



Web × IoT メイカーズチャレンジ 2019-2020

GitHub ハンズオンセミナー

GitHub の使い方

この文章に含まれる情報は、公表の日付の時点での Web × IoT メイカーズチャレンジ 運営委員会の考え方を表しています。市場の変化に応える必要があるため、Web × IoT メイカーズチャレンジ 運営委員会は記載されている内容を約束しているわけではありません。この文書の内容はリリース後も正しいとは保障できません。この文章は情報の提供のみを目的としています。

GitHub は GitHub Inc. の日本およびその他の国における登録商標です。

本文中に記載されているブランド名、会社名、製品名等は、それぞれ各社の登録商標または商標です。

この文章内での引用（図版やロゴ、文章など）は、GitHub Inc. と XXXXX からの許諾を受けています。

Copyright © 2020 Copyright © 2020 Web×IoT メイカーズチャレンジ実行委員会

目次

STEP 1.	GitHub って？	4
1.1	GitHub って？	5
1.2	まず知っておきたいこと	6
STEP 2.	GitHub をつかってみよう	7
2.1	GitHub を使うための準備	8
2.2	ドキュメントの追加	10
STEP 3.	コードレビューをしてみよう	15
3.1	Pull Request (プル リクエスト)	16
3.2	コードレビューしてみよう	20
3.3	コードレビューの内容を反映させる	24
3.4	Pull Request の Merge (マージ)	27
STEP 4.	ローカルの PC 上で開発する	29
4.1	Git のセットアップ	30
4.2	コミット メールアドレスの設定	31
4.3	ローカルの PC に GitHub の Repository を複製する	32
4.4	Local Repository で作業 (開発) する	34
4.5	Remote Repository (GitHub) へ反映する	37
STEP 5.	その他 おぼえておきたいこと	39
5.1	ローカルの PC 上で別の Branch で作業をする	40
5.2	その他	43

STEP 1. GitHub って？

GitHub について簡単に紹介します。

1.1 GitHub って？

➡ 開発者のためのプラットフォーム

GitHub は、ユーザーのみなさんからヒントを得て作成された開発プラットフォームです。[オープンソースプロジェクト](#)や[ビジネスユース](#)まで、GitHub 上にソースコードをホスティングすることで数百万人もの他の開発者と一緒にコードのレビューを行い、プロジェクトの管理をしながら、ソフトウェアの開発を行うことができます。

➡ GitHub はどうしてよく使われる？

GitHub は、個人で開発したものを公開したり、公開したものについて意見をもらったり、仲間と一緒に開発する時に、ファイルの重複や編集時のすれ違いを管理してくれる優れたツールです。

また、クラウド上で簡単にセキュアに複数人と成果物を共有することができ、さまざまなツールと連携していることから、世界中で使われていることも人気の理由です。

➡ GitHub Flow について

GitHub Flow は GitHub 社が推奨するソフトウェア開発フローです。

<https://guides.github.com/introduction/flow/>

今回のハンズオンでは、GitHub Flow を参考にしながら、GitHub 上で共有されている Chirimen with micro:bit の Examples を個人で利用する方法から体験していただきます。



1.2 まず知っておきたいこと

➡ Repository (リポジトリ)

公開したり、管理したいソースコードやファイルを格納したりする場所が Repository (リポジトリ) です。

主に、保管される場所によって、ローカル (自分の PC) と リモート (共有されている) に区別されています。



STEP 2. GitHub をつかってみよう

ブラウザー上で GitHub を使って、基本的な操作を体験しましょう。

2.1 GitHub を使うための準備

➡ GitHub にサインインする

GitHub は無料でも利用できます。

1. こちらのサイトから GitHub を使うためのアカウントを手順に沿って作成します。

アカウントの登録にはその場で確認できるメールアドレスが必要です。

<https://github.com/join>

2. 次に、作成したアカウントで、サインインします。

<https://github.com/login>

➡ Fork (フォーク) する

まずは、[Web x IoT メイカーズチャレンジの GitHub に置いてあるハンズオン用の Remote Repository](#) (リモート リポジトリ) から、自分の Repository (リポジトリ) に Fork します。

Fork することで、他の人の Remote Repository を自分の Repository に複製することができ、自分で変更したり、ファイルを追加したりできます。

● フォーク
fork

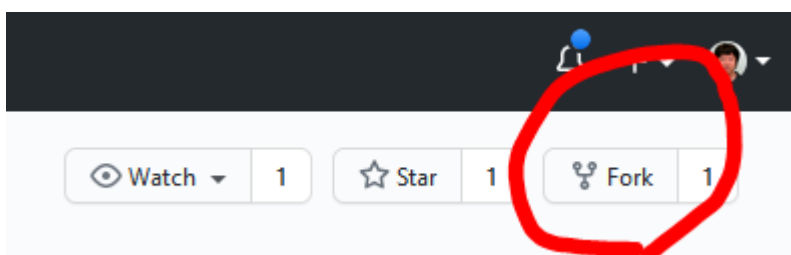


他の人のリモートリポジトリを自分のリモートリポジトリに複製
(オリジナルへの貢献が前提)

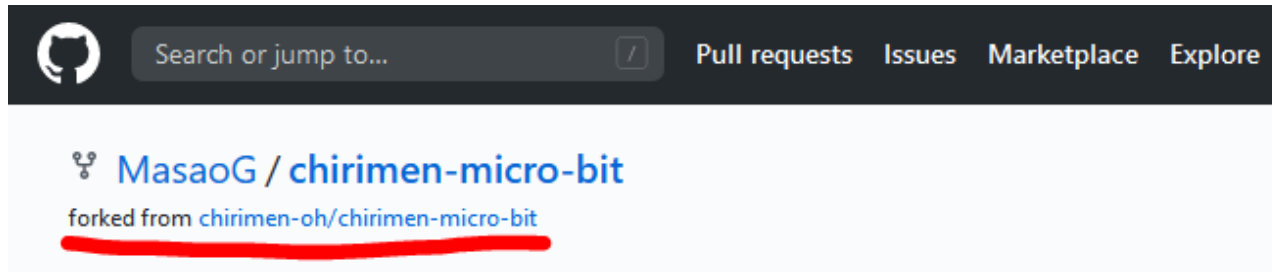
1. Chirimen with micro:bit の Example の Remote Repository にアクセスします。

<https://github.com/webiotmakers/GitHubExample>

2. 画面右上にある [Fork] ボタンをクリックします。



3. 少し時間がかかりますが、すぐに自分のアカウントに、chirimen-micro-bit の Remote Repository の内容が複製されます。



4. これで準備は完了しました。

Repository 名

MasaoG / chirimen-micro-bit

forked from chirimen-oh/chirimen-micro-bit

Watch 0 Star 0 Fork 1

Code Pull requests Actions Projects Wiki Security Insights Settings

master

This branch is even with chirimen-oh:master.

Pull request Compare

このリポジトリで管理されているファイル一覧

File/Folder	Commit Message	Time Ago
examples	Merge branch 'master' of github.com:chirimen-oh...	24 days ago
fritzing	Add GPIO Example 3	26 days ago
guidebooks	Update startup.md	14 days ago
imgs	Add GPIO Example 3	26 days ago
micro-bit	update latest makecode publish link	14 days ago
polyfill	fix: ReferenceError	last month
tests	リンク切れを修正	last month
LICENSE	Update LICENSE	12 months ago
README.md	rm web page section (I've set website of repo on ...)	12 months ago
_config.yml	Set theme jekyll-theme-cayman	13 months ago

このリポジトリの概要

Language: Japanese, English (Google Translation)

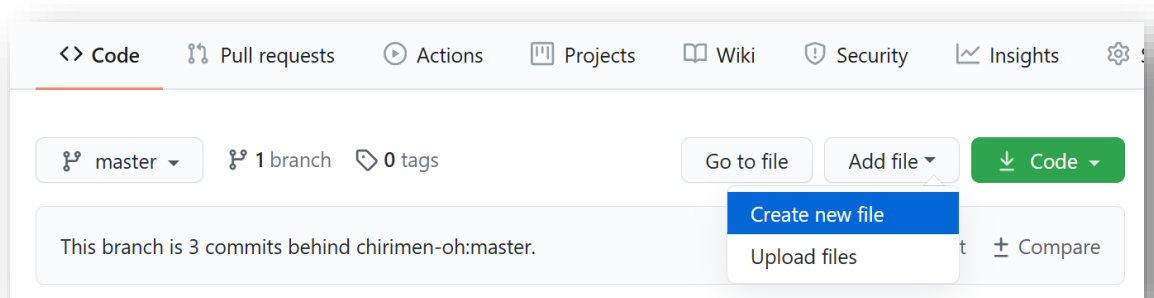
JavaScript 65.7%
HTML 17.6%
TypeScript 16.7%

2.2 ドキュメントの追加

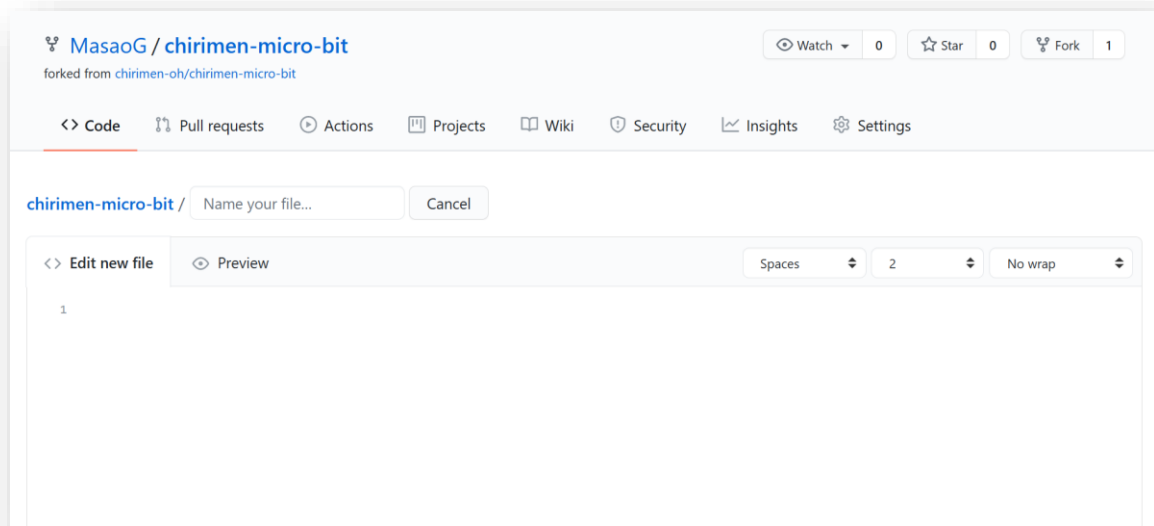
早速ドキュメントを追加してみましょう。実際の開発プロジェクトでは、これがソースコードの追加変更に相当します。

今回はブラウザー上でファイルの変更をおこないます。通常の開発時であれば、開発者のマシン上でエディターや IDE (統合開発環境) を使ってソースコードを変更すると思いますが、まずはブラウザー上で変更を行います。

1. ドキュメントの追加を行うには、下の図にある [Add file] ボタンをクリックし、プルダウンで出てきたメニューから “Create new file” をクリックします。



2. 下図のようにファイルの編集画面が表示されます。この編集画面上で、今回は皆さんの自己紹介を書いてみましょう。



3. まずは画面上部の “Name your file…” と書かれた箇所にファイルの名前を入力します。今回は self-introduction というディレクトリの下にファイルを作ることとします。その場合には、この入力欄にまず “self-introduction/” と “/” (スラッシュ) まで入力してみてください。そうすると、“/” をディレクトリ名の区切りとして扱うことができます。



4. ファイル名は「(GitHub のアカウント名).md」としましょう。“md”という拡張子は Markdown (マークダウン) というファイルで使われます。Markdown というのは、シンプルな記法でリッチテキストを記述するための記法です。GitHub では様々なテキスト入力エリアでこの Markdown を使うことができます。

Markdown の記法については、以下のドキュメントを参考にしてください。

<https://guides.github.com/features/mastering-markdown/>

3. ファイル名を入力したら、下の段の “Edit new file” タブの入力エリアに、以下のような内容を入力しましょう。

```
## 自己紹介
- 名前: (みなさんのお名前)
- 所属: (学校名、会社名)
- アカウント名: (GitHub アカウント名)

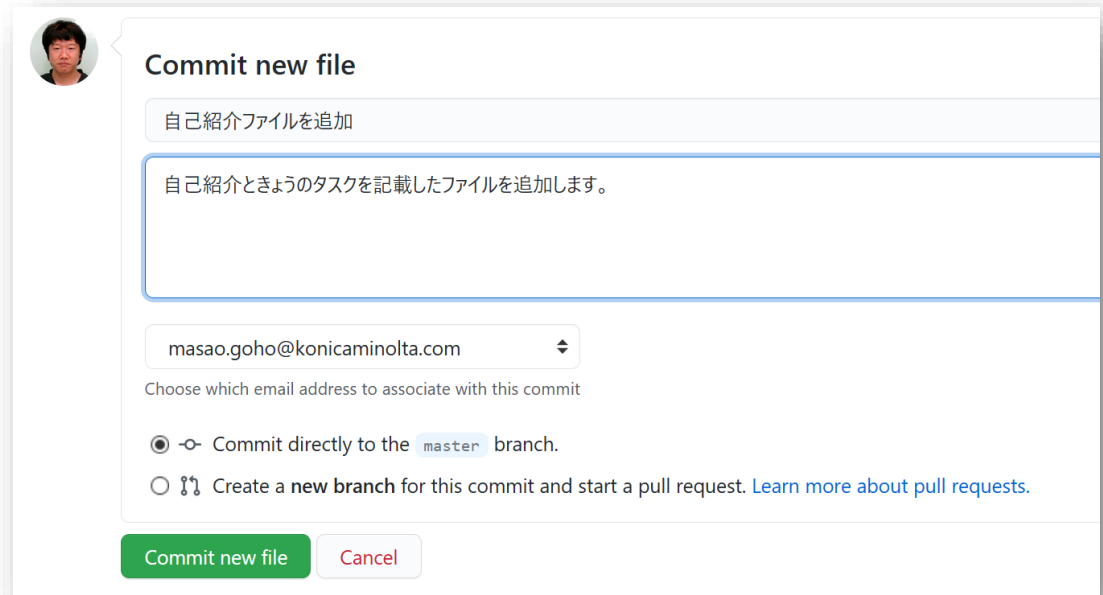
## きょうのタスク
- [ ] GitHub アカウントを作成する
- [ ] ファイルを追加、編集する
- [ ] コードレビューを行う
- [ ] 変更をマージする
```

(記号の間は半角スペースを開けてください。)

4. Web 上のエディターに上記の内容を追記した状態で、下図の赤枠で囲まれている “Preview” タブをクリックしてみましょう。すると、次のように変更後のファイルがどのように表示されるかプレビューを見ることができます。



5. 表示に問題なければ、次にこの変更を Commit (コミット) します。Commit するには、編集画面の下の方をご覧くださいと、次のように Commit を行うメニューを見つけることができます。



Commit new file

自己紹介ファイルを追加

自己紹介とさようのタスクを記載したファイルを追加します。

masao.goho@konicaminolta.com

Choose which email address to associate with this commit

☒ Commit directly to the `master` branch.

☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit new file Cancel

ここで、1 行テキストボックスに今回行った変更の概要を入力し、その下の複数行のテキストボックスには変更の詳細を入力します。

そのさらに下の 2 つの選択肢を持つラジオボタンがあります。ここは重要なポイントです。

このラジオボタンは、今行った変更をどの Branch (ブランチ) に Commit するかを選択する箇所です。

- ☒ Commit directly to the `master` branch.
- ☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

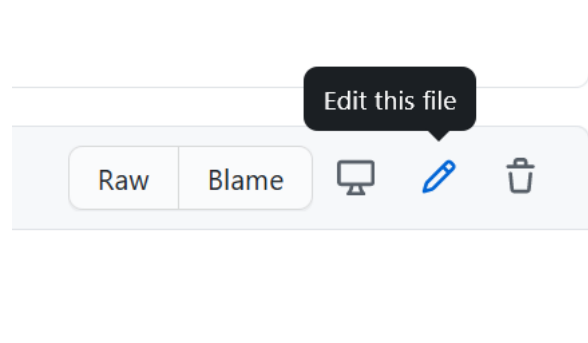
1 つ目の選択肢は `master` に直接 Commit する、2 つ目の選択肢はこの場で新しい Branch を作成し、その新しい Branch に Commit するという選択肢です。

ここではひとまず直接ファイルを Commit しますので (自分の Repository なので)、既定値のままにして “Commit new file” ボタンをクリックします。作成したファイルが表示されますので “(GitHub のアカウント名).md” ファイルをクリックしてください。

すると、次のような画面が表示されます。



引き続き、このファイルを編集する時は、画面右上にあるペンのアイコンをクリックすると、編集画面になります。



STEP 3. コードレビューをしてみよう

ブラウザ上で GitHub を使って、ファイルの変更内容をレビューしてみよう。

通常であれば、チーム内の同僚や仲間に自分が変更した内容についてレビューするリクエストを出して、レビューしてもらうという形になりますが、今回は、ご自身でコードレビューをしていただこうと思います。

ですので、ここからはコードレビューを依頼された立場になりきって手順を進めてみてください。

3.1 Pull Request (プル リクエスト)

先ほど作成した自己紹介ファイルを使って、Pull Request の作成に取り組みます。

Repository には通常 master と呼ばれる主流の Branch があり、ここでファイルの追加・編集・削除などの履歴を管理しています。先ほどまでの作業は、自分の Repository の master Branch に直接ファイルを作成・編集・追加を行ったことになります。

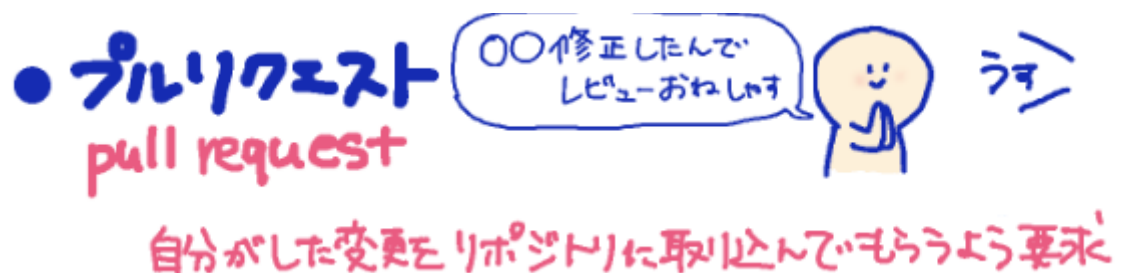


Pull Request は、コードを変更する際に直接変更を加えるのではなく、主流とは別の Branch を一旦作成し、その別の Branch 上で変更を加えてレビューして問題ないかどうかを確認してから、master Branch に統合 (= Merge (マージ)) するというフローを実現します。

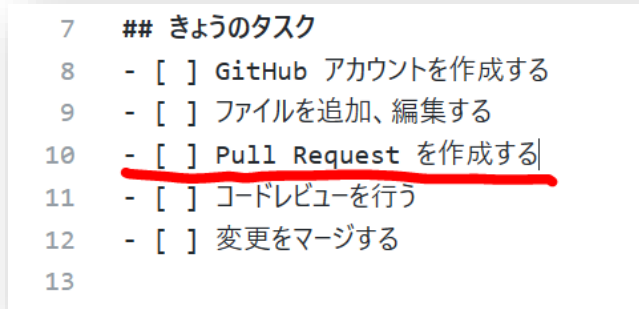


GitHub では、共同で開発する時に複数の Branch を持つことができるので master Branch に影響を与えずに開発作業を並行して進めていくことができる仕組みがあります。

この仕組みを Pull Request を作成するまでの手順を追って体験します。



1. さきほどの自己紹介ファイルを表示し、ペンアイコンをクリックして編集画面を開きます。
2. 編集画面で、##きょうのタスクのところに、“- [] Pull Request を作成する” という一文を追加してみましょう。



```
7  ## きょうのタスク
8  - [ ] GitHub アカウントを作成する
9  - [ ] ファイルを追加、編集する
10 - [ ] Pull Request を作成する
11 - [ ] コードレビューを行う
12 - [ ] 変更をマージする
13
```

3. “Preview changes” タブをクリックしてみると、変更点がハイライトされていることが分かります。この内容で問題なければ、次にこの変更を Commit するので、画面の下に移動します。
4. 1 行テキストボックスには、今回行った変更の概要を入力し、その下の複数行テキストボックスには変更の詳細を入力します。
5. 次に 2 つの選択肢のラジオボタンがありますが、2 つ目の選択肢を選びます。


今回は、新しい Branch 上に Commit して、レビューしてもらってから、この変更内容をマージしていきます。

“Create a **new branch** for this commit and start a pull request…” のラジオボタンを選択すると、自動的に新しい Branch 名が入力されます。

確認できたら緑の “Propose changes” ボタンを押すと、Branch の作成、Commit が実行され、さらに Pull Request の作成画面が表示されます。

きょうのタスク

- ☐ GitHub アカウントを作成する
- ☐ ファイルを追加、編集する
- ☒ Pull Request を作成する
- ☐ コードレビューを行う
- ☐ 変更をマージする



Commit changes

きょうのタスクに1行追記

きょうのタスクに "Pull Request" のタスクを追記。

masao.goho@konicaminolta.com

Choose which email address to associate with this commit

☐ Commit directly to the `master` branch.

☒ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

MasaoG-patch-1

Propose changes

Cancel

6. ここまででファイルの変更が完了し、続いて Pull Request を作成する画面が表示されます。

Pull Request とは、Branch 上で行っている変更について議論することです。

7. 下図のように Pull Request のタイトル、詳細情報を入力する欄が表示され、その下にはこれから Merge しようとしている変更の差分が表示されます。ここで文頭に [+] 文字がついて緑色にハイライトされている部分が追加されたことを意味しています。行を削除した場合は文頭に [-] 文字が付いて赤色にハイライトされます。まずはこの差分表示で変更内容が正しいかどうかを確認しましょう。

Open a pull request

The change you just made was written to a new branch named `MasaoG-patch-1`. Create a pull request below to propose these changes.

base: master ← compare: MasaoG-patch-1 ✓ Able to merge. These branches can be automatically merged.

きょうのタスクに1行追記

Write Preview H B I ≡ < > @ ↺ ↻

きょうのタスクに "Pull Request" のタスクを追記しました。
レビューをお願いします！

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Remember, contributions to this repository should follow our [GitHub Community Guidelines](#).

Reviews
No reviews

Assignees
No one—assign yourself

Labels
None yet

Projects
None yet

Milestone
No milestone

Linked issues
Use [Closing keywords](#) in the description to automatically close issues

Helpful resources
[GitHub Community Guidelines](#)

1 commit ± 1 file changed 0 comments 1 contributor

Commits on Aug 03, 2020

MasaoG きょうのタスクに1行追記

Showing 1 changed file with 1 addition and 0 deletions.

self-introduction/MasaoG.md

```

7 7 @@ -7,5 +7,6 @@
8 8  ## きょうのタスク
9 9  - [ ] GitHub アカウントを作成する
10 10  - [ ] ファイルを追加、編集する
11 11  + [ ] Pull Request を作成する
12 12  - [ ] コードレビューを行う
13 13  - [ ] 変更をマージする

```

No commit comments for this range

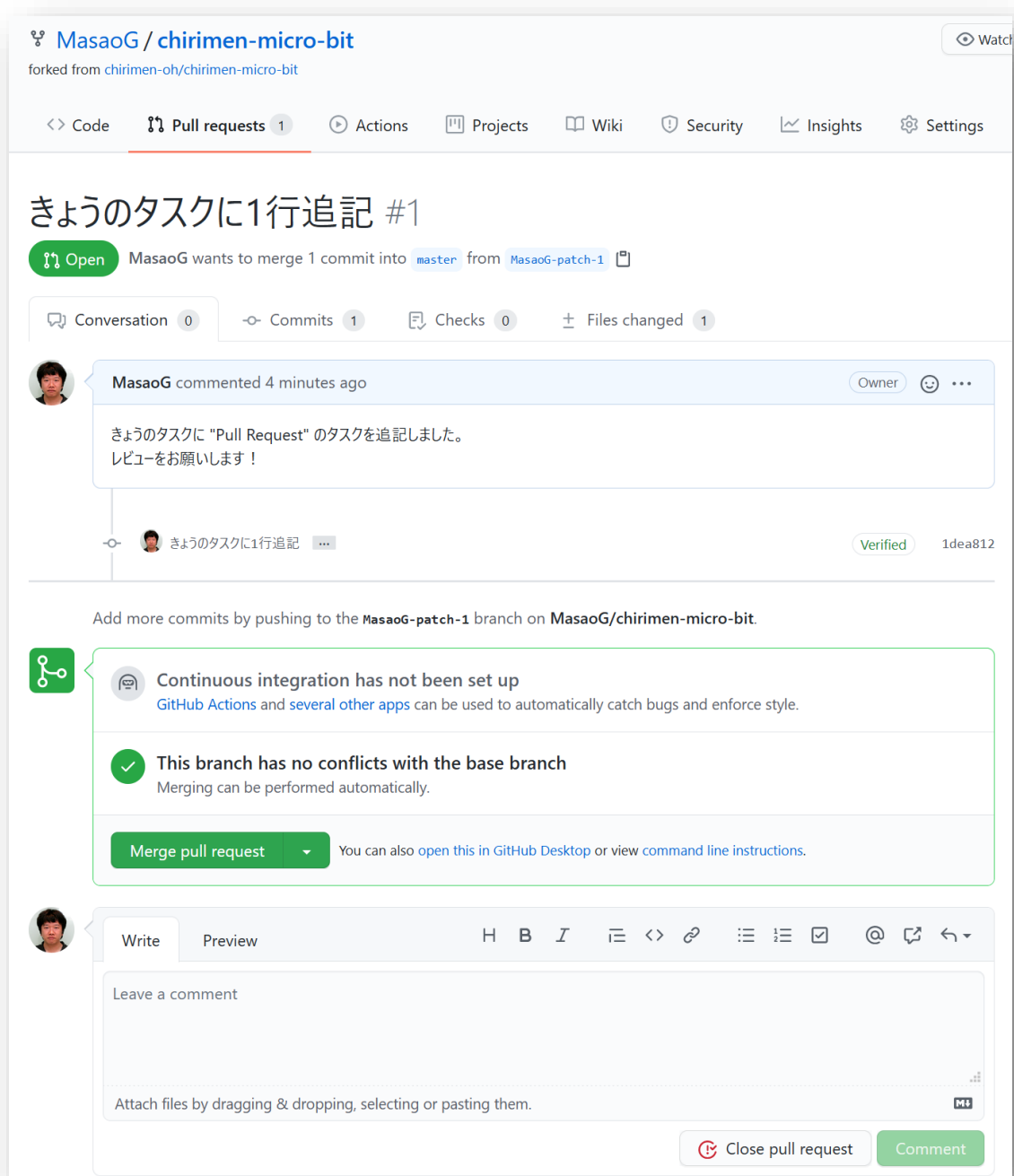
8. 必要に応じて、ページ上部のタイトルや詳細を入力する箇所の内容を修正してみましょう。詳細を入力する箇所では Markdown 記法を使うこともできます。また、ここではコードレビューを依頼したい人を @メンションして通知を送ることもできます。

すべての内容を入力し終わったら、緑色の “Create pull request” ボタンを押しましょう。これで Pull Request の作成が完了し、作成された Pull Request が表示されます。

3.2 コードレビューしてみよう

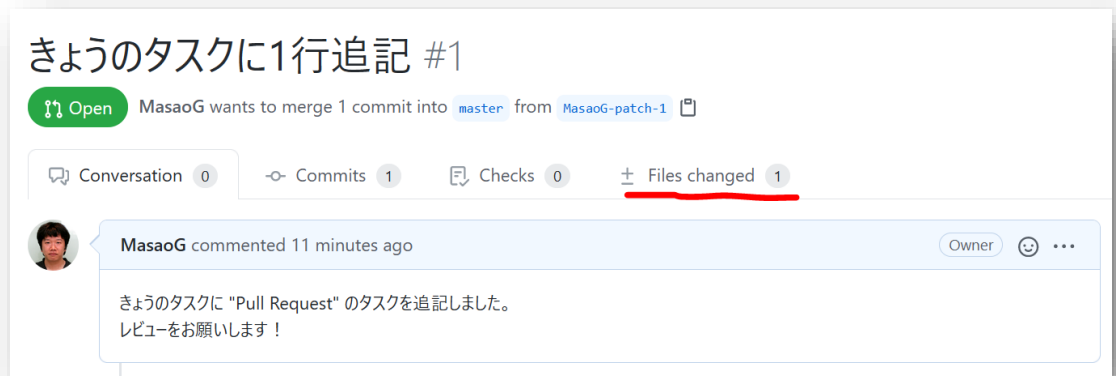
Pull Request の作成が完了しました。次のステップでやるべきことは、この変更内容で問題ないかどうかをレビューすることです。通常であれば、チーム内の仲間に先ほど作成した Pull Request にアクセスしてもらい、内容をレビューしてもらうという形になりますが、ここではご自身でコードレビューをしていただこうと思います。ですので、ここからはコードレビューを依頼された立場になりきって手順を進めてみてください。

コードレビューは先ほど作成した Pull Request 上で行います。前の手順を完了した時点で下図のような Pull Request の画面が表示されているはずです。

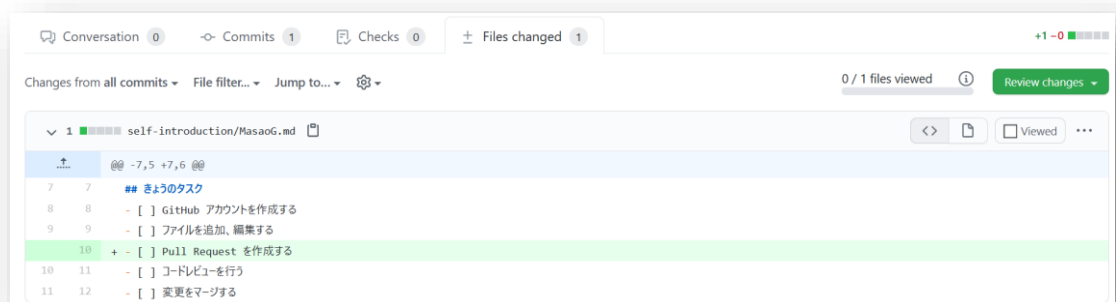


この画面は、この Pull Request で実行しようとしているタスクについて議論する場となっており、Conversation と呼びます。この画面の下部にはコメント入力フォームがありますので、そこからコメントを投稿することができます。

1. 実際に変更の内容をレビューしてみましょう。レビューを行うには、下図のような Pull Request の画面の “Files changed” と書かれたタブをクリックします。
(このタブの右に書かれている数字は変更されたファイルの数を表しています。)



2. すると、次のようにこの Pull Request で行っている変更の内容が表示されます。



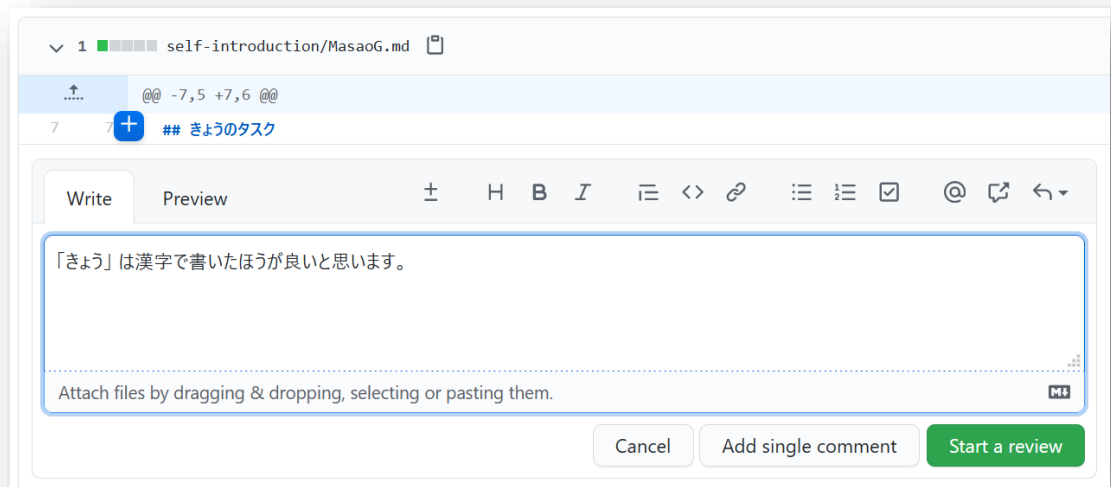
さらに、この変更内容が表示されている部分には好きな行にコメントを書くことができます。コメントを書きたい行にマウスカーソルを合わせてみてください。すると下図のように [+] アイコンが表示されます。



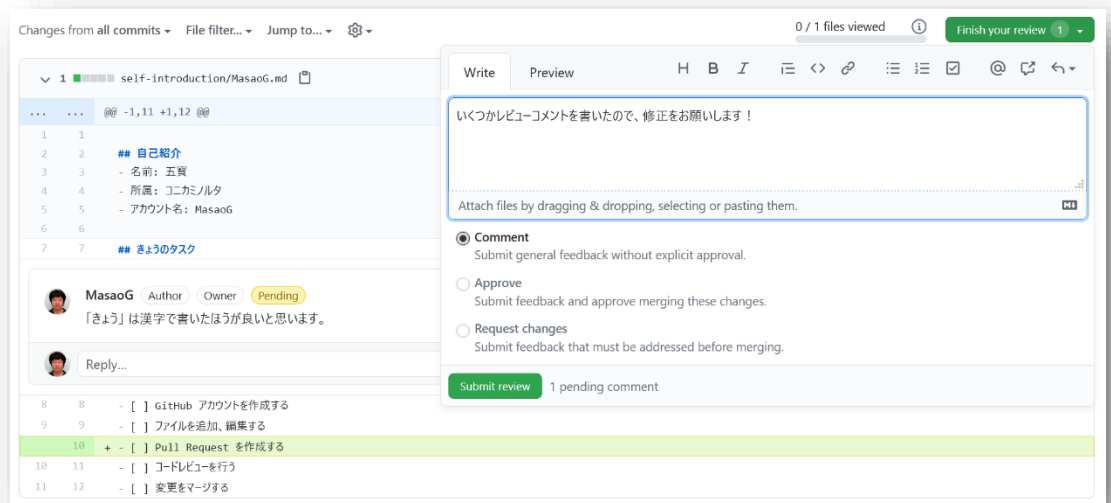
この青い [+] アイコンをクリックすると、コメントを入力するフォームが表示されます。

3. コメントを表示するフォームに、コメントを入れてみましょう。

(例) 「きょう」は漢字で書いたほうが良いと思います。



4. コメントを入力したら “Start a review” と書かれた緑のボタンを押します。すると、レビューがスタートします。通常レビューコメントは一つとは限らず、他の箇所にもコメントを書くケースがほとんどです。他の行にもレビューコメントをいくつか書いてみましょう。
5. すべてのレビューコメントが書き終わったら、右上にある緑の “Finish your review” ボタンを押します。



すると、レビュー全体についてのコメントを入力することができます。今回はレビューの結果、変更内容の修正が必要だったとします。そこで、Pull Request を作成した人（今回は自分ですが）に対して、修正が必要な旨のコメントを書いて “Submit review” ボタンを押しましょう

6. レビューコメントがすべて投稿されて、下図のように Pull Request の Conversation 画面でもレビューコメントを見ることができるようになります。

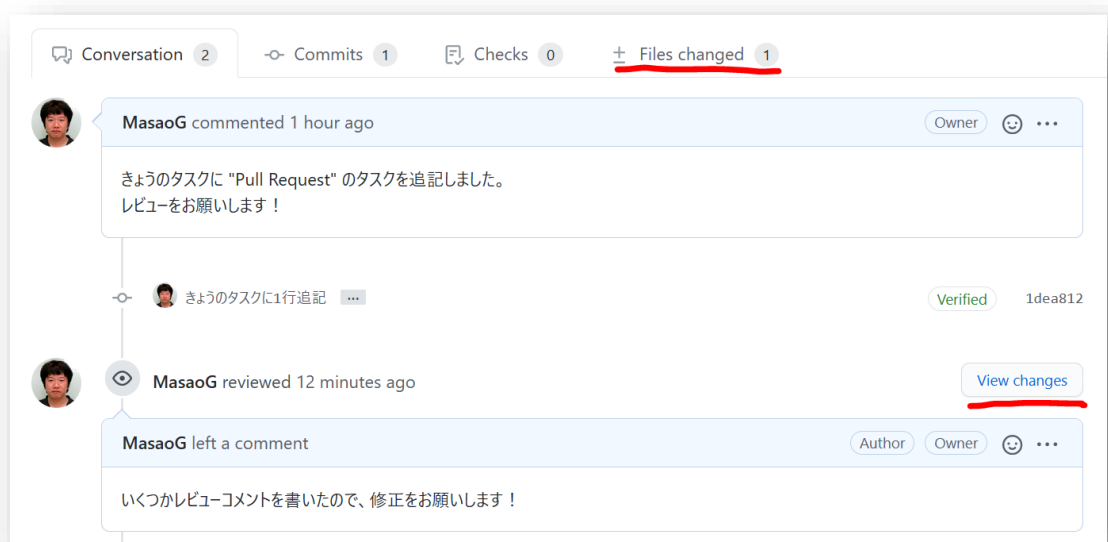


このように Conversation 画面ではレビューコメントやそれ以外のコメントも含めて、すべての議論のやり取りを上から下に時系列に従って確認することができるようになっています。これでコードレビューは完了です。

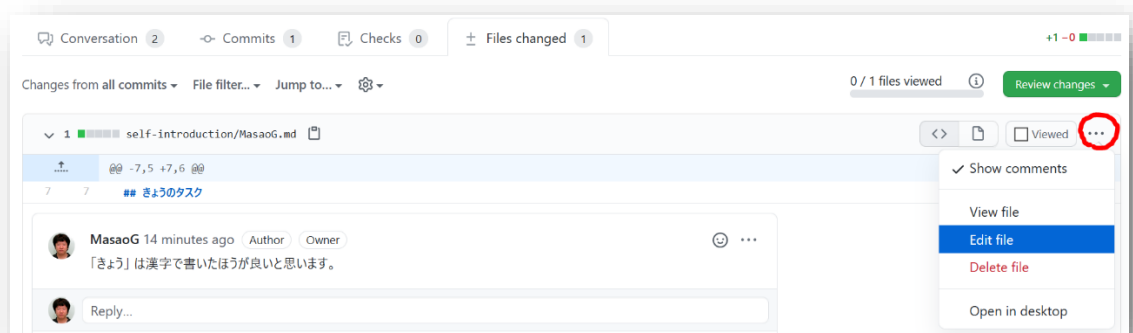
3.3 コードレビューの内容を反映させる

ここで、立場を Pull Request を作成した側に戻します。チームメンバーにコードレビューをもらった結果、修正が必要な箇所が明らかになりました。ここからはその修正を行います。

1. (GitHub アカウント名).md ファイルの内容をさらに修正するには、再度 “Files changed” タブまたは、レビューコメントの右上にある “View changes” ボタンを押し、変更内容の画面を表示してください。



2. つぎに下図の右上にある「・・・」をクリックし、プルダウンメニューから “Edit file” をクリックします。



3. (GitHub アカウント名).md ファイルの修正画面が表示されますので、ここでレビューコメントを受けた修正を行います。

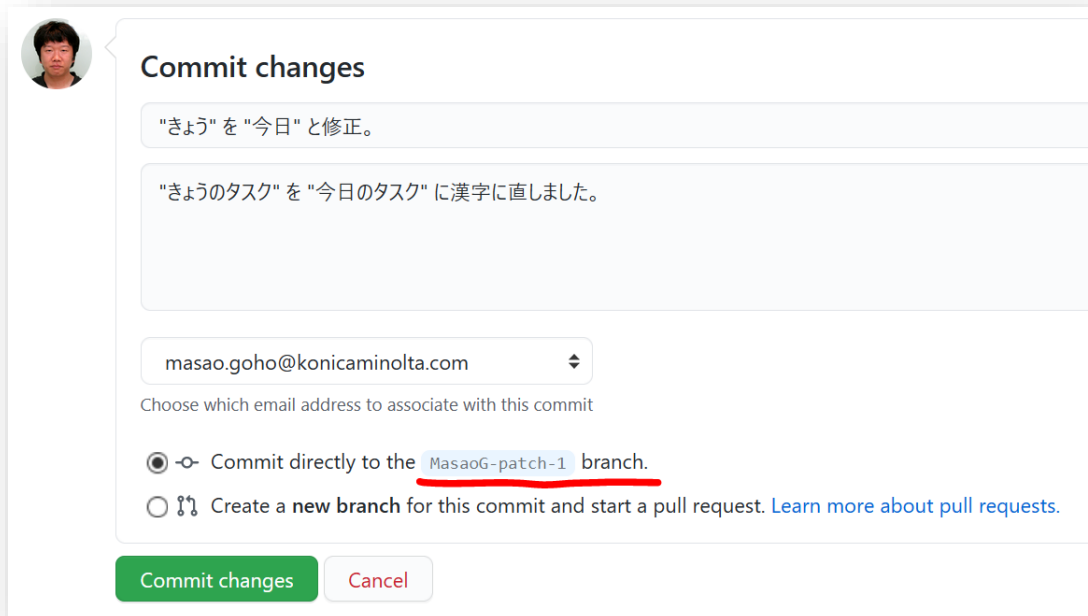
(例: “きょうのタスク” を “今日のタスク” に修正)


```

7  ## 今日のタスク
8  - [ ] GitHub アカウントを作成する
9  - [ ] ファイルを追加、編集する
10 - [ ] Pull Request を作成する
11 - [ ] コードレビューを行う
12 - [ ] 変更をマージする
13

```

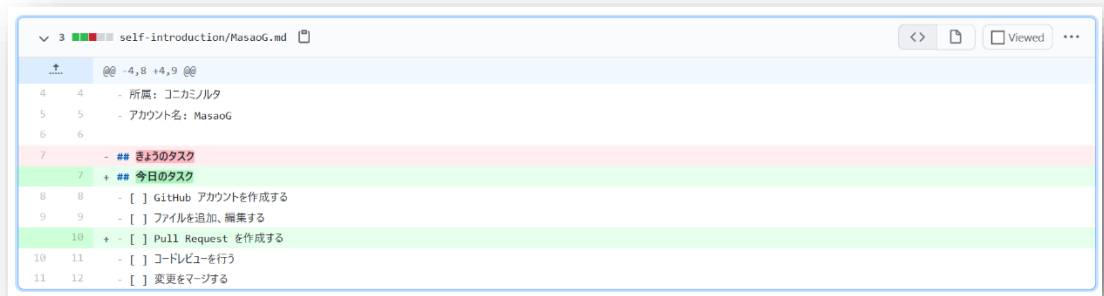
4. 変更が完了したら画面下部の Commit changes の入力フォームに、タイトルと詳細内容を記載します。



ここで注意が必要なのは、この Commit もまだ“(あなたの GitHub アカウント名)-patch-1” Branch に対して行う必要があるということです。追加で行った修正内容もレビューしてもらう必要があるからです。

以上のことに注意して、Commit を行いましょう。

5. Commit を行うと変更内容の画面が表示されます。ここで注目していただきたいのは、変更内容が 1 つ目の Commit と 2 つ目の Commit の両方の内容を合わせた内容になっている点です。



ここでは Pull Request は master Branch と “(あなたの GitHub アカウント名)-patch-1” の差分を表示しています。

- 文頭に [-] がついた赤くハイライトされたところは削除されたところ。
 - 文頭に [+] がついた緑でハイライトされたところは追加されたところ。
6. 次に、再度レビュアーにレビューをお願いして確認してもらう必要がありますが、先ほどのレビュープロセスと同じですので、ここではスキップします。

3.4 Pull Request の Merge (マージ)

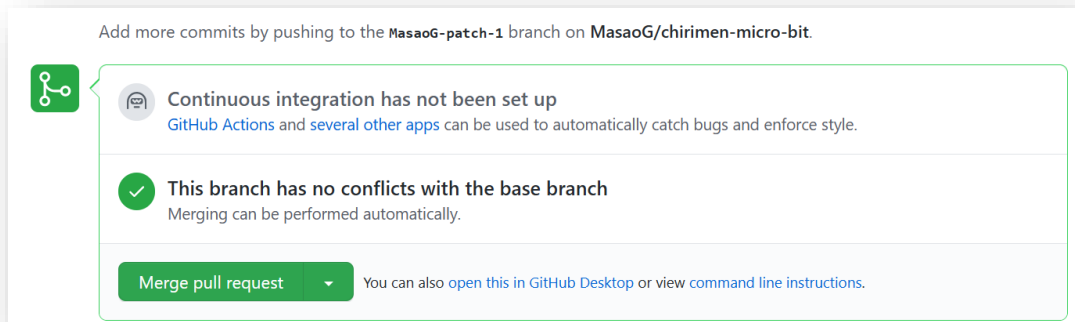
ここまでで、Pull Request 上で変更内容のレビューをしてもらい、追加の変更を行ってすべての変更内容が問題ないことをレビューしてもらうことが完了しました。

次の最後のステップが“(あなたの GitHub アカウント名)-patch-1” Branch に Pull Request でレビュー済みの修正内容を master Branch に Merge (マージ)、つまり取り込む作業になります。



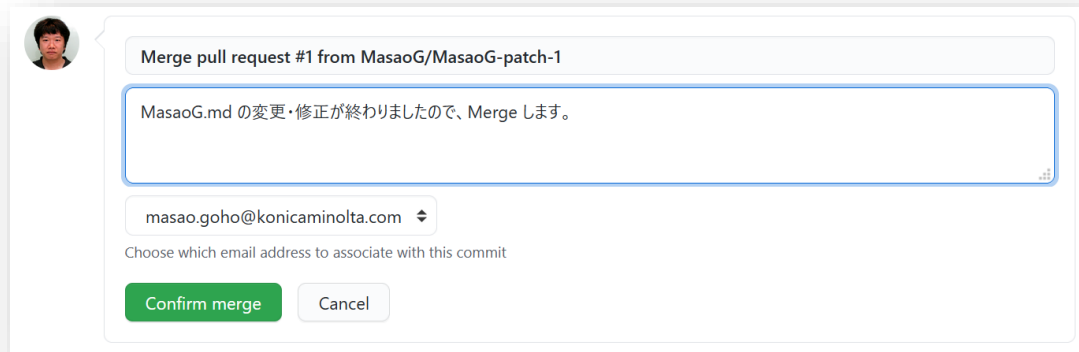
Merge を行うのは非常に簡単です。さっそく次の手順を実行しましょう。

1. レビューが完了した Pull Request の “Conversation” タブをクリックします。
2. Conversation 画面の下の方に次のような表示があります。

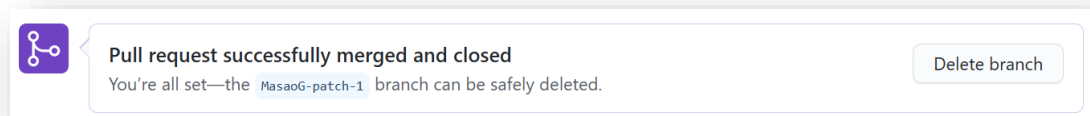


ここの緑の “Merge pull request” ボタンを押してください。

3. Merge することのコメント入力フォームが表示されますので、コメントを入れて緑の “Confirm merge” ボタンを押します。



4. おめでとうございます！これで初めての Pull Request を無事に Merge することができました！



無事 Pull Request が Merge されると上図のメッセージが表示されます。

このメッセージが表示されたら変更・修正が master Branch に無事取り込むことができたこととなります。

右にある “Delete branch” ボタンで今回の変更作業用に使った Branch を削除することができますが、次の Step でこの Branch を使いますので、残しておきます。

STEP 4. ローカルの PC 上で開発する

これまではブラウザ上のみで GitHub を使って、ファイルを追加・変更して Pull Request を送ってレビュー後、変更内容を master Branch に Merge する一通りの流れを体験しました。

開発によっては、自分の PC 上のお気に入りの開発ツールで開発・修正などを行いたいことがあります。

その場合、GitHub.com 上にある Repository を、自分の PC 上に複製して開発を行い、追加・修正したファイルを再度 GitHub.com の Repository に取り込むという作業を行います。

ここからは、Git という PC 上でコマンドラインで Git の操作を行うことができる、オープンソースのバージョン コントロールシステム (VCS) を使って、GitHub 上の Repository (Remote Repository) と自分の PC 上にコピーした Repository (Local Repository) を連携させていく方法について、手順を追って進めていきます。

行う内容は、これまで STEP1 から 3 までに行った作業と同じ、新しくファイルを追加・編集して Merge していきます。

4.1 Git のセットアップ

コマンドラインで Git を使うには、みなさんのコンピュータに Git をインストールし、設定する必要があります。

ローカルで Git を動かしたいけれどもコマンドラインは使いたくない場合、代わりに [GitHub Desktop](#) クライアントをインストールすることができますが、ここでは GitHub Desktop の使い方は紹介しませんので、使ってみたい方は「[GitHub Desktop を試してみる](#)」を参照してください。

ローカルでファイルを扱う作業をする必要がない場合、GitHub により、多くの Git 関連のアクションをブラウザで直接実行できることは確認してきました。

ここからは、コマンドライン上で Git を使っていきます。

1. Git をセットアップします。

次のリンクから、みなさんがお使いの OS (Mac OS X, Windows, Linux/Unix) に合わせて、Git の最新バージョンをダウンロードしてインストールしてください。

<https://git-scm.com/downloads>

セットアップはすべてデフォルトのまま“次へ”を進めます。

2. Git でのユーザー名を設定します。

Git ユーザー名は GitHub ユーザー名と同じではありません。皆さんの PC にあるすべてのリポジトリ用の Git ユーザー名を設定します。

3. Git Bash (または Terminal) を開きます。

4. Git のユーザー名を次のコマンドで設定してください。

```
$ git config --global user.name “(ユーザー名)”
```

5. Git のユーザー名が正しく設定されたことを次のコマンドを入力して確認します。

```
$ git config --global user.name  
> (ユーザー名)
```

4.2 コミット メールアドレスの設定

編集や Merge のような Web ベースの Git のオペレーションと関連するための GitHub 上のプライマリ メールアドレスを設定します。

GitHub は、コミット メールアドレスを使って、Commit を GitHub アカウントに関連付けます。コマンドラインから Push (プッシュ) する Commit や、Web ベースの Git 操作に関連付けられるメールアドレスは選択することができます。

コマンドラインから Commit などの操作が自分に関連付けられ、コントリビューション グラフ (GitHub 上での活動状況がみれるグラフ) に表示されるようにするには、自分の GitHub アカウントに追加したメールアドレスか、メール設定で GitHub から提供された noreply メールアドレスを使います。

今回は、GitHub にアカウントを作成された際に使われたメールアドレスをコミット メールアドレスに設定します。

1. Git Bash を開きます。
2. 次のコマンドでメールアドレスを設定します。

```
$ git config --global user.email "email@example.com"
```

3. 次のコマンドで正しくメールアドレスが設定されているか、確認します。

```
$ git config --global user.email  
email@example.com
```

4.3 ローカルの PC に GitHub の Repository を複製する

ローカルの PC に GitHub の Repository を複製することを clone (クローン) と呼びます。

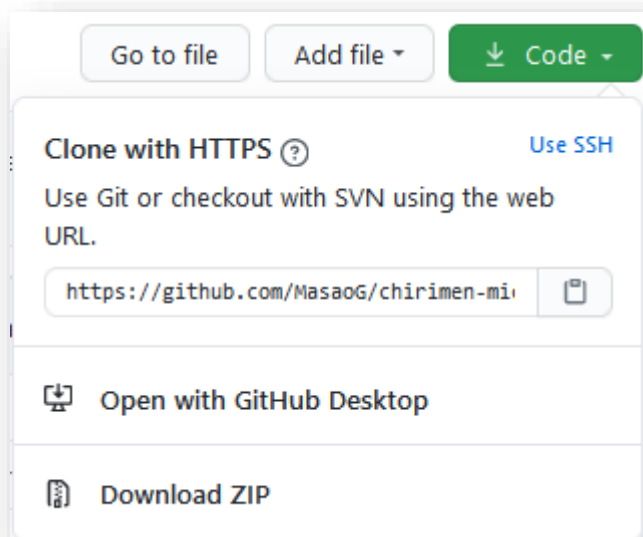
Clone する場合、HTTPS または SSH を使って GitHub で認証する必要がありますが、今回は HTTPS を使います。

● クローン
clone

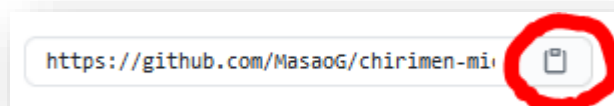


リモートリポジトリの内容をローカルにコピー

1. まずは clone する Repository の画面から “Code” タブを開き、右上の緑の “Code” ボタンをクリックします。すると次のようなメニューが表示されます。



2. URL が表示されている右横にあるクリップボードのアイコンをクリックすると、この URL がコピーされます。



3. Git Bash を起動し、次のコマンドで GitHub 上の Repository をローカルの PC にコピーします。

Git clone と入力した後に半角スペースを開けたあと、コピーした URL を貼り付けます。


```
$ git clone https://github.com/(GitHub アカウント名)/chirimen-micro-bit.git
```

4. Enter キーを押して実行すると、複製が始まります。

次のように最後に表示されたら、完了です。

```
Resolving deltas: 100% (769/769), done.
```

5. 実際のコード類はどこに clone されたのか確認しましょう。

デフォルトの場合は、それぞれ次の場所に “chirimen-micro-bit” フォルダーがあります。

- Windows では、C:¥Users¥<PC のアカウント名>¥chirimen-micro-bit
- Mac OS では、XXXXX
- Linux/Unix では、XXXXXX

6. これで ローカルで作業をする準備が整いました。

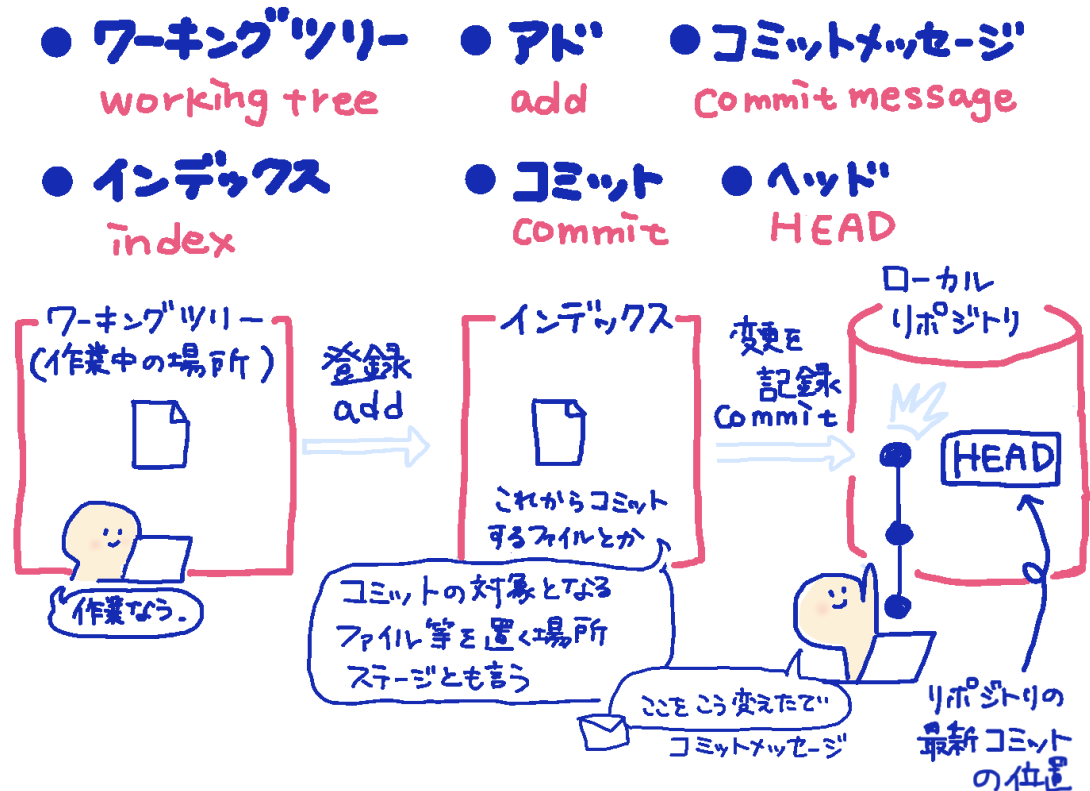
Git Bash で次のコマンドを入力し、PC に複製した Local Repository の場所へ移動しておきます。

```
$ cd ./chirimen-micro-bit/
```

4.4 Local Repository で作業（開発）する

ここでは、PC 上に複製した Local Repository でファイルを追加・変更してみます。

Local Repository でこれから実施する作業内容は、こんなイメージです。



1. Local Repository のフォルダーを PC 上で表示し、その中にある "self-introduction" フォルダーを開きます。
2. 開いた "self-introduction" フォルダーに、新しく "AddedFile.txt" といったファイルを作成し、中に何か記述してください。
3. Git Bash を起動し、~/chirimen-micro-bit フォルダーに移ります。

```
$ cd ./chirimen-micro-bit/
```

4. `git status` コマンドを入力して、新しく追加・変更されたファイルを確認します。

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to conclude in what will be committed)
    Self-introduction/AddedFile.txt

Nothing added to commit but untracked files present (use "git add" to track)
```

5. 次のコマンドを入力して “AddedFile.txt” をインデックスに登録します。

```
$ git add .
```

最後の “.” (ドット) を忘れないように！

6. 再度 `git status` コマンドを実行すると、Changes to be committed: の項目にファイルが表示され、インデックスにファイルが反映されます。

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   Self-introduction/AddedFile.txt
```

7. 次に `git commit` コマンドで、インデックスから、Local Repository にこの追加・変更を Commit します。

```
$ git commit -m “新しいファイルを追加しました。”
```

-m オプションをつけると、コミットメッセージを追加できます。

8. これで、Local Repository 上で、ファイルを追加して、Commit することができました。

あとは、この繰り返しで PC 上でファイルを作成・追加や、編集したら、add して commit することの繰り返しです。

9. Commit の履歴（つまり変更履歴）を確認するには、次のコマンドを実行します。

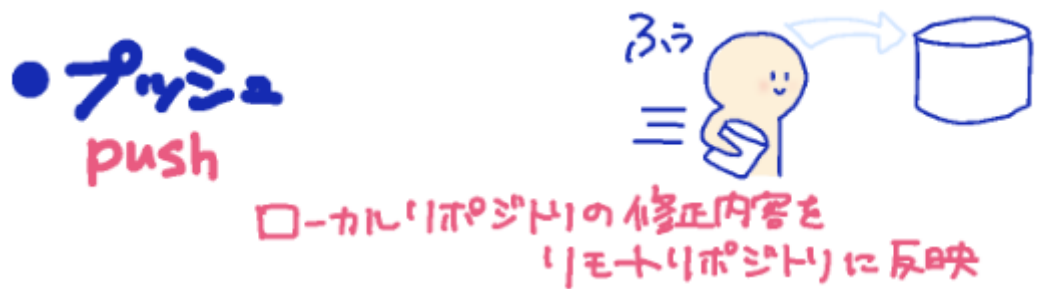
```
$ git log -n 3
```

-n <数字> オプションは、表示する Commit 数を指定しています。今回は 3 つ。

4.5 Remote Repository (GitHub) へ反映する

Local Repository は自分の PC の中の Repository なので、ここまでは Remote Repository (GitHub) から持ってきたサンプルを自分の PC の中でのみ、修正変更してきました。

次は、自分の Local Repository で変更・追加した内容を、GitHub 上の Remote Repository へ反映させるために Push (プッシュ) します。



1. まずは、念のために、今、どの Remote Repository が構成されているか、次のコマンドで確認します。

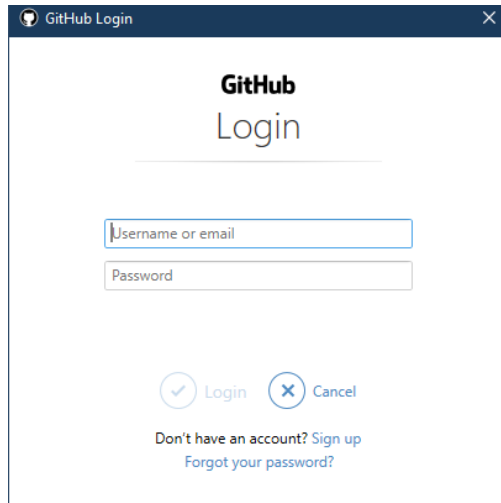
```
$ git remote -v  
Origin https://github.com/<アカウント名>/chirimen-micro-bit.git (fetch)  
Origin https://github.com/<アカウント名>/chirimen-micro-bit.git (push)
```

Origin という名前で、Github 上の Remote Repository が追加されていることが分かります。

2. 次の Push コマンドで、変更した Branch (今回は master を直接編集した) を Remote Repository (master) に Push (プッシュ) します。

```
$ git push origin master
```

3. すると、ここで GitHub の Login 画面がでてきます。これは GitHub 上の Remote Repository に Push できるユーザーであるかどうかを確認するためです。みなさんの GitHub アカウントで Login してください。



4. これで無事に GitHub 上の Remote Repository へ反映することができました。

ブラウザーで GitHub 上の Repository を表示して、“self-introduction” フォルダーに “AddedFile.txt” ファイルが追加されていることを確認してください。

STEP 5. その他 おぼえておきたいこと

ここまでは、ブラウザ上での GitHub の使い方、ローカル PC 上で Git コマンドを使って、Local Repository と Remote Repository とのやり取りのよく使う基本の操作を行ってきました。

それぞれのコマンドにはいくつかオプションがあり、他にも便利なコマンドもありますので、詳しく知りたい場合は次のサイトを参照ください。

GitHub.com ヘルプドキュメント (日本語)

<https://docs.github.com/ja/github>

共同で開発・作業をする場合は、ローカル PC 上でもあらかじめ作成済みの Branch 上で行ったり、Fork 元の Remote Repository と同期を取ったりという場合もあると思います。

仲間と共同作業をする場合に、もう一步、おぼえておく便利なことについて、ここでは体験していきます。

5.1 ローカルの PC 上で別の Branch で作業をする

共同で作業をする場合、master Branch で直接作業をするのではなく、作業用に Branch を作って、Pull Request を送り、コードレビューしてから Merge することについて、Step 3. で学びました。

ここでは、PC 上の Local Repository で編集したものを、作業用の Branch に Push します。Pull Request を作ってコードレビュー依頼を出すところまで行います。

その後の作業は、Step.3 と同じです。

1. Local Repository で開発する場所を、Step.3 で作成した “<アカウント名>-patch-1” に切り替えます。まずは、次のコマンドで Remote Branch の一覧を表示しましょう。git branch -r

```
$ git branch -r
```

2. 確認できたら、“<アカウント名>-patch-1” ブランチに、次のコマンドで切り替えます。

```
$ git checkout “<アカウント名>-patch-1”
```

(重要！) 切り替えた branch が origin となります。



3. Local Repository での開発場所が “<アカウント名>-patch-1” になっているかを次のコマンドで確認します。

```
$ git branch
```

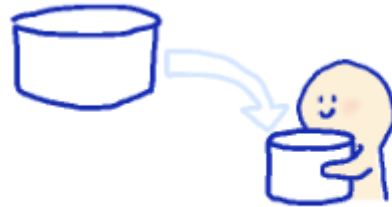
4. ローカル PC の Repository の “self-introduction” フォルダの中を確認します。

“<アカウント名>-patch-1” branch は、Step.3 のままなので、Step.4 で追加したファイルが表示されていません。

5. そこで、Step.4 で GitHub 上の master branch に追加した内容を、ローカルリポジトリの今いる branch (“<アカウント名>-patch-1” が origin) に pull して同期します。


```
$ git pull
```

すると、“self-introduction” フォルダーに Step.4 で追加した “AddedFile.txt” が追加されていることが確認できます。




リモートリポジトリの内容をローカルリポジトリに取り込む

6. 同期ができたところで開発をしますが、ここでは新しくファイルを追加し、① `git add` . そして、② `git commit -m “コメント”` のコマンドを実行します。
例) “LocalTest.txt” というファイルを “self-introduction” フォルダーに作成しましょう。
7. 最後に Push します。

```
$ git push origin “<アカウント名>-patch-1”
```

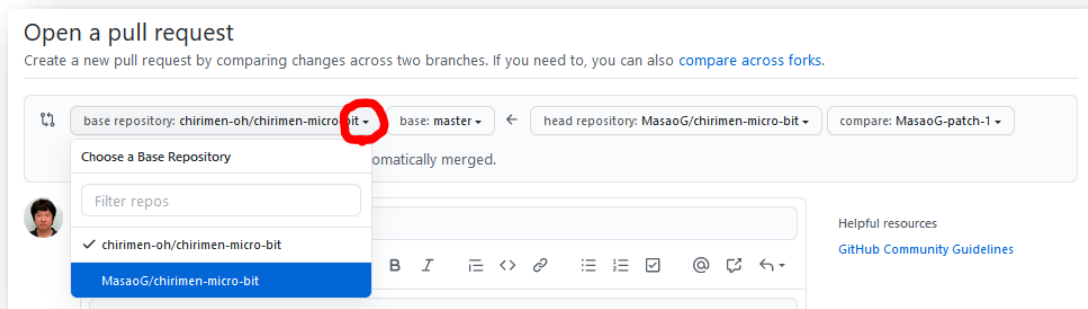
これは、今いる Local Repository (=origin) の内容を、GitHub 上の Remote Repository の “<アカウント名>-patch-1” branch に push する、という意味になります。

8. ブラウザー上で GitHub を開き、自分のリポジトリを表示し、“Pull Requests” タブを開きます。開くと一番上に次のような表示があるので緑の “Compare & pull request” ボタンを押します。

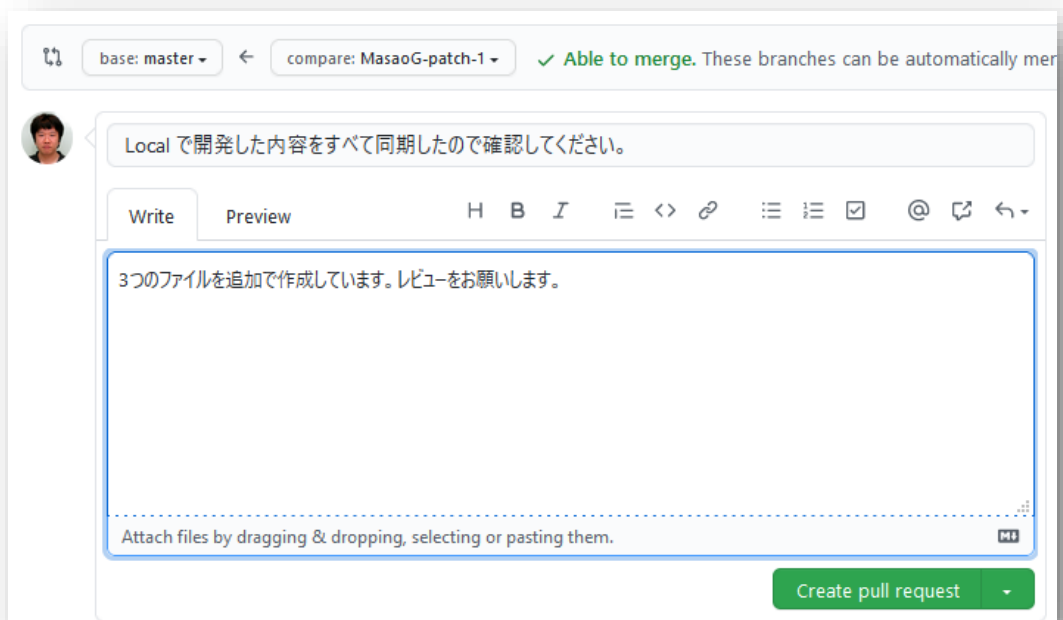
 MasaoG-patch-1 had recent pushes 12 minutes ago

Compare & pull request

9. すると Pull Request の作成画面になります。ここで、どこからどこに Pull Request をするのかを設定できます。今回は自分の Remote Repository の “<アカウント名>-patch-1” branch から、自分の Remote Repository の master への Pull Request なので、左側の base repository: をクリックしてプルダウンメニューから自分の Remote Repository を選択します。



10. 自動的に今度は merge が可能か確認され、Pull Request が作成されます。後は、Step.3 と同じになりますので省略します。



5.2 その他
