

# GitHub

# チーム開発編

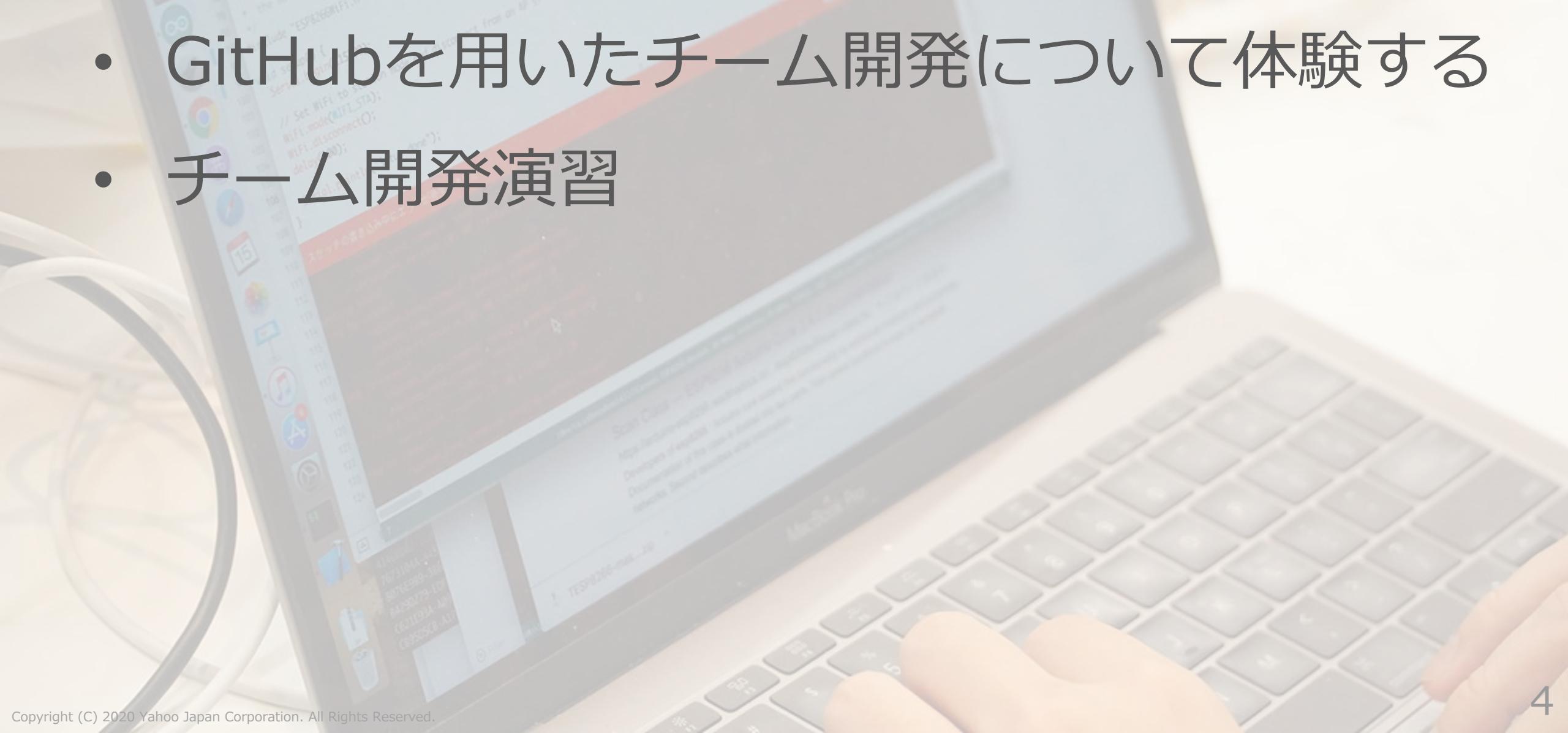
2021年2月17日

Hack U Projects



# イベントで学ぶこと

- GitHubを用いたチーム開発について体験する
- チーム開発演習



# このイベントのゴール

- ・ コード管理の大切さを体験してもらう
- ・ チームで問題を解決できる
- ・ チーム開発を円滑に行えるようになる
- ・ Hack U の開発体制についてチームで共有する

# Hack U に向けて

GitHub はコード管理をアシストするツールです

-> 開発速度を求める際には使わない判断も重要

- コードレビュー
- アイディア出しや議事録をまとめる
- 完成したコードの共有

用途を限定して使うことも考えてみましょう

# イベントの進め方

- 事前準備の確認
- 講義
  - チーム開発の進め方
  - Conflict について
- チームハンズオン
- まとめ

# 注意

「GitHub WS 入門編」の内容を前提知識として進みます

- Git/GitHub 下記機能についての知識が必要です
  - **add/commit/push**
  - **branch/Pull Request**

また、複数人で作業することを前提の課題となっています  
一人で行う場合は複数のブランチを使います

# 問題が起きたときは

- まずはチームメンバー間で助けあいましょう
  - チームで解決できないときは **#質問部屋** に下記内容を記載して質問してください。
    - 状況
    - 問題内容(エラー文)
    - 実現したいこと

※サポーターの人数に限りがあるため、回答に時間がかかる場合があります

# 演習に向けて

- ・ コマンド叩けることよりもGitHubの使い方を知ってもらうことが目的です
  - ・ 詰まった場合は、進んでいるメンバーに情報を共有して貰いましょう
    - ・ Zoom の画面共有やSlackのチャットで共有 など
    - ・ 演習ではメンバーに役割を割り振って作業を行います
  - > 実際の開発をイメージして進めましょう

事前準備は  
お済みですか？

YAHOO!  
JAPAN

# 事前準備

セットアップ資料を参考に準備項目の確認

- ・ チーム共通のリポジトリの作成
- ・ リポジトリにメンバーをアサイン
- ・ **team\_hands-on.zip** の内容を push
- ・ メンバー各自でローカルリポジトリを作成

# チーム開発でコード管理どうしますか？



# このような経験をしていませんか？

よし、開発の続きをやるぞ！



index  
.html



index(1)  
.html



最新版  
index.html

あれ？ 最新版ファイルどれ？？

私が持ってるのがたぶんそー  
後で送るね



# このような経験をしていませんか？



こここの機能開発しておいたよ。  
ソースコードはメールで添付して送るから取り込んでね。

ありがとう！早速コピペして・・・  
あ！自分が作ってた部分が上書きされちゃった



# このような経験をしていませんか？



こここの機能開発しておいたよ。

Git / GitHub を使って  
解決しましょう！

ありが

あ！自分が作ってた部分が上書きされちゃった

んでね。

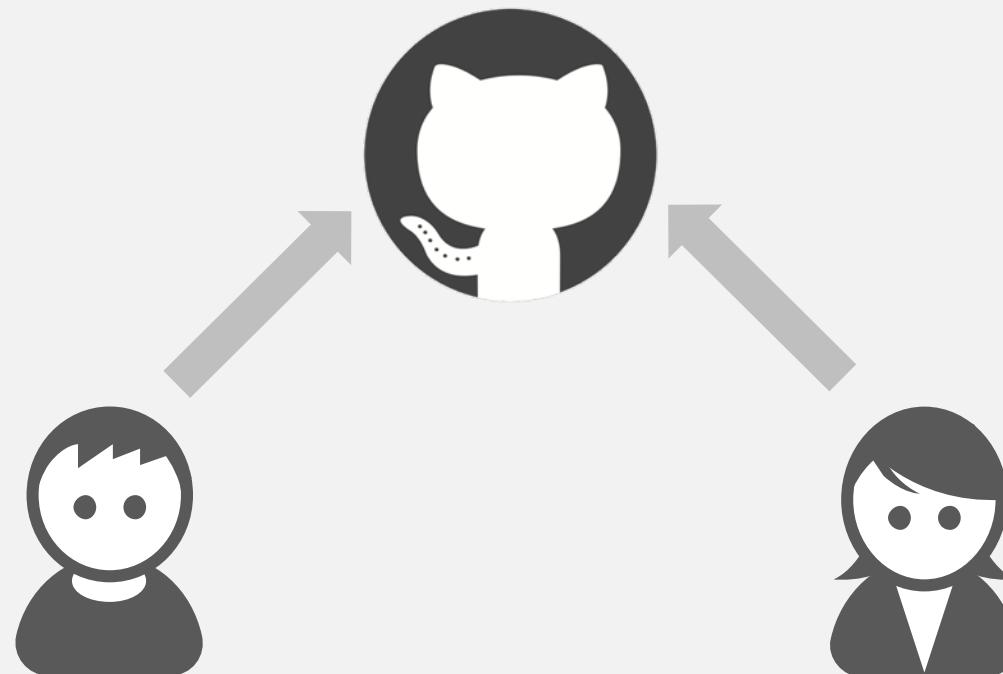


# GitHub をチームで使う

YAHOO!  
JAPAN

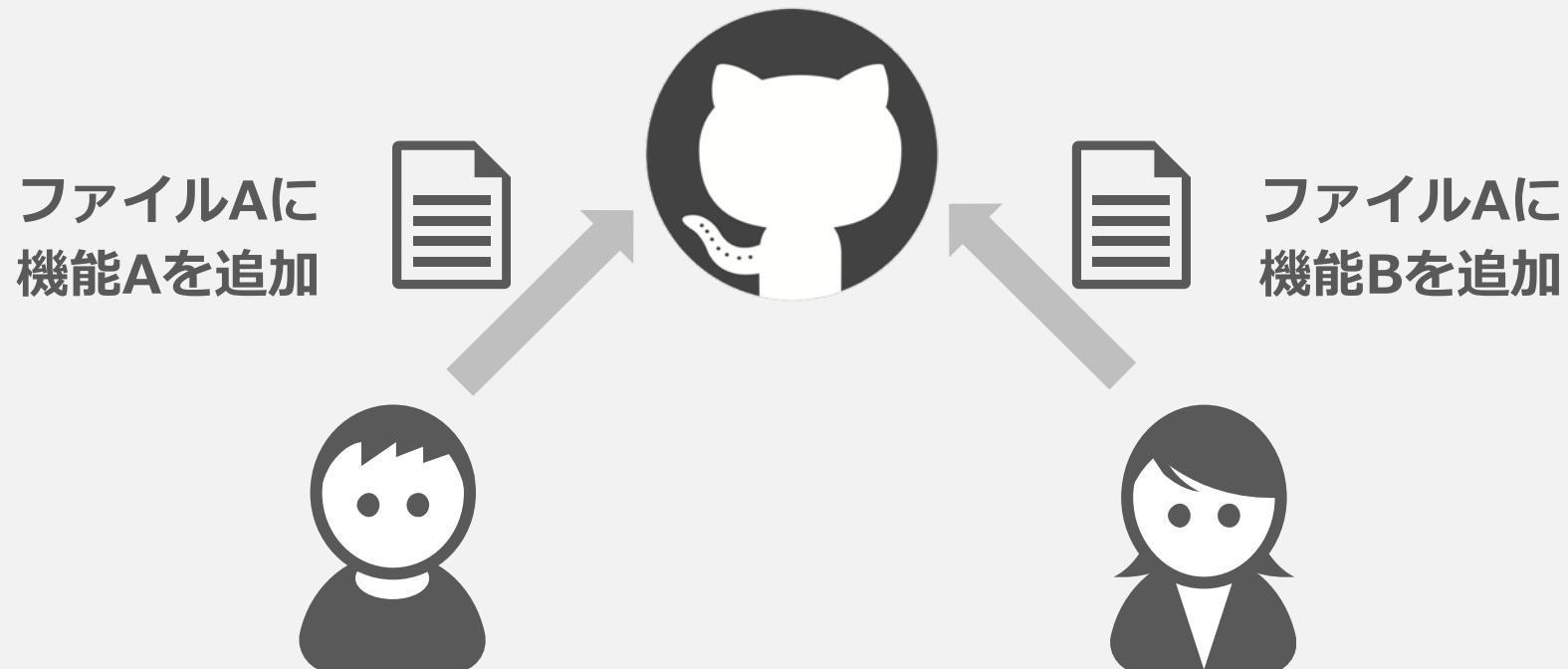
# チーム開発では

- 1つのリポジトリに対して複数人で作業する



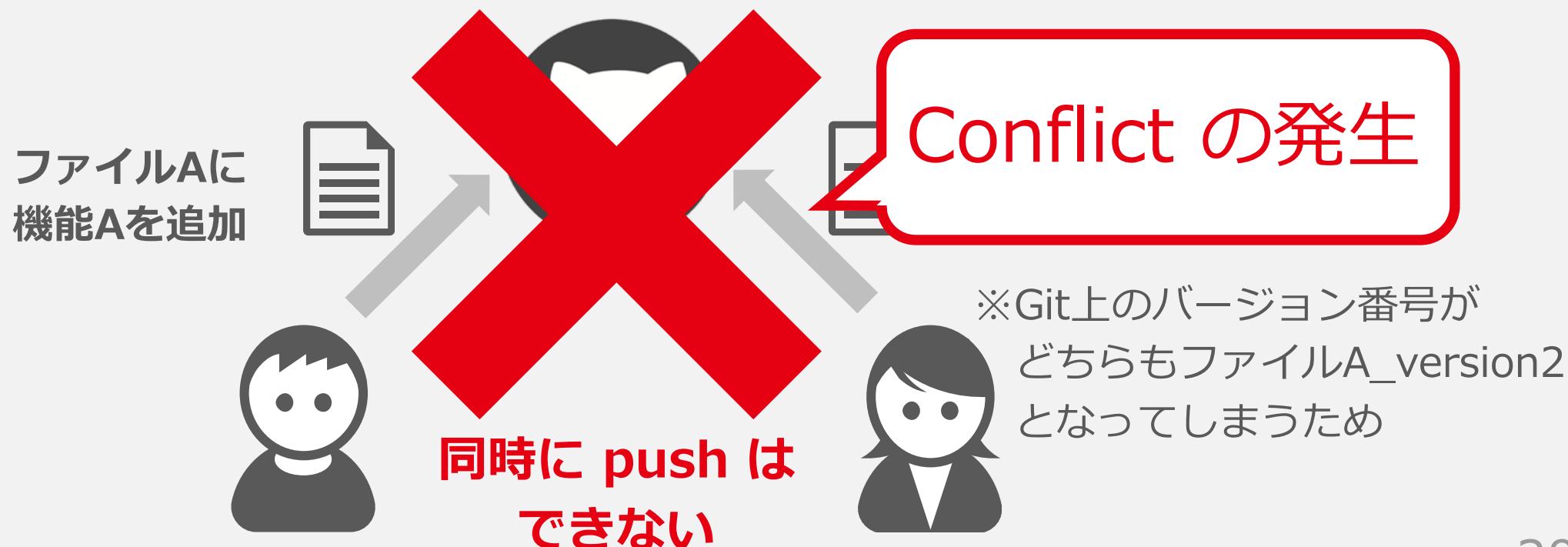
# チーム開発では

- ・ 1つのリポジトリに対して複数人で作業する
- ・ 同じファイルを同時に修正してしまう可能性も



# チーム開発では

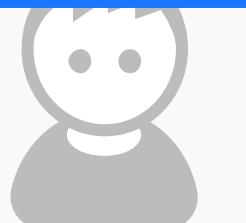
- ・ 1つのリポジトリに対して複数人で作業する
- ・ 同じファイルを同時に修正してしまう可能性も



# チーム開発では

- ・ 1つのリポジトリに対して複数人で作業する
- ・ 同じコードを同時に修正してしまう可能性もある

**Conflict (競合) を避けるため  
作業者ごとに  
ブランチを分けましょう**



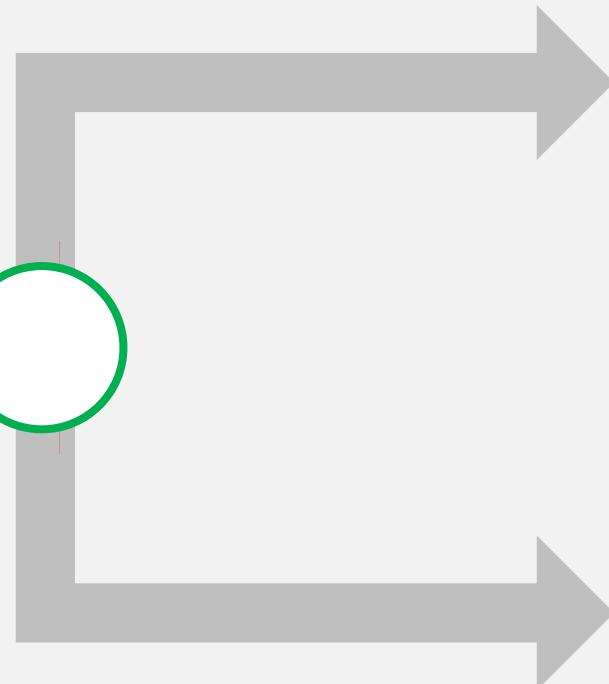
同時に push は  
できない



# チーム開発では



Aさんの作業ブランチ「A」

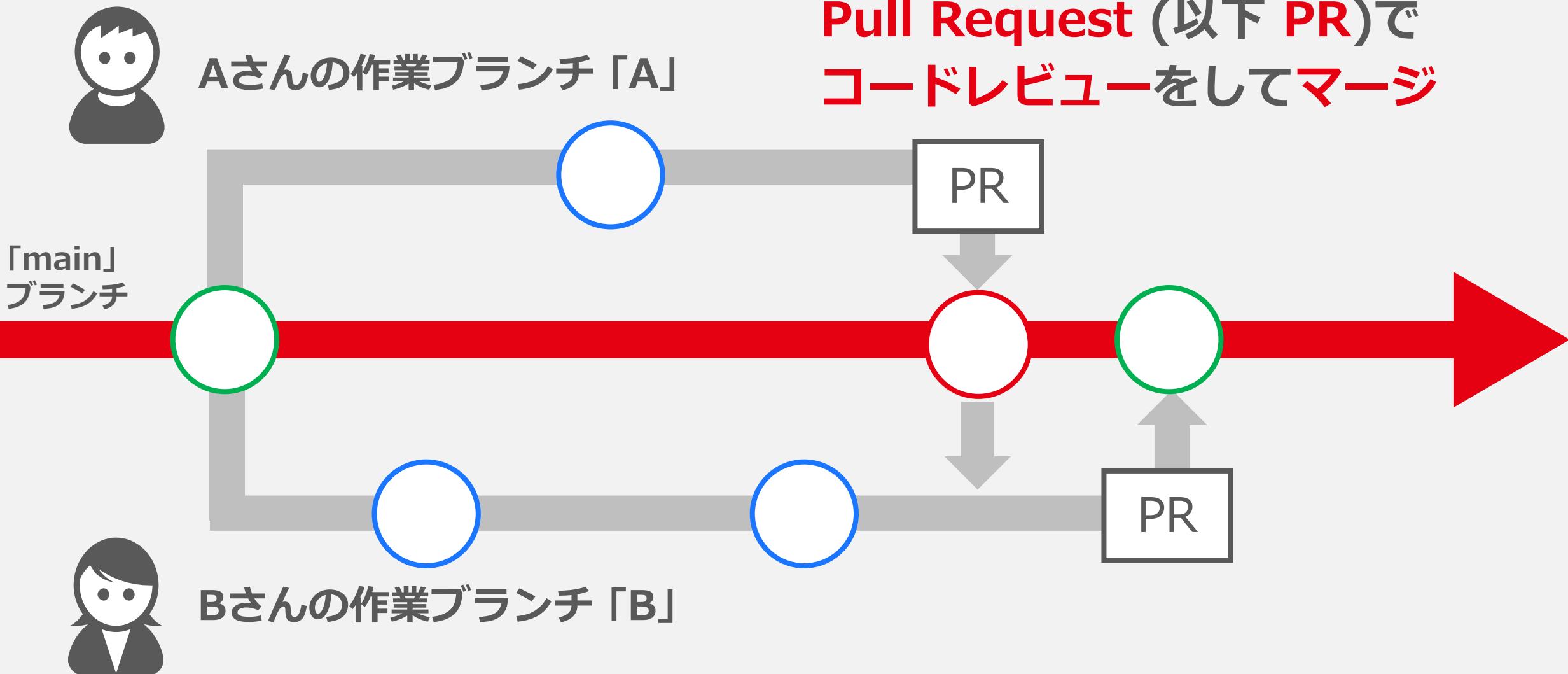


作業者がそれぞれブランチを作成



Bさんの作業ブランチ「B」

# チーム開発では



# チーム開発では



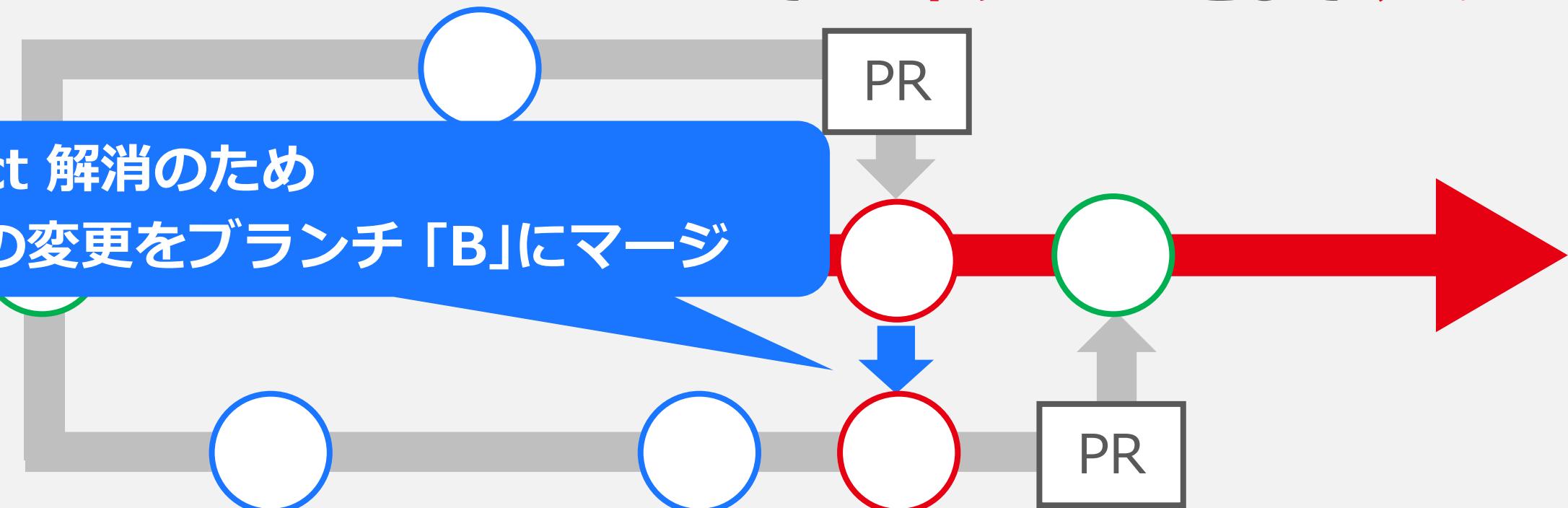
Aさんの作業ブランチ「A」

作業者がそれぞれブランチを作成  
→ PRでコードレビューをしてマージ

Conflict 解消のため  
main の変更をブランチ「B」にマージ



Bさんの作業ブランチ「B」



# Conflictの解消

YAHOO!  
JAPAN

# Conflict の発生

PR の「Merge pull request」がグレーアウト

- merge 先のブランチとの間で Conflict が発生



# Conflict 箇所の特定

「Resolve conflicts」から Conflict を確認

This branch has conflicts that must be resolved

Use the [web editor](#) or the [command line](#) to resolve conflicts.

Conflicting files

src/index.js

Merge pull request ▾ You can also [open this in GitHub Desktop](#) or view [command line instructions](#).

**Resolve conflicts**

A screenshot of a GitHub pull request interface. At the top left is a network icon. Next to it, a warning icon (triangle with exclamation) contains the text "This branch has conflicts that must be resolved". Below this, a message says "Use the [web editor](#) or the [command line](#) to resolve conflicts.". A red box highlights the "Resolve conflicts" button at the top right. In the middle section, "Conflicting files" are listed as "src/index.js". At the bottom, there's a "Merge pull request" button with a dropdown arrow, and a note saying "You can also [open this in GitHub Desktop](#) or view [command line instructions](#)".

# Conflict 箇所の特定

Conflict が発生したファイル内容が表示される

- 下記のようになっている箇所を確認

```
131 <<<<< A1
132   function add() {}
133   function sub() {}
134 =====
135   function mul() {}
136   function div() {}
137 >>>>> main
```

# Conflict 箇所の特定

Conflict が発生した PR を出したブランチでの  
- 下記のようにコード変更点

```
131 <<<<< A1
132     function add() {}
133     function sub() {}
134     =====
135     function mul() {}
136     function div() {}
137     >>>>> main
```

# Conflict 箇所の特定

Conflict が発生したファイル内容が表示される

- 下記のようになっている箇所を確認

```
131 <<<
132
133 function multiply()
134 =====
135     function mul() {}
136     function div() {}
137 >>>>> main
```

# Conflict の修正

Conflict 篇所のコードを確認

- Conflictしたコードのどちらかを消す
- Conflict を示す記号を消し、Conflictを起こしたコードをどちらも残す

状況に応じてどちらかで対応する

# Conflict の修正方法

Conflict の修正方法は2つの方法で行える

- GitHub 上で該当コード箇所を修正
- ローカルでコードを修正して再度 push

# GitHub で修正する

1. PR -> 「Resolve conflicts」のページからコード修正
2. 修正したファイルで「Mark as resolves」を押しチェックを付ける
3. 全てのファイルを修正したら  
「Commit marge」ボタンから commit

# コード修正前 (PRの修正内容を残す場合)

A1 のコード内容のみ残したいため、Conflict を示す記号と main のコード部分を消す

## Add add & sub #1

Resolving conflicts between A1 and main and committing changes → A1

1 conflicting file      src/index.js      1 conflict      Prev ^      Next v      Mark as resolved

index.js  
src/index.js

```
127      /* 演習の編集範囲 はじめ */
128
129      /* 演習1 ここから */
130
131      <<<<< A1
132      function add() {}
133      function sub() {}
134      =====
135      function mul() {}
136      function div() {}
137      >>>>> main
138      /* 演習1 ここまで */
```

矢印の1行と青枠内の4行  
を削除する  
(132,133行目を残す)

# コード修正前（どちらも残す場合）

A1 と main どちらコードも残したいため、  
Conflict を示す記号だけ消す

## Add add & sub #1

Resolving conflicts between A1 and main and committing changes → A1

1 conflicting file      src/index.js      1 conflict      Prev ⌂      Next ⌂      ⚙      Mark as resolved

index.js      src/index.js

```
127      /* 演習の編集範囲 はじめ */
128
129      /* 演習1 ここから */
130
131      <<<<< A1
132      function add() {}
133      function sub() {}
134
135      =====
136      function mul() {}
137      function div() {}
138      >>>>> main
139
140      /* 演習1 ここまで */
```

矢印の3行を削除する

# コード修正後

「Mark as resolves」をクリック

Conflict が発生しているファイルを全て修正する

## Add add & sub #1

Resolving conflicts between A1 and main and committing changes → A1

1 conflicting file	src/index.js	1 conflict	Prev ^	Next v	⚙️	Mark as resolved
 index.js src/index.js	<pre>127      /* 演習の編集範囲 はじめ */ 128      /* 演習1 ここから */ 129      function add() {} 130      function sub() {} 131      function mul() {} 132      function div() {} 133      /* 演習1 ここまで */ 134 135</pre>					

# 修正を commit

全てのファイルを修正したか確認（青枠部分）

「Commit merge」ボタンから修正を commit

Add add & sub #1

Resolving conflicts between A1 and main and committing changes → A1

Commit merge

1 conflicting file ✓ Resolved

index.js src/index.js

```
src/index.js
127      /* 演習の編集範囲 はじめ */
128
129
130      /* 演習1 ここから */
131      function add() {}
132      function sub() {}
133      function mul() {}
134      function div() {}
135      /* 演習1 ここまで */
```

The screenshot shows a GitHub pull request interface for a merge between branches A1 and main. The title is 'Add add & sub #1'. The status bar indicates 'Resolving conflicts between A1 and main and committing changes → A1'. A green 'Commit merge' button is highlighted with a red border. On the left, a table shows '1 conflicting file' with 'index.js' and 'src/index.js' listed, both marked as 'Resolved' with a checkmark icon. The right side shows the code editor for 'src/index.js' with conflict markers and a diff view. A blue box highlights the 'index.js' row in the file list.

# ローカルで修正

1. 作業ブランチに merge 先のブランチを pull する (例: A1にmainをmerge)
2. Conflict 箇所を修正して commit/push
3. PR で merge 可能になったか確認

# 修正が完了したら

PR が merge できるか確認  
必要なら再度レビューをもらう

A screenshot of a GitHub pull request merge interface. At the top left is a green icon with a gear and a branch. To its right, a grey circle contains a white icon of a computer monitor with a bar chart. Next to the icon is the text "Continuous integration has passed" followed by "GitHub Actions and several other". Below this, a green circle with a white checkmark contains the text "This branch has no conflicts". To the right of the checkmark, it says "Merging can be performed". At the bottom left is a green button with the text "Merge pull request". A blue arrow points from the text "作業者とその他でボタンが変わります" in the callout box to the "Merge pull request" button. The callout box also contains the text "緑色になつていれば Conflictが解消され merge 可能です". At the bottom center, there is a message: "You can also open this in GitHub Desktop or view command line instructions."

作業者とその他でボタンが変わります  
緑色になつていれば Conflictが解消され  
merge 可能です

# Conflict を起こさない

YAHOO!  
JAPAN

# Conflict を起こさない

Conflict が発生しないように開発する

- 編集ファイルが被らないように分担
- 予めコードの編集箇所を定めておく
  - 例: 実装予定の関数を先に push しておく

# まとめ

Conflict が起きないように作業分担しましょう

もしConflict は起きた場合は

- 小規模なら GitHub で修正
- 範囲が大きい場合は作業者がローカルで修正

上記のように対処するのがおすすめです

# チーム演習

Javascript で電卓をつくろう



# チーム開発

配布した状態のプログラムでは、  
=ボタンを押したときの挙動が未実装です

Javascriptで電卓をつくろう

0   =

AC	7	8	9	+
	4	5	6	-
	1	2	3	×
.	0	=	÷	

# 演習1

# Conflict 発生と解消



# 演習1の進め方 1

## 演習1の作業内容:

足し算/引き算用の関数 **add** と **sub**を実装

1. チーム内でそれぞれに実装者を1名ずつ選出
2. 作業者は後述するブランチ名でコードを実装

# 演習1の進め方 2

## index.js にコードを実装していきます

```
128  /* 演習の編集範囲 はじめ */
129
130  /* 演習1 はここにかく */
131  // TODO: このコメントアウトを削除して書く ←
132  /* 演習1 ここまで */
133
134  /* TODO: 演習2 以降は下記に自由に記入 */
135
136  /* 演習の編集範囲 おわり */
```

131行目のコメント  
アウトを削除して、  
下記2つの関数を実装  
します  
- **add**関数  
- **sub**関数

# 作業ブランチと実装コード

足し算用関数 作業ブランチ名: A1

```
function add(){}
```

引き算用関数 作業ブランチ名: B1

```
function sub(){}
```

# 演習1の進め方 3

3. 実装が終わったら push して PR を作成
4. 作成されたPRを作業者以外がレビュー
5. 問題なければコメントをしてmerge
6. 2つ目のPRの確認と修正は画面共有して行う

# 演習1の進め方 補足1

2つ目の PR を merge する際に

**Conflict が発生**します

- add/sub 関数が**どちらも実装された形**で  
Conflict を解消してください

# 演習1の進め方 補足2

下記のようになつていればOKです  
(改行などが入っていても大丈夫です)

```
127
128      /* 演習の編集範囲 はじめ */
129
130      /* 演習1 ここから */
131      function add() {}
132      function sub() {}
133      /* 演習1 ここまで */
134
```

# 演習1の進め方 4

7. PR が全て merge されたらローカル環境の main ブランチを pull する

※他のブランチで作業途中だった場合はそのブランチに main ブランチを pull する

# 演習2

Conflict を発生させない



# 演習2の進め方 1

## 演習2の作業内容:

関数 **add** と **sub** の処理を実装

1. チーム内でそれぞれに実装者を1名ずつ選出
  2. 作業者は後述するブランチ名でコードを実装
- ※コード内容は、**code.txt** に記載されています  
※編集箇所は演習1で追加した add/sub 関数になります

# add関数 実装コード

作業ブランチ名: A2

ハイライトされた行を実装

```
function add(){
    eqlNum.value = parseFloat(midNum[0].value) + parseFloat(midNum[1].value);
}
```

# sub関数 実装コード

作業ブランチ名: B2

ハイライトされた行を実装

```
function sub(){
    eqlNum.value = parseFloat(midNum[0].value) - parseFloat(midNum[1].value);
}
```

## 演習2の進め方 2

3. 実装が終わったら push して PR を作成
4. 作成されたPRを作業者以外がレビュー
5. 問題なければコメントをして merge
6. 2つの PR を merge 後、ローカルを更新

# 演習2の進め方 3

## 7. 実装コードの動作確認

7.1. `index.html` をブラウザで開く

7.2. 電卓で足し算/引き算の動作確認

# 追加演習

# 電卓アプリの完成



# 追加演習

## 追加の作業内容:

関数 `mul` と `div` を実装して電卓アプリを完成させる

- 演習1,2 を参考にチーム内で役割を変えて作業する
- 進め方はチーム内で話し合って決めてください

※コード内容は、`code.txt` に記載されています

# 追加 実装コード

## 作業者C 掛け算用関数

```
function mul(){
    eqlNum.value = parseFloat(midNum[0].value) * parseFloat(midNum[1].value);
}
```

## 作業者D 割り算用関数

```
function div(){
    if(parseFloat(midNum[1].value) == 0){
        eqlNum.value = "ERROR";
    }else{
        eqlNum.value = parseFloat(midNum[0].value) / parseFloat(midNum[1].value);
    }
}
```

# 演習終了

YAHOO!  
JAPAN

# 演習が終わったら

追加演習とアンケートに回答まで終わった場合は、  
残りの時間で下記の内容に取り組んでみてください

- 資料巻末の **Appendix** を行う
- Hack U の**開発体制のすり合わせ**
- 作業者を変えて同様の内容を行う
- 演習プログラムを改良してPR/レビューし合う

# 演習のポイント

YAHOO!  
JAPAN

# レビュー

タイミングミスや Conflict などをコメントで指摘

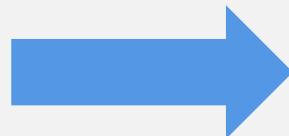
The screenshot shows a GitHub pull request interface. At the top, a comment from a user named 'Add sub' is visible, with a timestamp of 'commented 19 seconds ago'. The comment text is highlighted with a blue border: 'Conflict が発生しているため修正お願いします'. Below this, a message encourages pushing more commits to the branch: 'Add more commits by pushing to the B1 branch on magrain/github\_team\_ws.' A warning icon indicates that 'This branch has conflicts that must be resolved'. It provides instructions to use the 'web editor' or 'command line' to resolve conflicts and includes a 'Resolve conflicts' button. A list of 'Conflicting files' shows 'src/index.js'. At the bottom, there are buttons for 'Merge pull request' and 'You can also open this in GitHub Desktop or view command line instructions.'

# 演習1 の Conflict 状態と解消

行数の黄色ハイライトが消えていることを確認  
修正時にコードや改行の追加をすることも可能

修正前

```
---  
130      /* 演習1 ここから */  
131      <<<<< B1  
132      function sub() {}  
133      =====  
134      function add() {}  
135      >>>>> main  
136      /* 演習1 ここまで */  
137
```

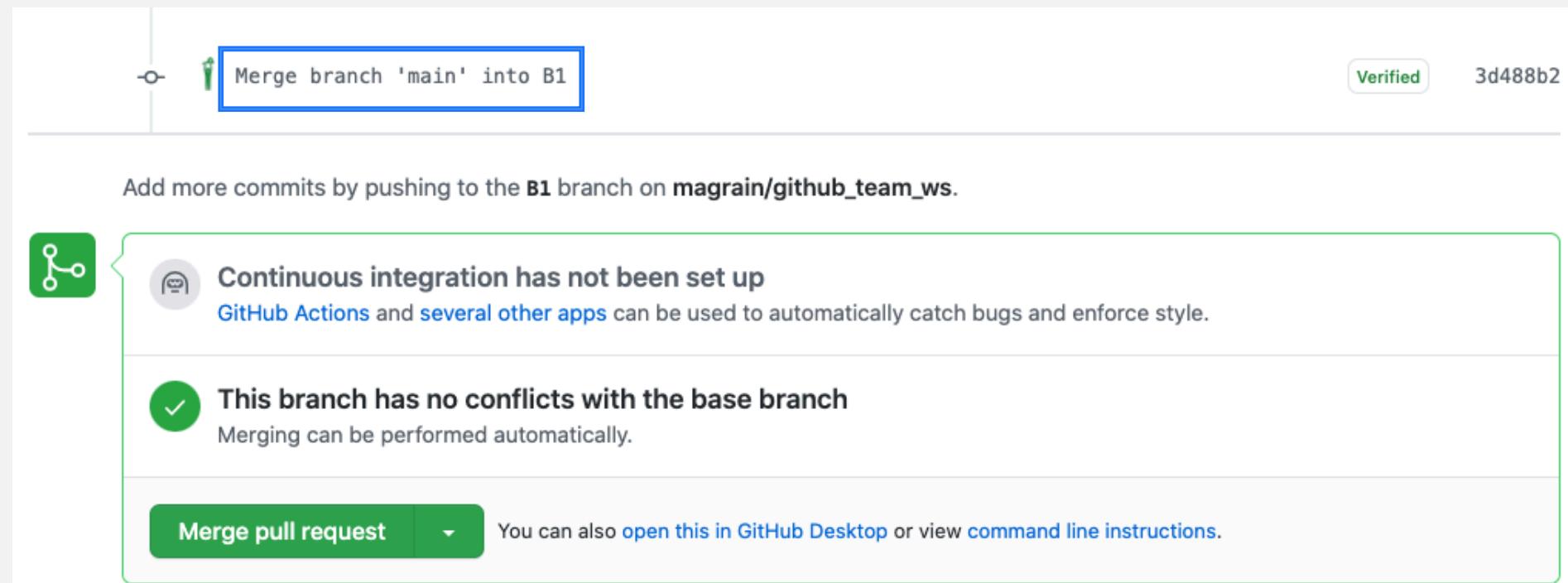


修正後

```
130      /* 演習1 ここから */  
131      function add() {}  
132  
133      function sub() {}  
134  
135      /* 演習1 ここまで */  
136
```

# GitHub での commit

GitHub上で commit した場合、  
commitメッセージは自動で記入されます



# 演習2 実装時の File changed

## 「A2」ブランチ add 関数の実装

```
130      130          /* 演習1 ここから */
131      -      function add() {}
131      +      function add() {
132      +          eqlNum.value = parseFloat(midNum[0].value) + parseFloat(midNum[1].value);
133      +      }
```

## 「B2」ブランチ sub 関数の実装

```
133      133          function sub() {}
133      +      function sub() {
134      +          eqlNum.value = parseFloat(midNum[0].value) - parseFloat(midNum[1].value);
135      +      }
```

# 演習2 での Conflict

改行などがConflictする場合があります

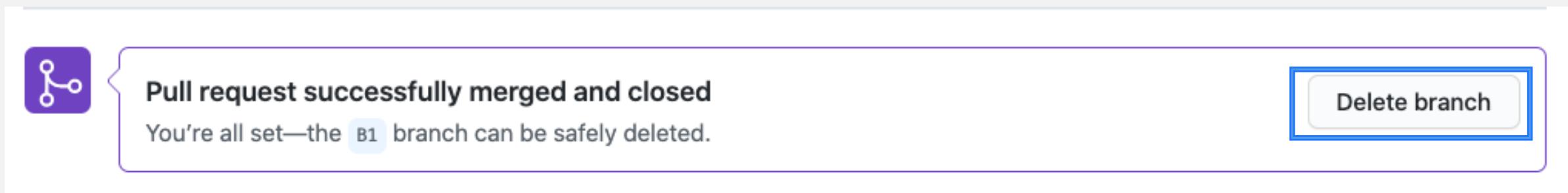
特に気にせずConflictを解消してください

```
---  
139 <<<<<< B2  
140  
141 =====  
142 >>>>>> main
```

# 不要なブランチの削除

使わなくなったブランチは削除が可能です

PR を merge 後に「Delete branch」から、  
リモートブランチの削除できます



おさらい

YAHOO!  
JAPAN

# チーム開発のコード管理

- 複数人で同時にファイルを編集すると Conflict (競合) が発生する場合がある
- ブランチやPRによるレビューに用いて Conflict の解消や想定している実装を行う
- Conflict を起こさないような開発体制をチームで共有する

# HackU に向けて

GitHub はコード管理をアシストするツールです

-> 開発速度を求める際には使わない判断も重要

- コードレビュー
- アイディア出しや議事録をまとめる
- 完成したコードの共有

用途を限定して使うことも考えてみましょう

# GitHub チーム開発編

以上



# おまけ

- **Hack U をフォローしよう**

1. <https://github.com/hackujp> にアクセス
2. hackujp を Follow
3. hackujp/github\_tutorial を Watch & Star

**hackujp の他のリポジトリも確認してみてください**  
※どれも任意です

**最新情報は connpass/Twitter(@hackujp) をフォロー**

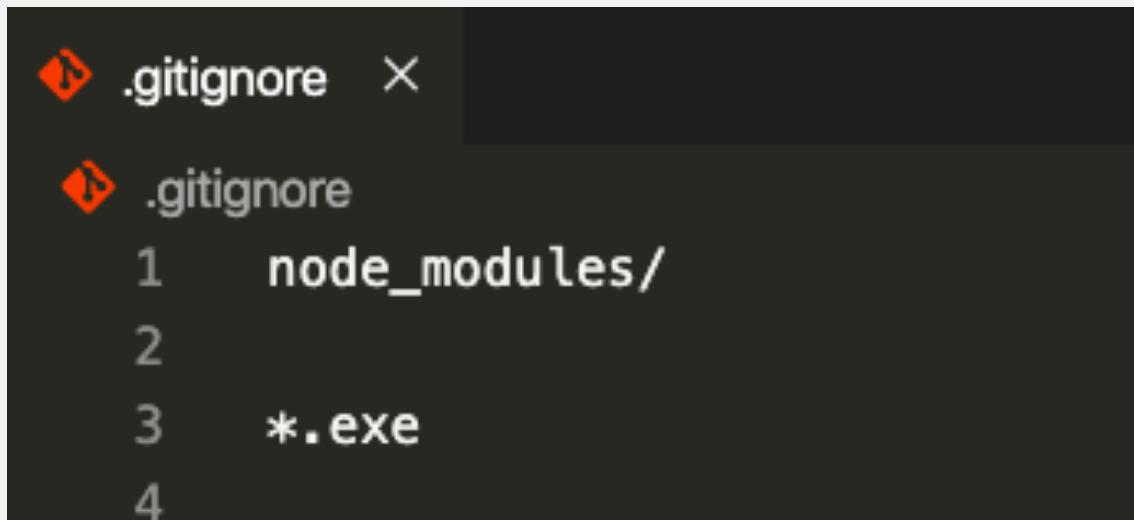
# Appendix



# .gitignore ファイル

- 「.gitignore」ファイルは、Gitの管理に含めないファイルを指定するためのファイルです
- npm における node\_modules ディレクトリや実行ファイルなど、開発時に自動生成されるディレクトリは Git に push する必要がない場合が度々あるため、「.gitignore」ファイルを使うと簡単に管理対象から外すことができます
- 詳しくは調べてみてください

.gitignoreの書き方の例



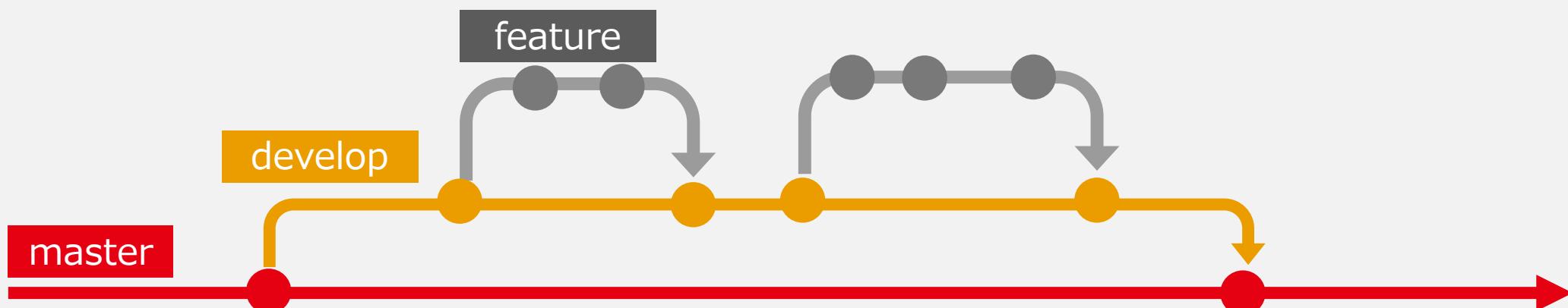
```
① .gitignore ×  
② .gitignore  
1 node_modules/  
2  
3 *.exe  
4
```

# ロールバック

- リポジトリに commit や pushをした後に、間違いなどに気づいた場合、前の状態に戻したいときがあります。
- そのような時に「git reset」や「git revert」などのコマンドを使います
- それぞれのコマンドは場合に応じてオプションを使い分けたりするなど、説明が煩雑になるためここでは省きます
- 詳しい使い方は、各自で調べてみてください。

# ブランチの使い方

- ・ ブランチは、チームや開発する内容によって使い方を変えたほうがよい場合があります。調べてみると面白いと思います。
- ・ しかし一番大事なことは、チーム全員でブランチの使い方にについて認識があっているということです。



# コードレビュー

- Pull Request は、最低1人は他の人にみてもらって Approve をもらってからマージするようにし、  
基本的にはセルフマージ（チェックなしのマージ）は  
しないようにしましょう。
- レビューを行う場合は、コードに対するコメント機能をうま  
く使うと伝わりやすいです。
- コードレビューの行いかたも、チームである程度ルールを決  
めておくとよいかもしれません。