input field with a magnifying glass icon and the placeholder text 'Find people'. At the bottom, there is a green button labeled 'Select a collaborator above'.

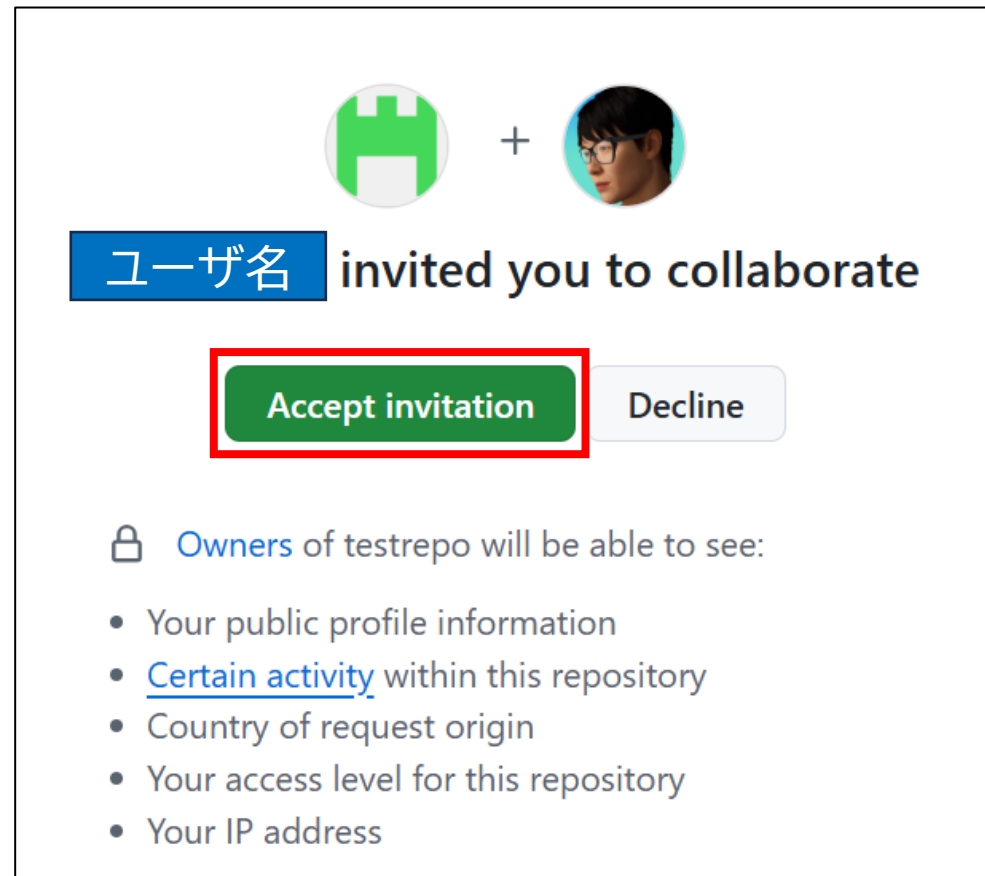
GitHubを使ったチーム制作

- 招待メールが届くので **[View invitation]** をクリック



GitHubを使ったチーム制作

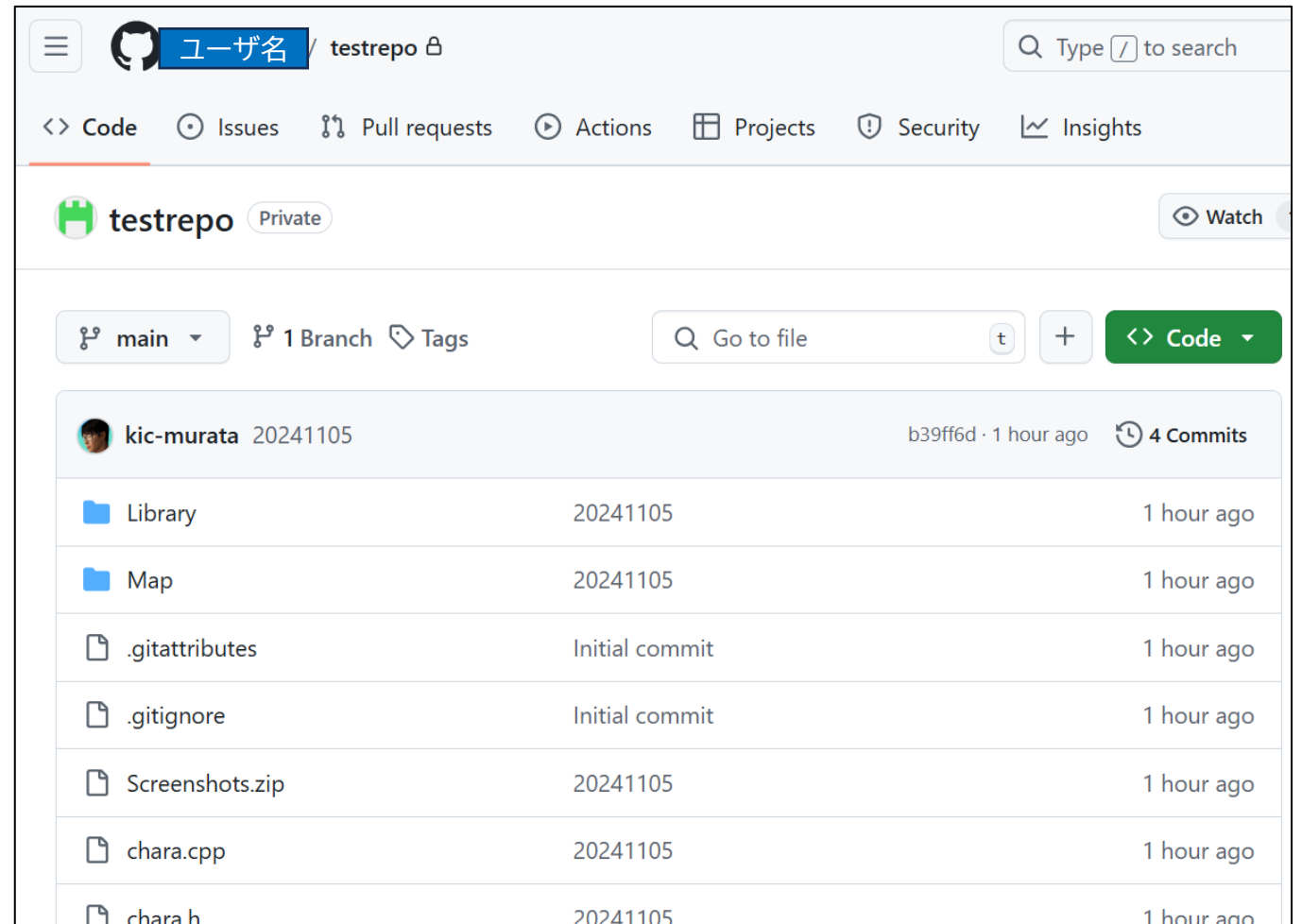
- 招待を許可するかの画面が表示されるので
[Accept invitation]をクリック



404エラーが出た場合
GitHubにサインイン
してください

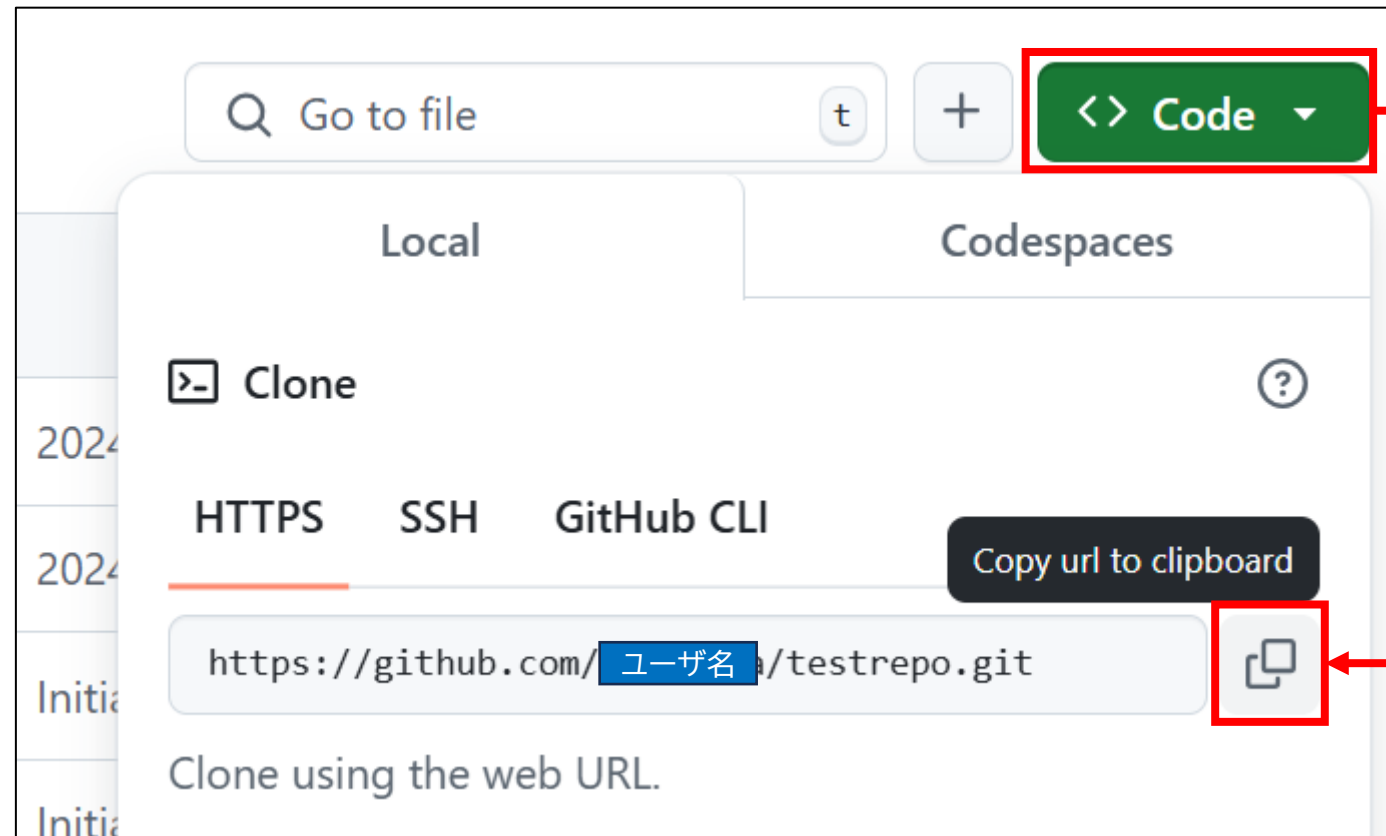
GitHubを使ったチーム制作

- GitHubの画面に招待されたリポジトリの情報が表示されればOK



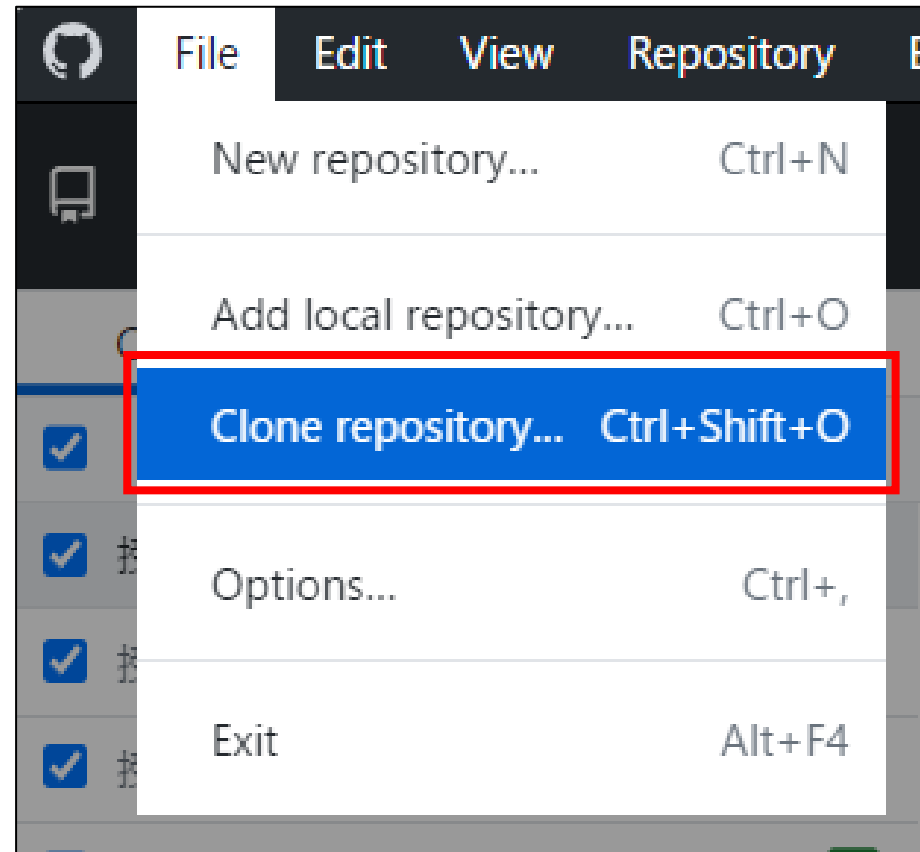
GitHubを使ったチーム制作

[<>Code]のボタンをクリックして、コピーボタンをクリックする



GitHubを使ったチーム制作

- GitHub Desktopを起動して、
[File]>[Clone repository]をクリック



GitHubを使ったチーム制作

- [URL]タブをクリック、URL欄に先ほどコピーしたURLを貼り付ける。LocalPathはC:¥GitHub¥リポジトリ名になる

コピーしたURLを
ここに貼り付ける

Clone a repository

GitHub.com GitHub Enterprise **URL**

Repository URL or GitHub username and repository
(hubot/cool-repo)

https://github.com/ユーザ名/testrepo.git

Local path

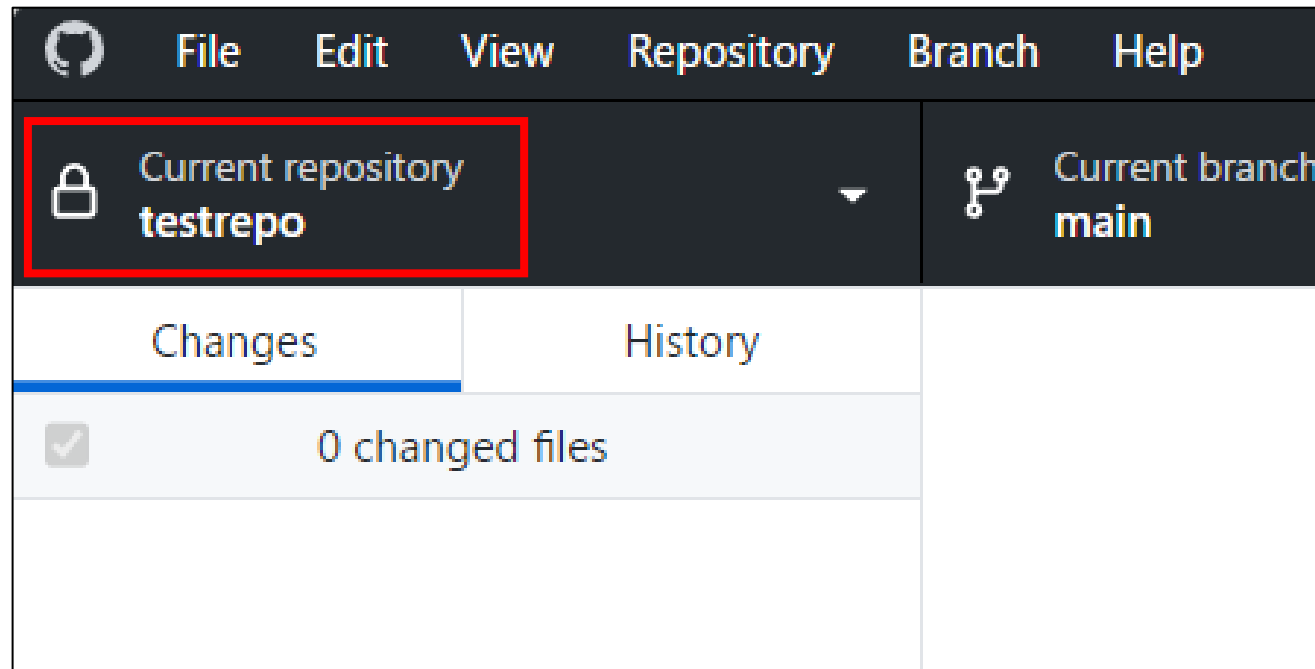
C:¥GitHub¥testrepd Choose...

入力を確認してクリック

Clone Cancel

GitHubを使ったチーム制作

- [Current repository]がクローンしたリポジトリ名になっていればクローン(複製)完了



GitHubを使ったチーム制作

- そもそも「**リポジトリ**」って何？

開発に必要なファイルやフォルダの**保存場所**のこと

- **リポジトリ**は2種類ある！

ローカルリポジトリ: PC内の保存場所

リモートリポジトリ: **GitHub.com**上の保存場所

GitHubを使ったチーム制作

- リポジトリを最新に保つには？

ローカルリポジトリとリモートリポジトリの内容を全く同じに同期する必要がある

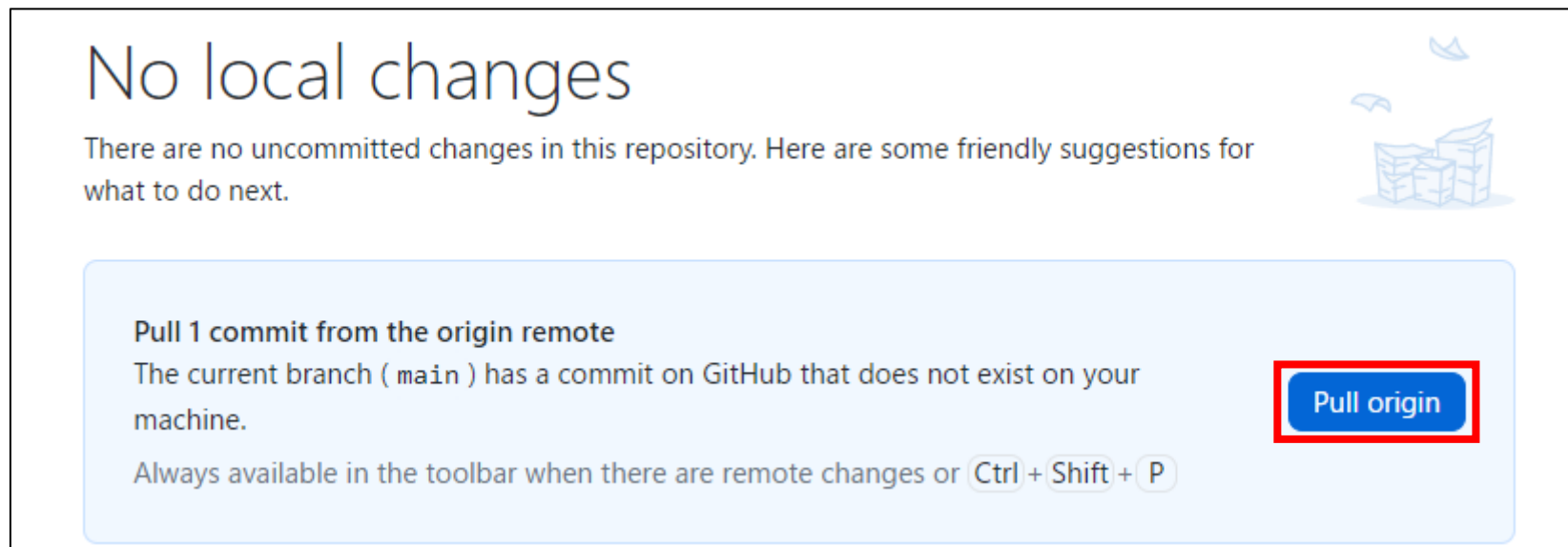
- クローンしたものを編集した場合は、ローカルのほうが新しくなる

→ **Push**という作業をしてリモートも最新にする

GitHubを使ったチーム制作

- 共同開発している誰かがPushしたことでリモートのほうが自分のPC内のローカルリポジトリより新しくなってしまった場合

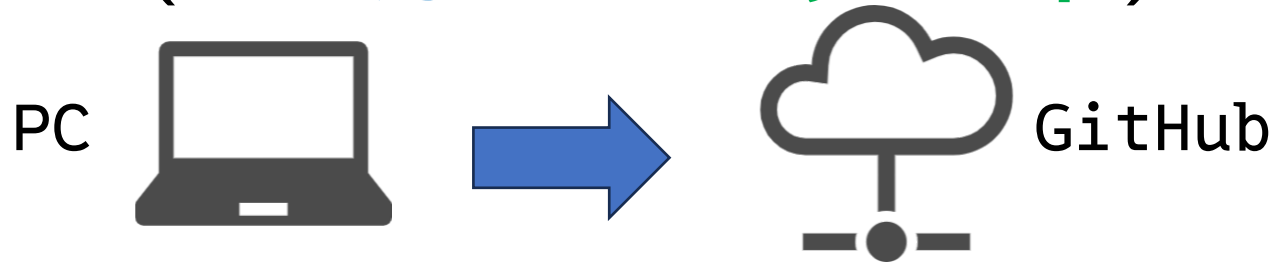
→ Pullという作業をしてローカルを最新に



GitHubを使ったチーム制作

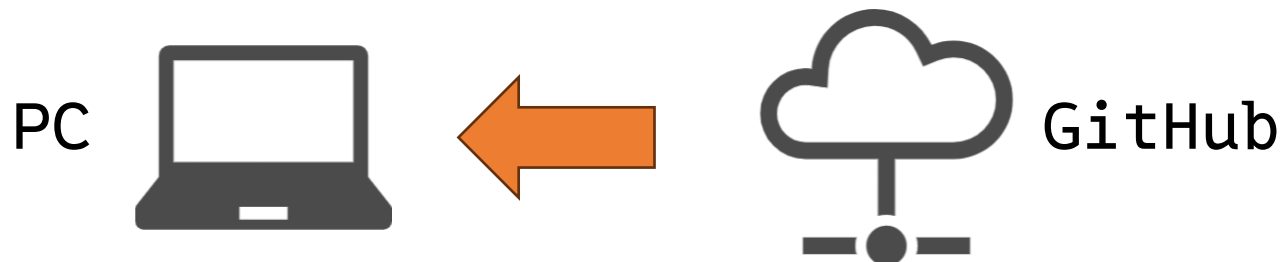
• Push

ローカルリポジトリの更新内容をリモートリポジトリへコピー(ローカル → リモート)



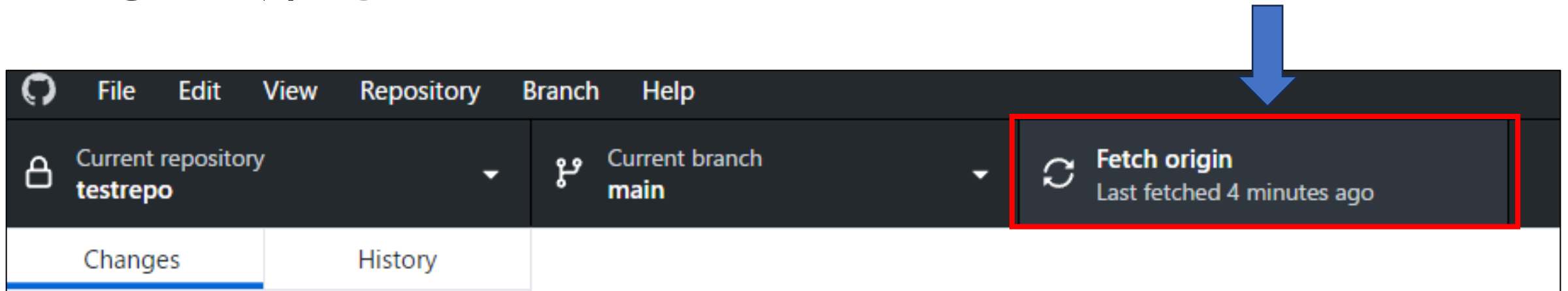
• Pull

リモートリポジトリから最新の内容をローカルリポジトリへコピー(リモート → ローカル)



GitHubを使ったチーム制作

- ローカルとリモートのどちらが新しいかわからないときは[Fetch]することでどちらが最新かを判定してくれる



PushかPullかわからない場合はFetchをする！

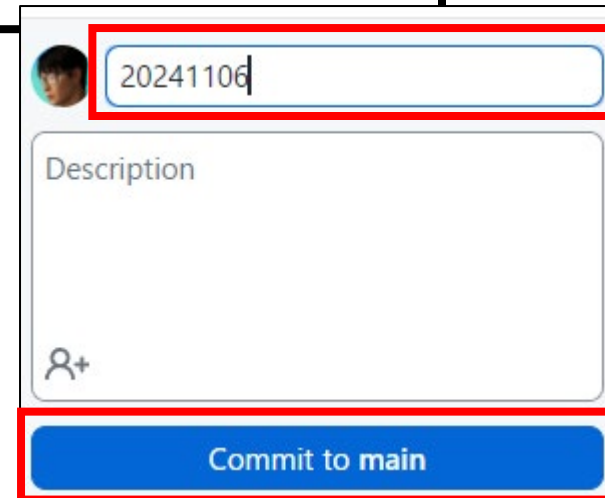
GitHubを使ったチーム制作

• Push

ローカルリポジトリの更新内容をリモートリポジトリへコピー(ローカル → リモート)

Pushする前にコミット(Commit)する必要がある(必須作業！)

コミットはローカルで更新したファイルの中で、どれをリモートへ送信するかリストを作るようなもので、同時にファイルにコメントを付ける

A screenshot of the GitHub commit interface. It shows a user profile picture and a text input field containing '20241106'. Below this is a 'Description' label and a text area. At the bottom, there is a blue button labeled 'Commit to main'. Red rectangular boxes highlight the user profile area, the text input field, and the 'Commit to main' button.

20241106

Description

Commit to main

GitHubを使ったチーム制作

- ・チーム制作を行う上で大切なこと

誰かがリモートリポジトリを勝手に更新して、
バグのあるプログラムに更新してしまった場合、
それをローカルへPullすると、バグが混入した
プログラムがチーム全員に同期されてしまう...

そういったことが起こらないようにする必要がある

GitHubを使ったチーム制作

- ブランチ(Branch)を使った並列作業

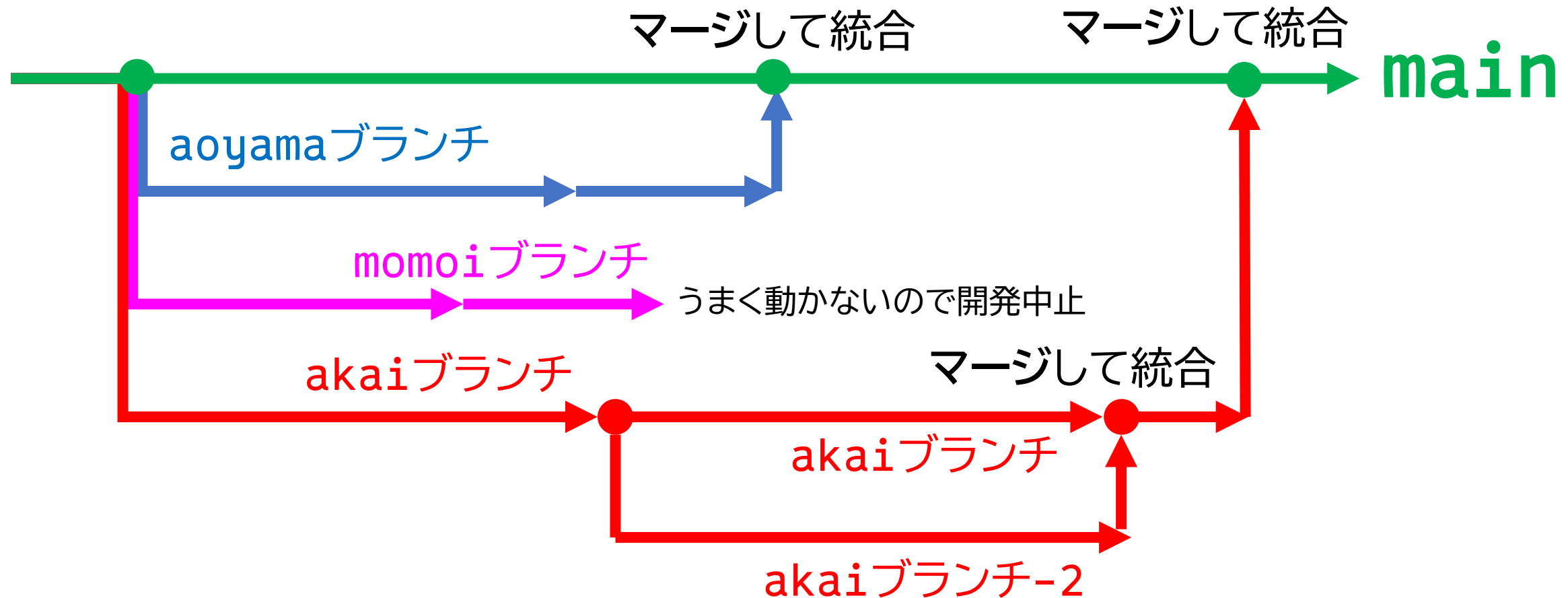
ブランチとは簡単にいうと分岐のこと

作成したばかりのリポジトリはmainブランチしかない

mainブランチから分岐して開発することで、作業を分担して、バグの混入を防ぐことができる

GitHubを使ったチーム制作

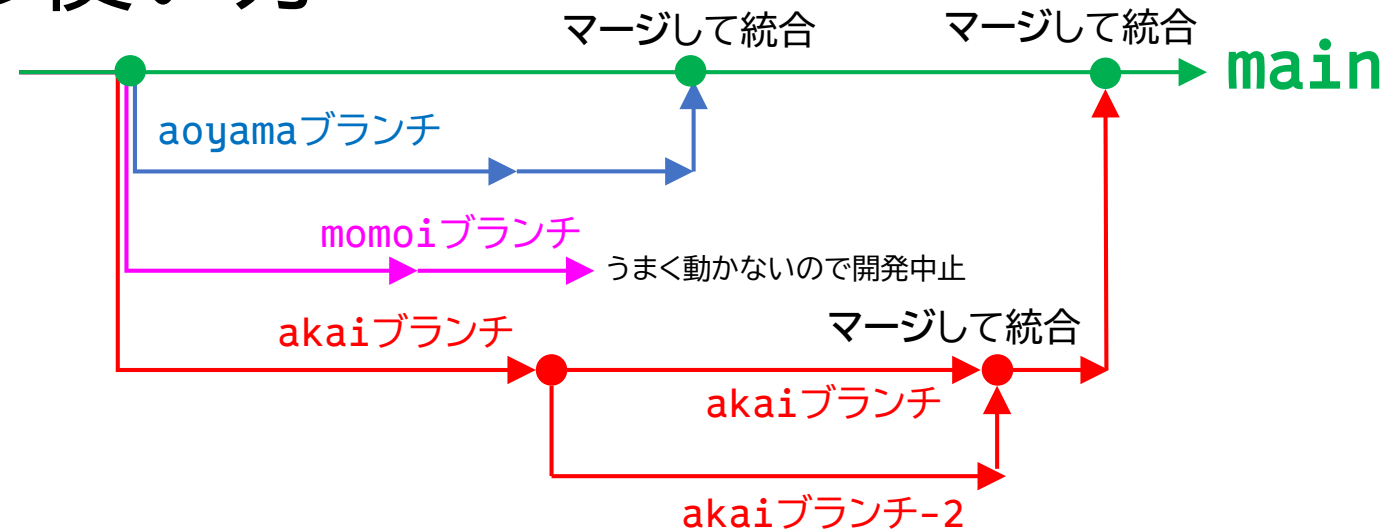
- ブランチ(Branch)のイメージ



GitHubを使ったチーム制作

• ブランチ(Branch)の使い方

チームメンバー毎に
ブランチを作成する
mainでは作業しない！



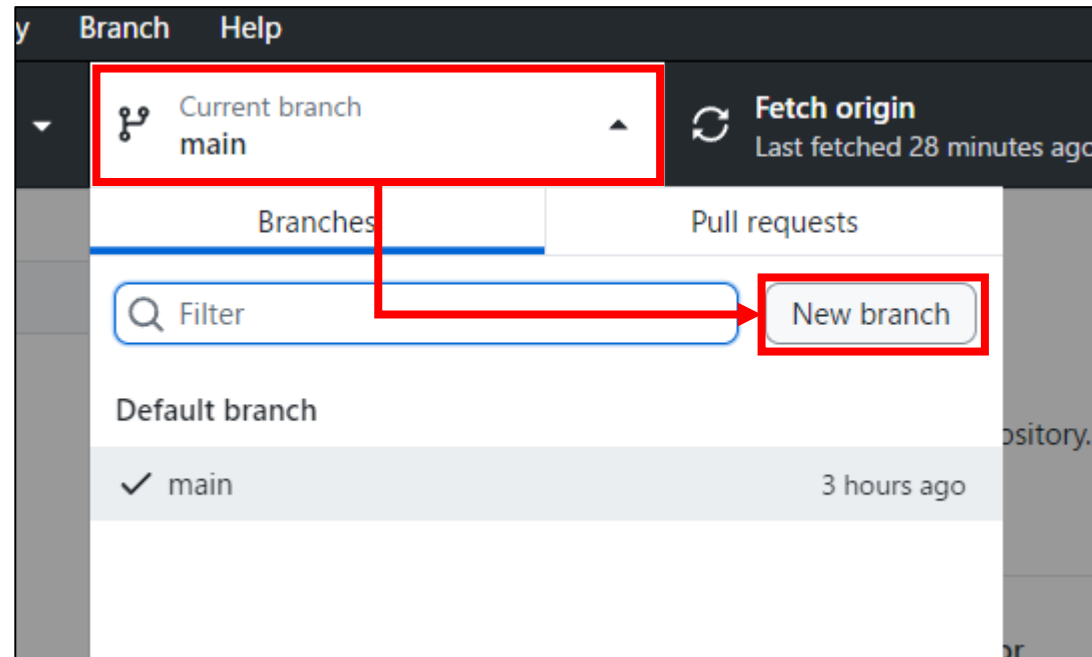
ブランチで作業を行い、うまく
動けば**main**のほうへ**統合(マージ)**する

ブランチをさらにブランチさせることも可能

GitHubを使ったチーム制作

- ブランチ(Branch)の作り方

Current branchをクリックすると以下の画面が表示されるので[New branch]をクリック



GitHubを使ったチーム制作

- ブランチ(Branch)の作り方

ブランチ名を入力して[Create branch]をクリックする

ブランチ名はメンバー名にしておく

Create a branch

Name

murata

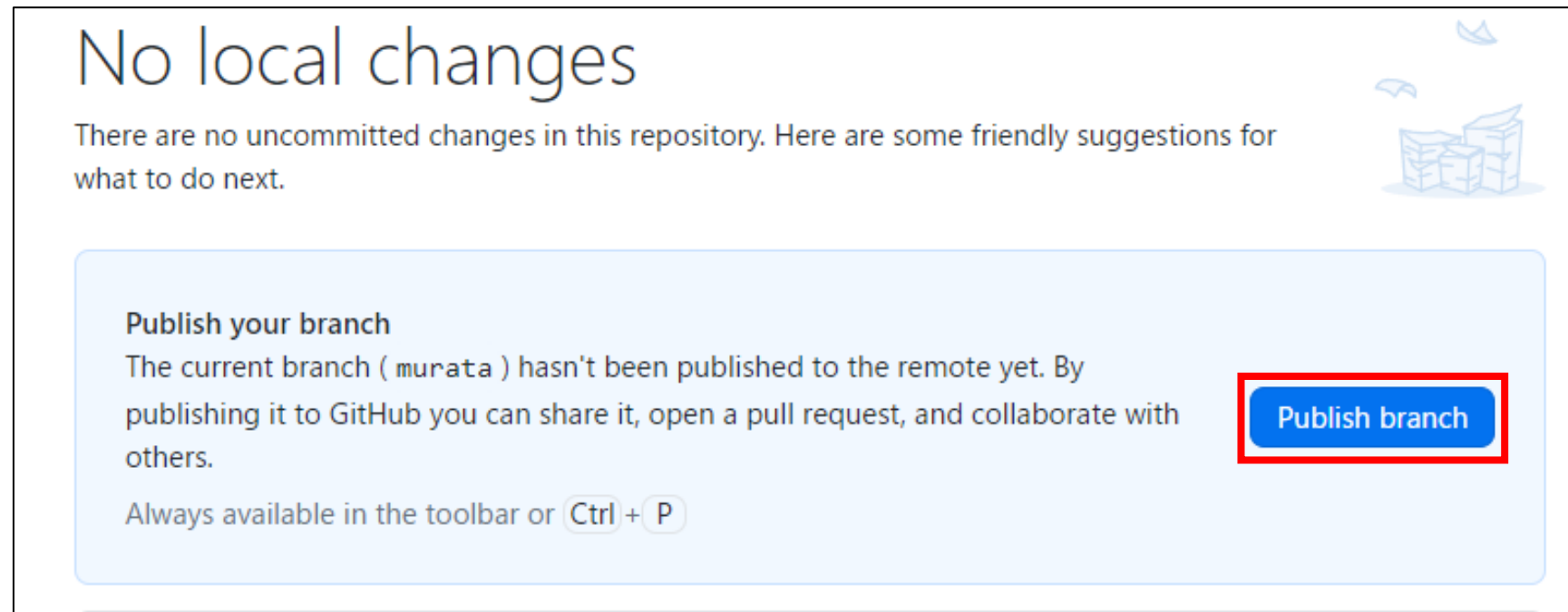
Your new branch will be based on your currently checked out branch (main). main is the [default branch](#) for your repository.

Create branch Cancel

GitHubを使ったチーム制作

- ブランチ(Branch)の作り方

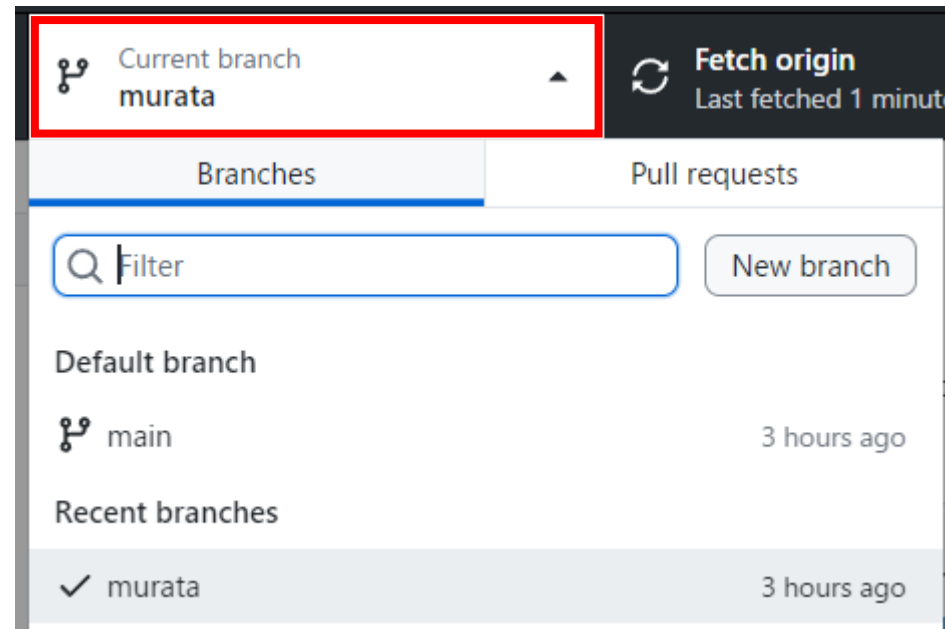
ブランチはローカルリポジトリに作っただけなので
リモートのほうにもPublishして反映させる



GitHubを使ったチーム制作

- ブランチ(Branch)の作り方

Current branchが新しく作ったブランチに切り替わっていることを確認しておく



GitHubを使ったチーム制作

- ブランチ(Branch)での注意事項

ブランチに切り替えた後、ローカルリポジトリの
ファイルを更新した際は、リモートリポジトリの
ブランチへPushする必要がある

例)

ローカル: murataブランチ

↓ Push

リモート: murataブランチ (mainにPushは×)

GitHubを使ったチーム制作

- ブランチ(Branch)の統合(マージ)

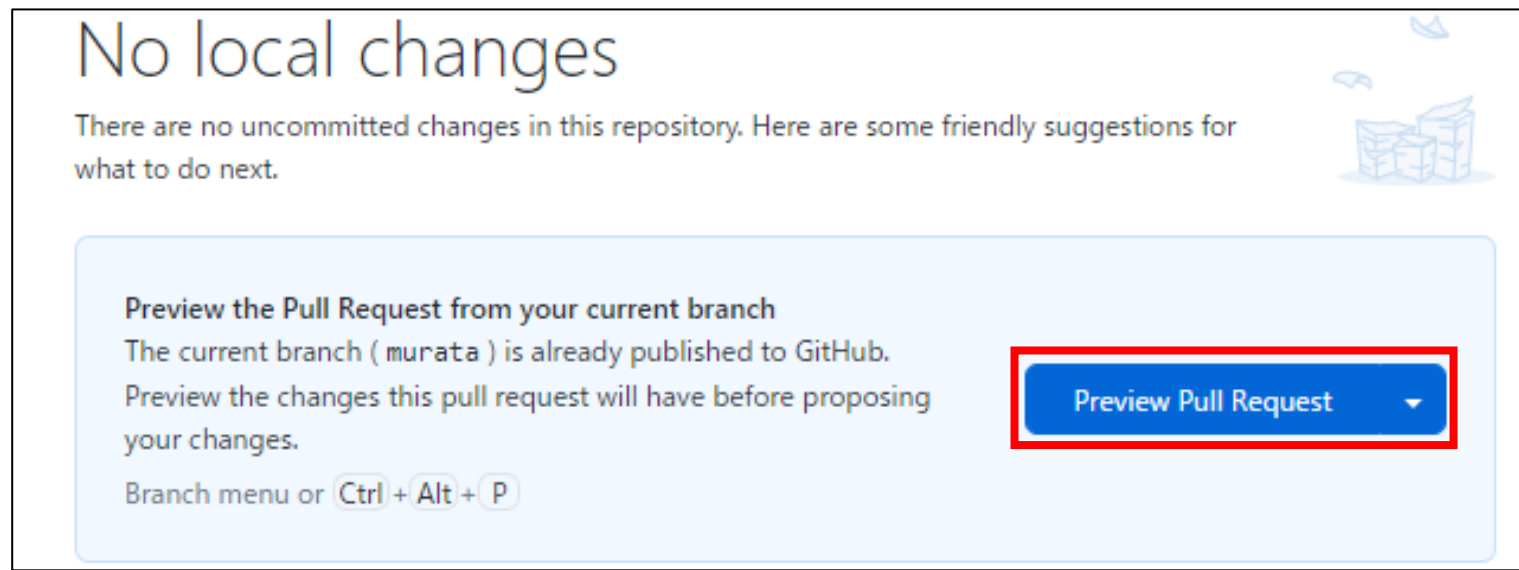
ブランチでの更新作業が終わったら、mainブランチに統合(マージ)する必要がある

マージする際に競合(コンフリクト)が発生したときは、勝手にmainブランチにマージせずに、リーダーもしくはメインプログラマにレビュー(変更結果の確認)を依頼する

GitHubを使ったチーム制作

- 統合(マージ)する方法

[Preview Pull Request] ボタンをクリック

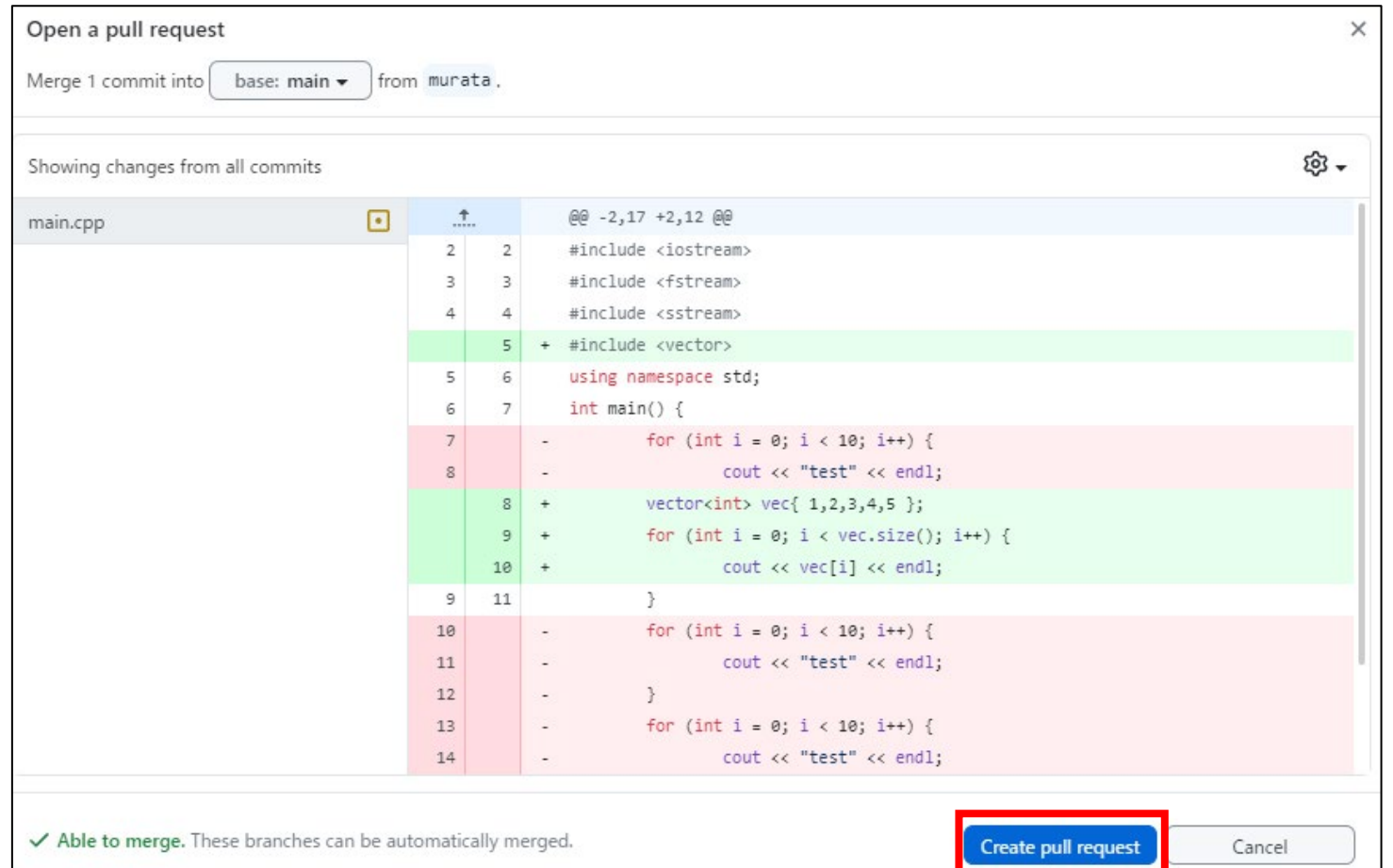


GitHubを使ったチーム制作

- 統合(マージ)する方法

変更したファイル
一覧と変更内容
が表示される

[Create pull
request]を
クリック



GitHubを使ったチーム制作


- 統合(マージ)する方法

変更についてのタイトル
や内容について書いた
あとで[Create pull
request]をクリック

Open a pull request













Create a new pull request by comparing changes across two branches. If you need to, you can also [compare here](#).

base: main ← compare: murata ✓ Able to merge. These branches can be automatically merged.

 Add a title

20241105

Add a description

Write Preview H B I            

変更内容

- vectorコンテナの追加
- vectorの内容を表示

Markdown is supported | Paste, drop, or click to add files

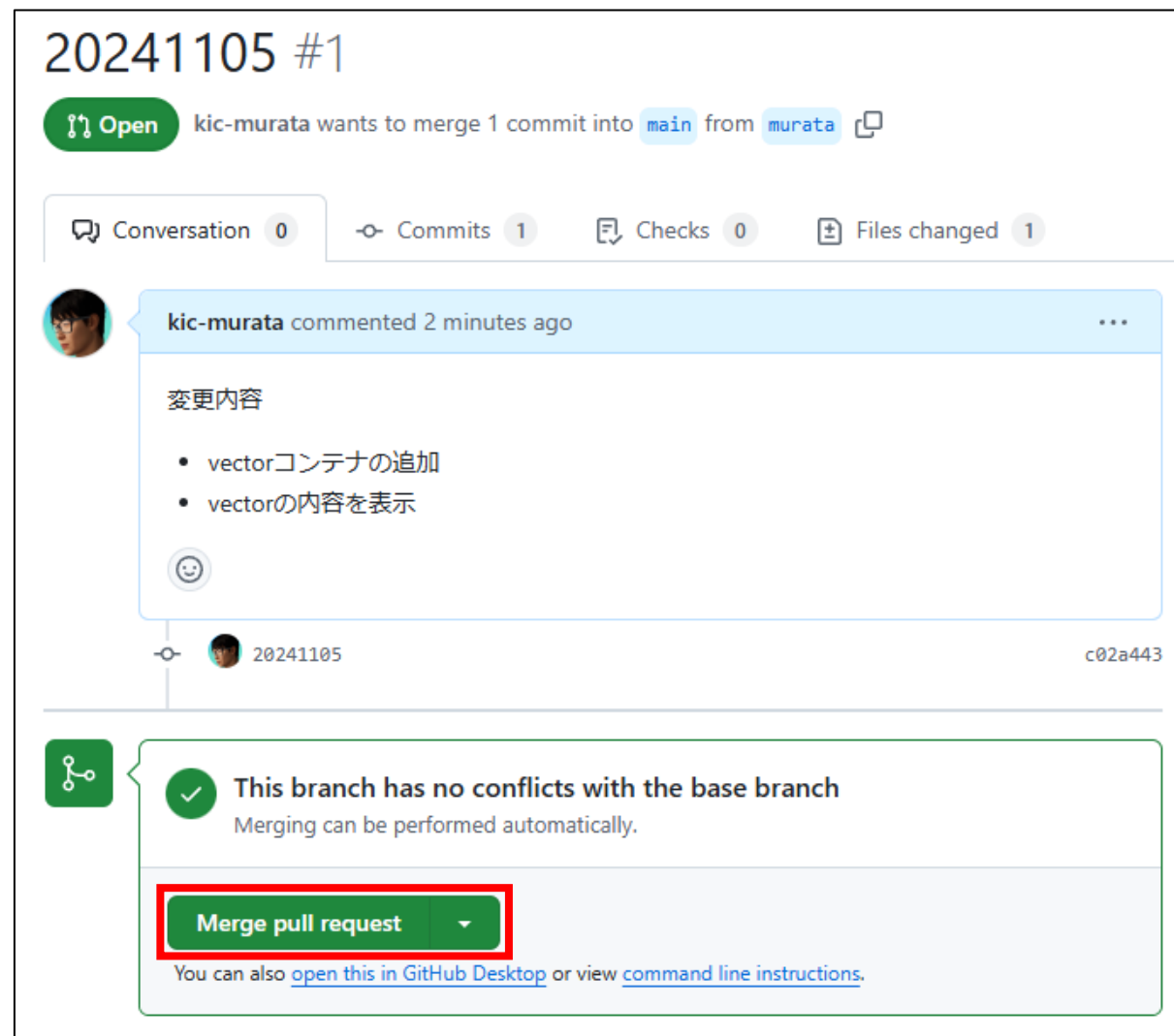
Create pull request

GitHubを使ったチーム制作

• 統合(マージ)する方法

特に問題がなければ右のような画面になる

[Merge pull request]をクリック



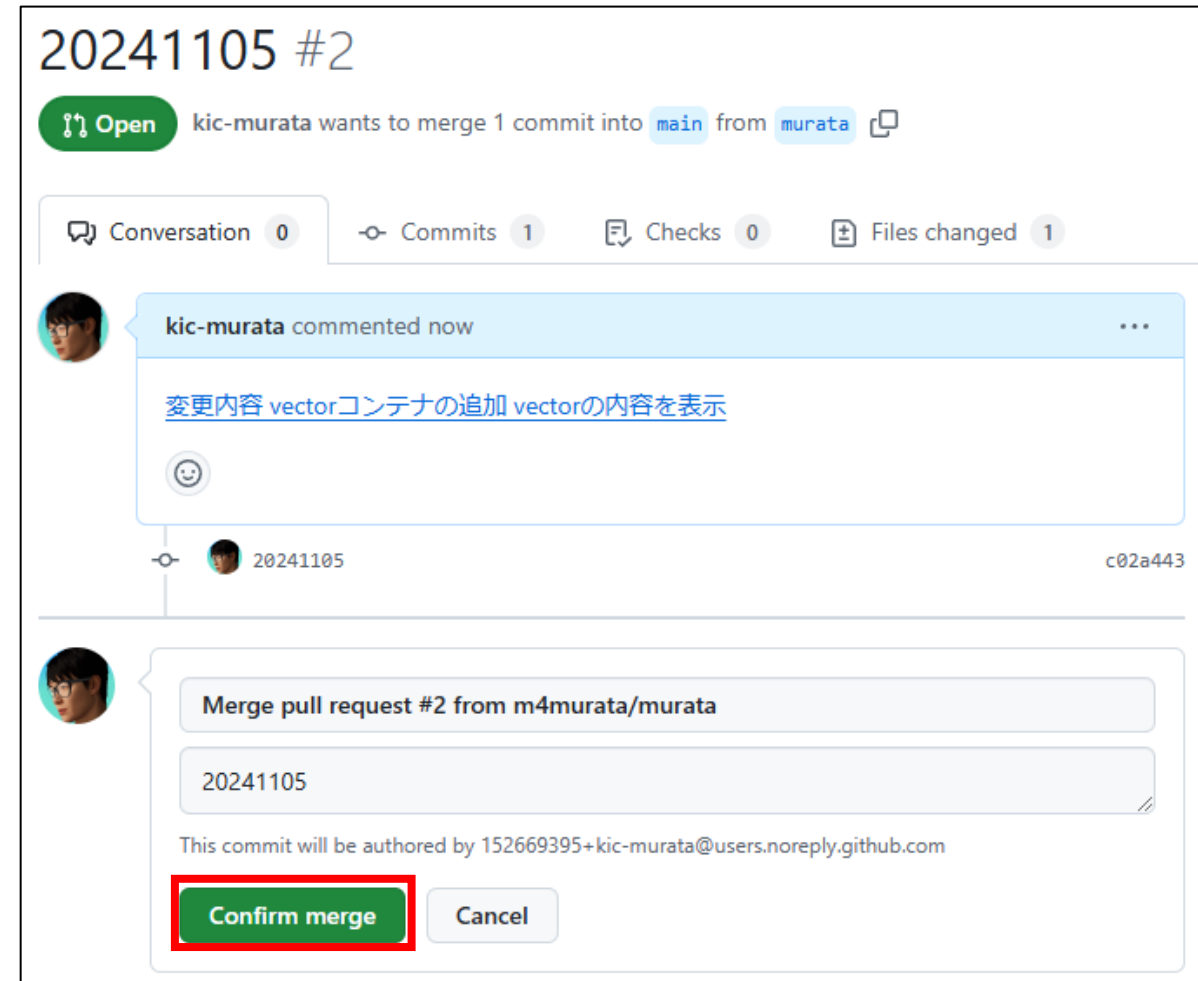
GitHubを使ったチーム制作

• 統合(マージ)する方法

マージするための最終確認が表示されるので問題なければ

[Confirm merge]

をクリック



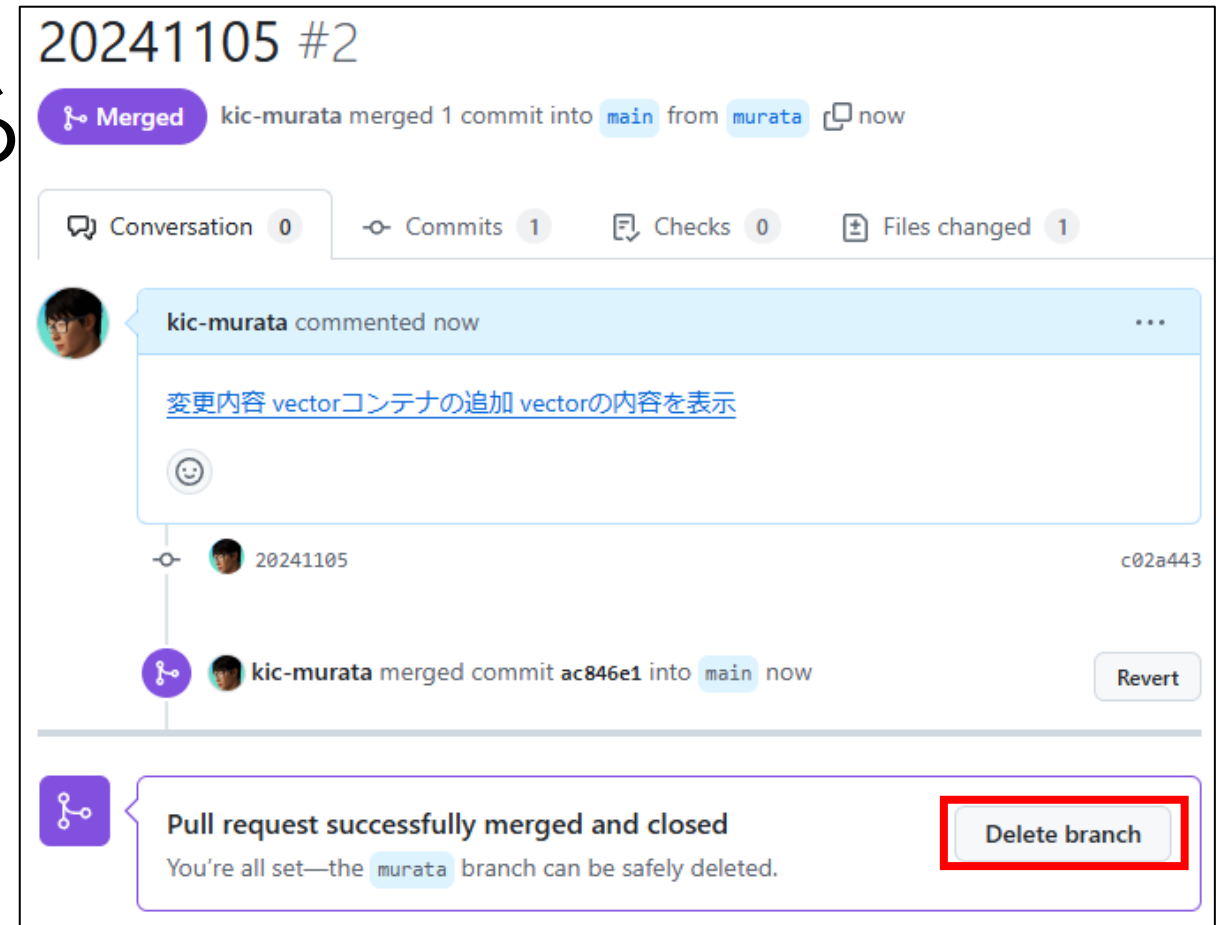
GitHubを使ったチーム制作

・統合(マージ)する方法

Merged と表示されるとマージ完了

マージした後はブランチは不要になるので削除する

[Delete branch]
をクリック

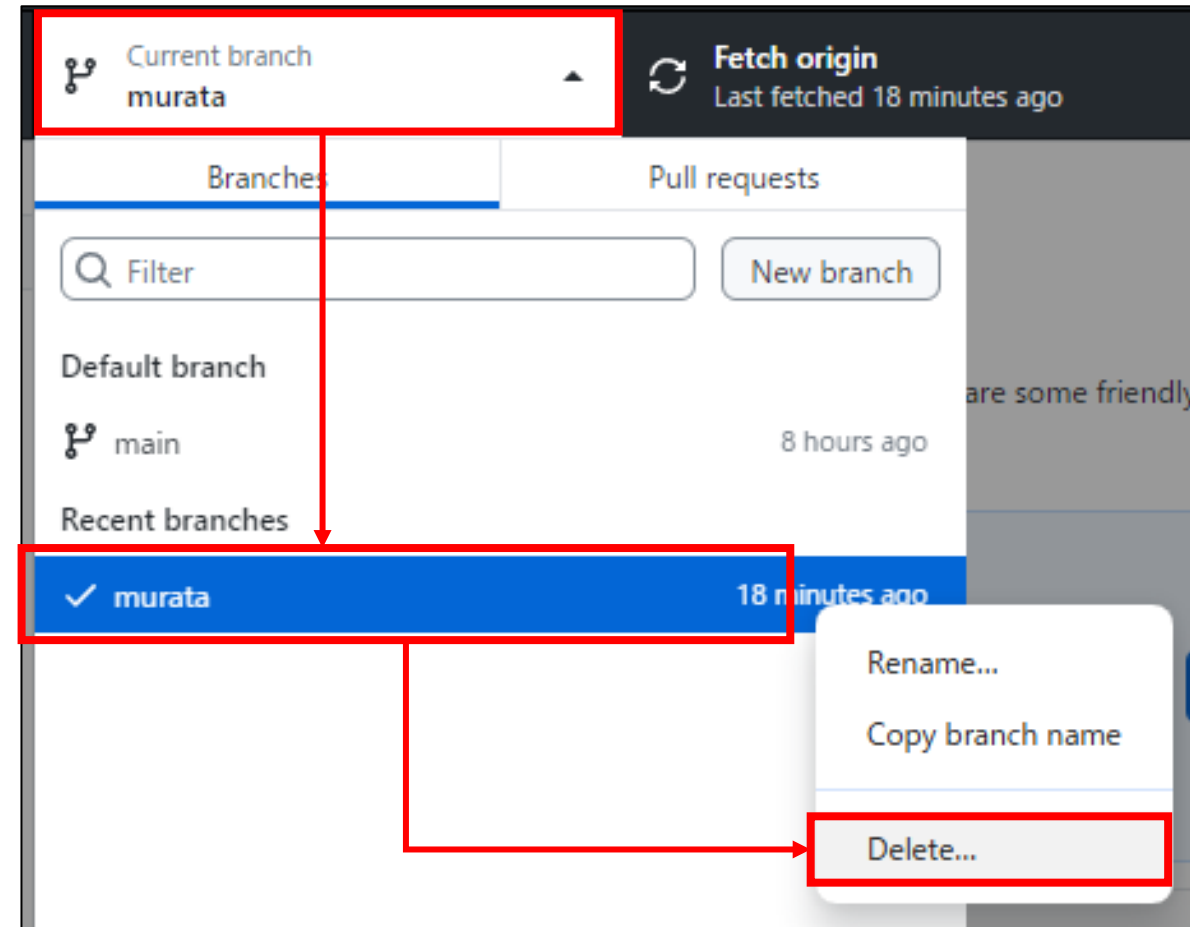


GitHubを使ったチーム制作

- 統合(マージ)する方法

リモートのブランチは削除されたので、ローカルのブランチも削除する

Current branchのメニューからブランチを選択して[Delete]をクリック



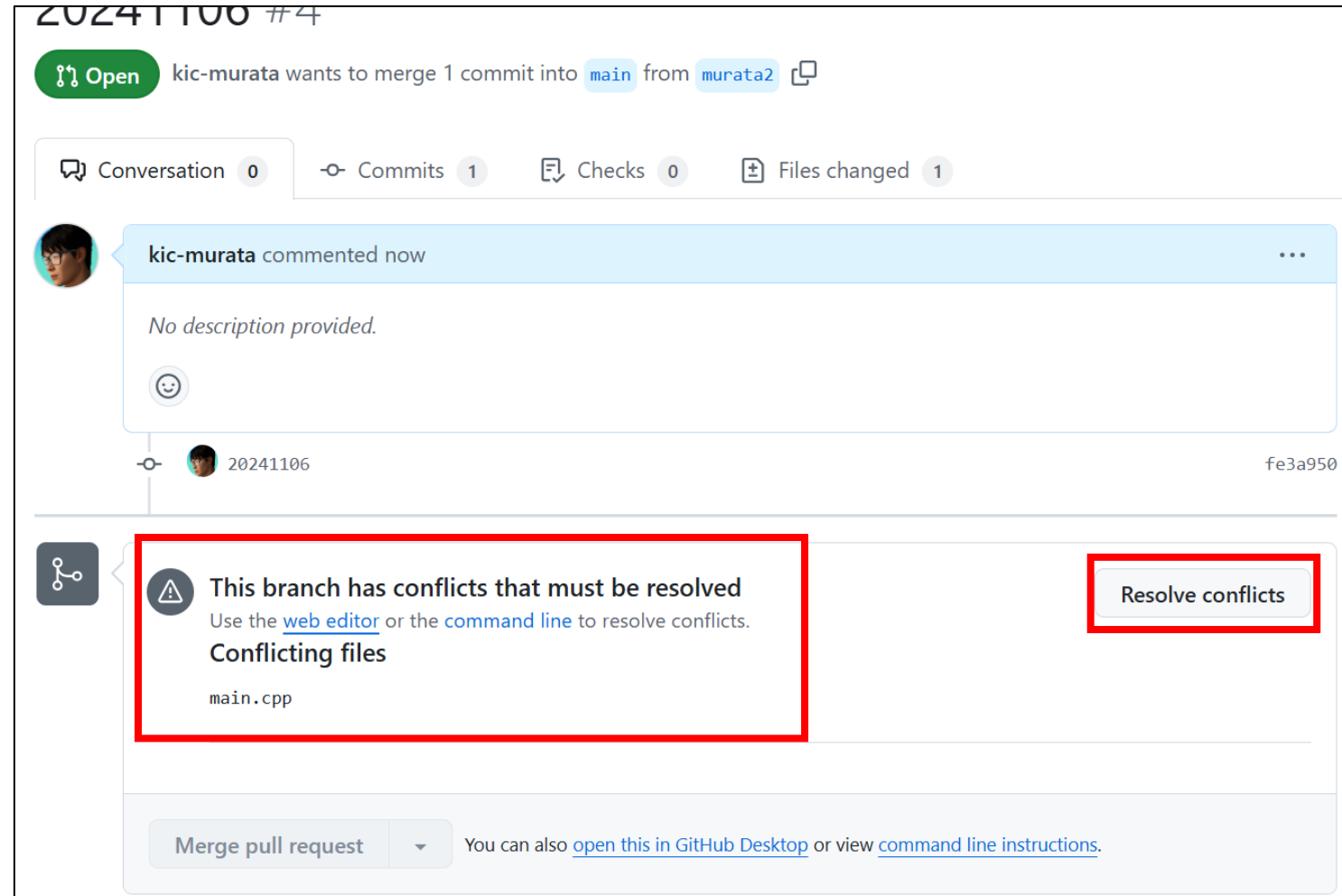
GitHubを使ったチーム制作

- 競合(コンフリクト)が発生したとき

右図のように競合(コンフリクト)が発生するとマージができない

[Resolve conflicts]

から競合を解決する必要がある



GitHubを使ったチーム制作

・競合(コンフリクト)が発生したとき

20241106 #4

Resolving conflicts between `murata2` and `main` and committing changes → `murata2`

1 conflicting file

main.cpp

1 conflict Prev ^ Next v [Settings] Mark as resolved

main.cpp
main.cpp

```
1  #include "chara.h"
2  #include <iostream>
3  #include <fstream>
4  #include <sstream>
5  #include <vector>
6  using namespace std;
7  int main() {
8      <<<<<< murata2
9          vector<int> vec{ 1,2,3,4,5,6,7 };
10         for (int i = 0; i < vec.size(); i++) {
11             cout << vec[i] << endl;
12         }
13         =====
14         // Comment
15         vector<int> vec{ 1,2,3,4,5,6 };
16         for (int i = 0; i < vec.size(); i++) {
17             cout << vec[i] << endl;
18         }
19         map<string, int> tmp{{"ABC",1},{ "DEF",2}};
20     >>>>>> main
21     return 0;
22 }
23
```

競合が発生しているファイル
main.cpp

競合している箇所

GitHubを使ったチーム制作

- 競合(コンフリクト)が発生したとき

```
5    #include <vector>
6    using namespace std;
7    int main() {
8    <<<<<< murata2
9        vector<int> vec{ 1,2,3,4,5,6,7 };
10       for (int i = 0; i < vec.size(); i++) {
11           cout << vec[i] << endl;
12       }
13       =====
14       // Comment
15       vector<int> vec{ 1,2,3,4,5,6 };
16       for (int i = 0; i < vec.size(); i++) {
17           cout << vec[i] << endl;
18       }
19       map<string, int> tmp{{"ABC",1},{ "DEF",2}};
20 >>>>>> main
21     return 0;
```

murata2ブランチで
変更した箇所

どちらの変更を残すか
チーム内で決定して
不要な箇所は削除する

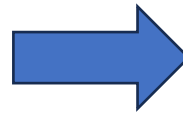
mainブランチで
変更した箇所

GitHubを使ったチーム制作

- **競合(コンフリクト)**が発生したとき

競合を解決して **[Mark as resolved]** をクリック

```
5  #include <vector>
6  using namespace std;
7  int main() {
8  <<<<<< murata2
9      vector<int> vec{ 1,2,3,4,5,6,7 };
10     for (int i = 0; i < vec.size(); i++) {
11         cout << vec[i] << endl;
12     }
13     =====
14     // Comment
15     vector<int> vec{ 1,2,3,4,5,6 };
16     for (int i = 0; i < vec.size(); i++) {
17         cout << vec[i] << endl;
18     }
19     map<string, int> tmp{{"ABC",1},{ "DEF",2}};
20     >>>>>> main
21     return 0;
```



```
5  #include <vector>
6  using namespace std;
7  int main() {
8      // Comment
9      vector<int> vec{ 1,2,3,4,5,6 };
10     for (int i = 0; i < vec.size(); i++) {
11         cout << vec[i] << endl;
12     }
13     map<string, int> tmp{{"ABC",1},{ "DEF",2}};
14     return 0;
```



Mark as resolved

GitHubを使ったチーム制作

- ・競合(コンフリクト)が発生したとき

[Commit merge]をクリック

20241106 #4

Resolving conflicts between `murata2` and `main` and committing changes → `murata2`

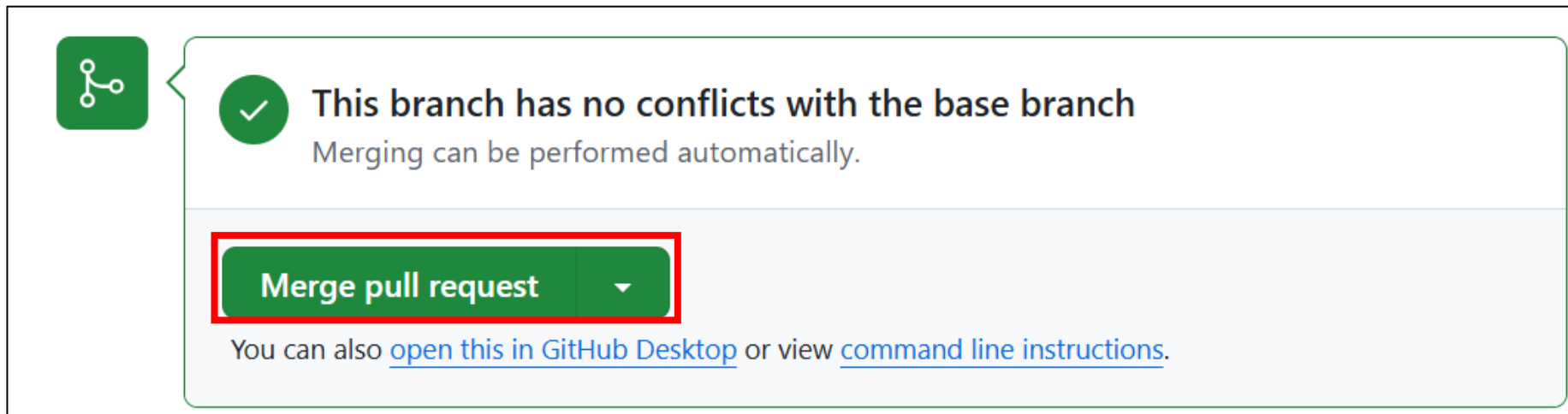
Commit merge

1 conflicting file	main.cpp	✓ Resolved
main.cpp main.cpp	<div>1 <code>#include "chara.h"</code> 2 <code>#include <iostream></code> 3 <code>#include <fstream></code> 4 <code>#include <sstream></code> 5 <code>#include <vector></code></div>	

GitHubを使ったチーム制作

- **競合(コンフリクト)**が発生したとき

あとは競合が発生しないときと同じ手順で
統合(マージ)を行う **[Merge pull request]**
をクリック



GitHubを使ったチーム制作

・開発の流れまとめ

- ① 新しい機能を追加したいときは、まずはローカルブランチを作ってリモートブランチをPublishする
- ② ローカルブランチで作業したのちリモートブランチへPushする（※②の作業を繰り返す）
- ③ 安定動作することが確認できたら、Pull request してmainブランチへマージ
- ④ リモートブランチをサイト上で削除
- ⑤ ローカルブランチをGithubDesktopを使って削除

GitHubを使ったチーム制作

- GitHub Desktop以外のアプリ

GitHubを使うためのアプリは他にも存在する

- ① SourceTree
- ② Fork
- ③ TortoiseGit

全ブランチの可視化やマージがしやすくなっていたりするので、気になる人は調べてみてください