初心者チームワーク全体ミーティングのまとめ

kip2

2022.9.24

目次

1 今日のワークについて

今日の目標

ブランチを切ってマージ、プルリクエストまでを行う。

2 バージョン管理システムについて

VCS(Version Control System) の利点

- 履歴を残せる
- 作業の競合を防ぐことができる

VCS では参照・再現・分岐が可能。そして分岐がメインの機能になる。これが今回のワークの目的。

GitHub の扱いについて

- ルールなく VCS を利用すると上手くいかない。
- 上手くいってないときは動かないものを反映していたり、複数の変更を盛り込んでいる。
- 運用にあたって最低限のルールを守ることが大事。

3 作業 1

1. コマンドで practice リポジトリに移動
2. 自分のブランチを作成
3. 自分のブランチに移動してファイルに変更を加える

```
git switch -c USERNAME
```

4. コミット前に VScode の左の Git マークを押し、何が変更か確認してからコミットする。

```
git add index.html


git commit -m ''third commit''
```

   コメントのコツとして、変更箇所に対してのコメントはあまり意味がない。**なぜ変更したのか**を書くのがよい。

5.
```
git push origin USERNAME
```

6. Pull requests を行う。
   GitHub 画面から行う。
   プルリクエストのタイトルに**誰の**コミットかを書く。

3.1 作業 1 のチームの実習

```
git branch
-- 現在のブランチを把握


git switch -c BRANCHNAME
-- ブランチを作成して移動する


git add index.html
```

```
git commit -m ''COMMENT''
-- ファイルを追加する

git push origin BRANCHNAME
-- ブランチ作成時の名前を指定して push する
```

3.2　マージコンフリクトの解消について

※以下は GitHub 上での操作

1. コンフリクトが発生したプルリクに移動し、[Resolve conflicts] を押す。

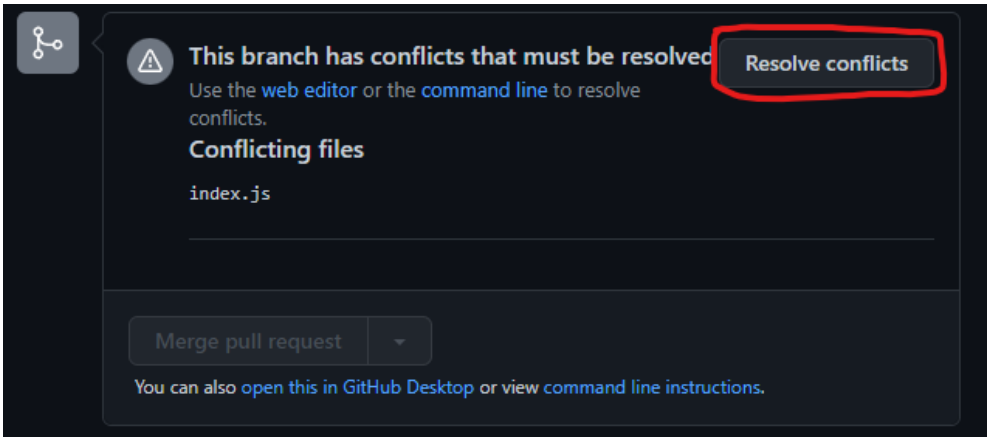


図 1　Resolve conflicts を押す

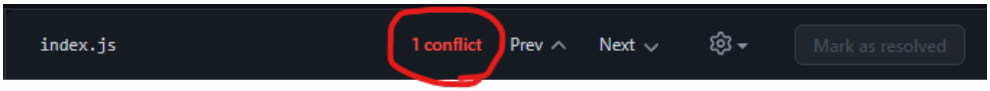


図 2　1conflict と表示されてるのが確認できる

2. コードの該当箇所は以下の形式で表示される

```
<<<< branch\_name
    （user が変更した内容）
======
    （元々の内容）
>>>> develop
```

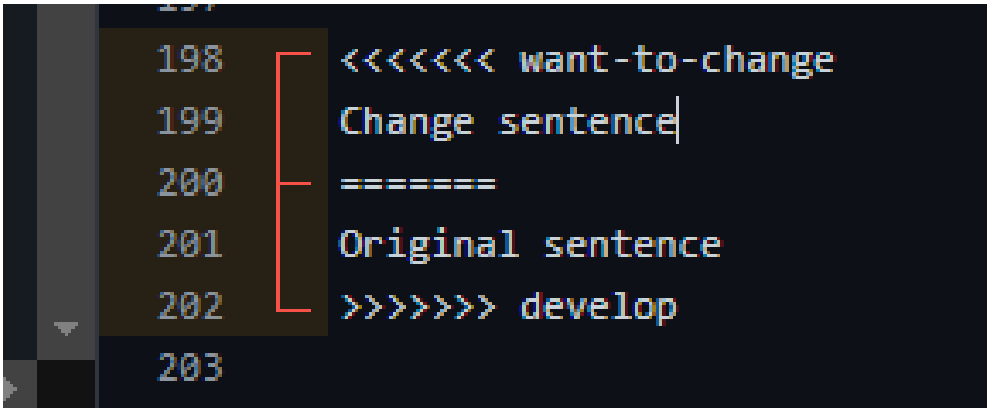


図 3　コンフリクトが起こった箇所

3. 残したい方を残して、もう一方を消去する。

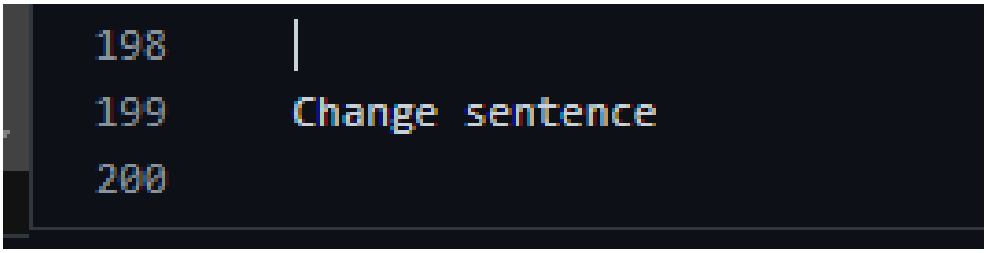
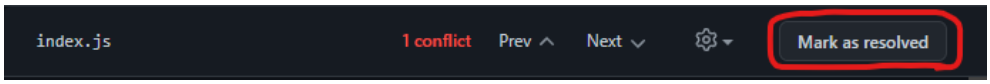


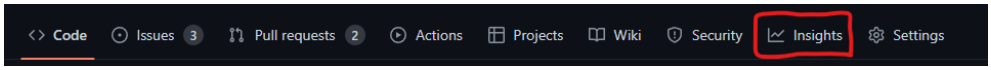
図 4　コンフリクトが消えた

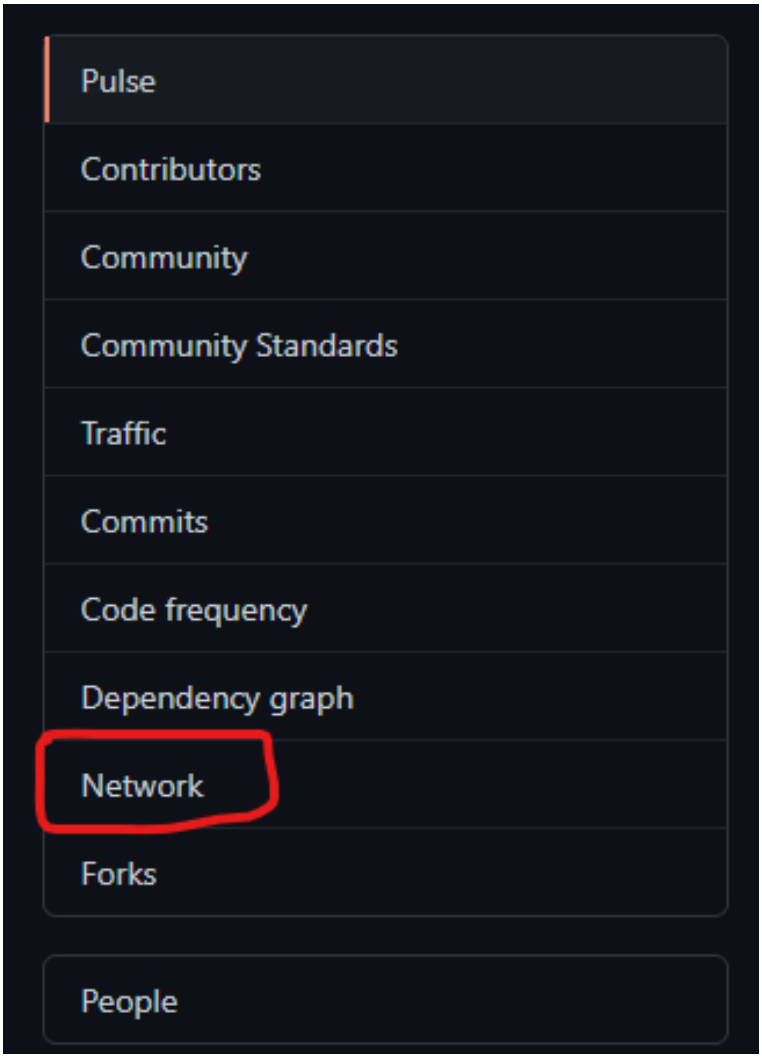
4.「Mark as resolved」ボタンを押す



3.3　全体のブランチの確認方法

インサイトのネットワークから確認できる




Pulse
Contributors
Community
Community Standards
Traffic
Commits
Code frequency
Dependency graph
Network
Forks
People

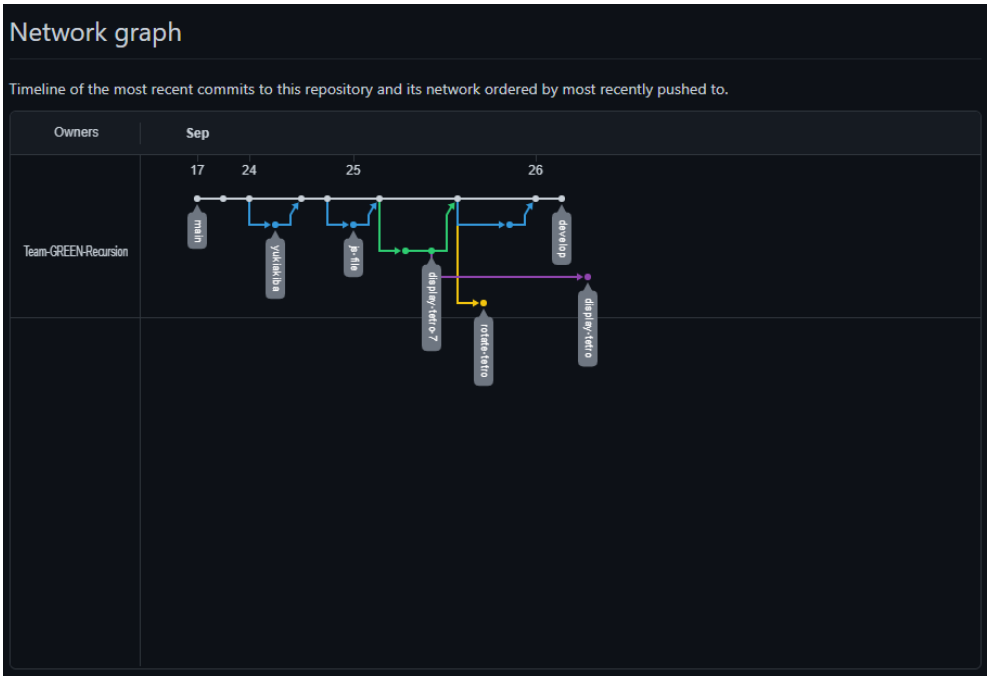


図 5　現在の状態がグラフの形で確認できる

何か変更をしたい場合は develop の最新をさらに変更する必要がある。

4　issue の作り方

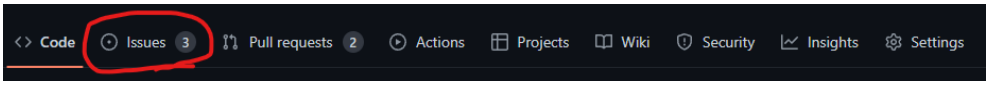


図 6　Issues をクリック

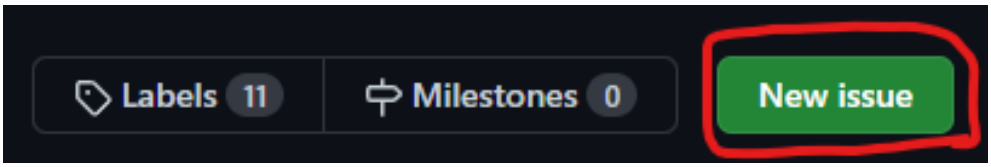


図 7　New issue をクリック

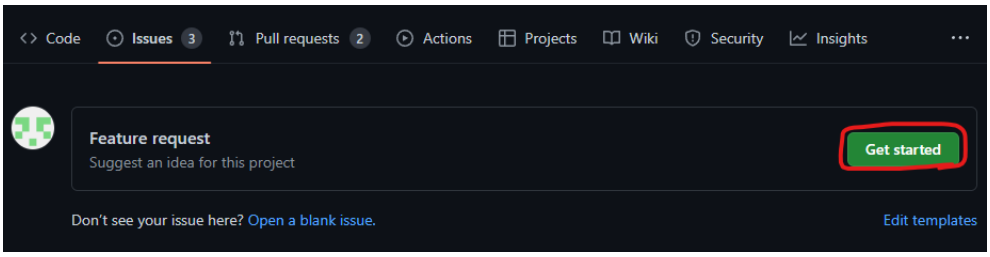


図 8　Get started をクリック

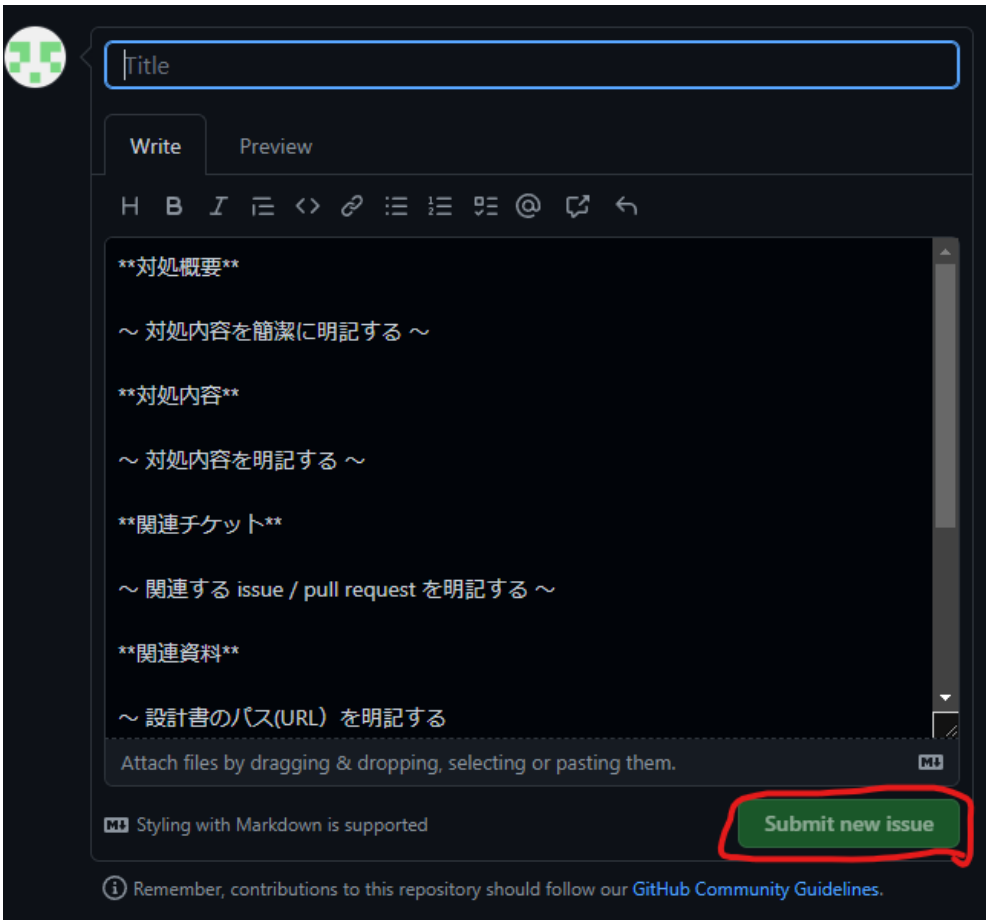


図 9　内容を記述し、Submit new issue をクリック

Title に feat: を書いて issue を作る
Write の中に内容を書く。どういった機能を入れたかなど
チェックリストを作成する。
Assignees で相手を選択する。作業をやってほしい相手を指定する。指定は基本的に一人が良い。
ラベルをつけることができる。優先順位のラベルなどを作れる。ラベルは自分で作成することができる。

Create new labels で color などを選んで作成する。

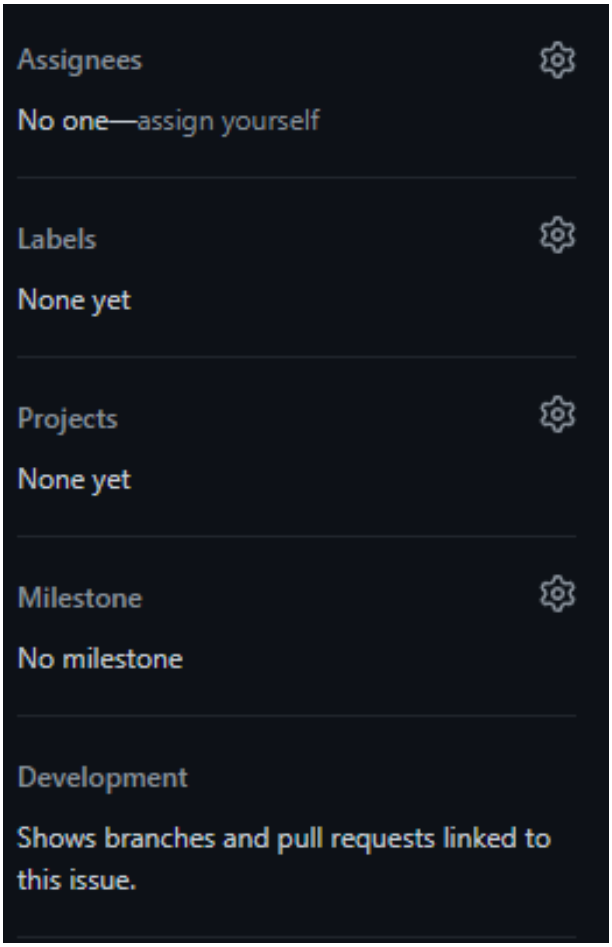


図 10　Issue 作成画面右側にある

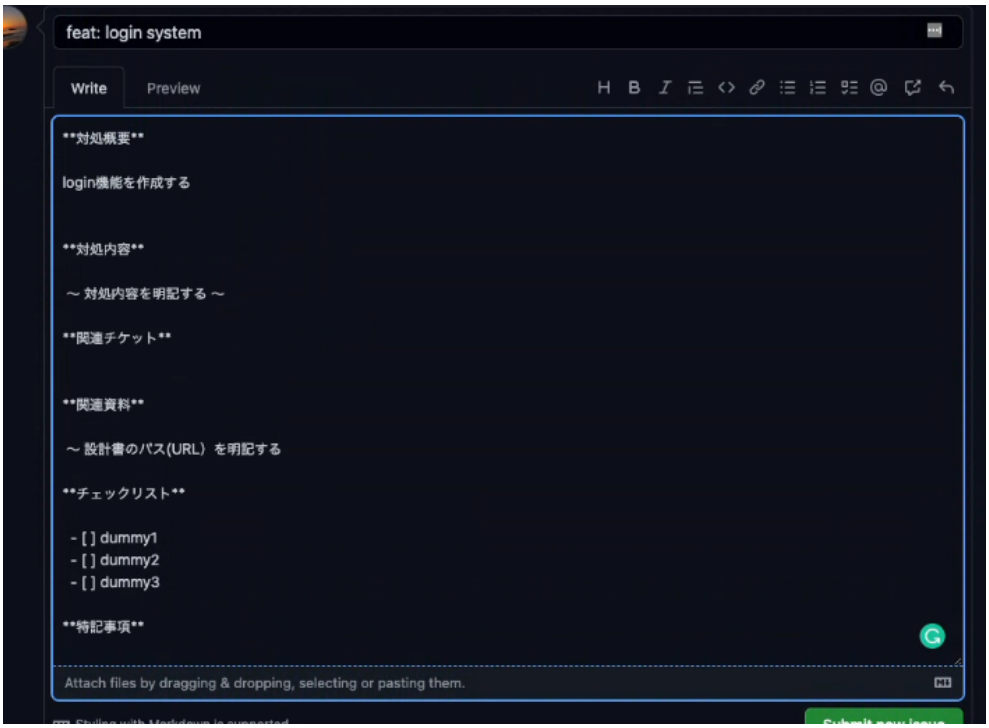


図 11　Issue のテンプレート一例

feat などの補足

補足　以下の URL が参考になる。

`https://qiita.com/numanomanu/items/45dd285b286a1f7280ed`

プレフィックスという考え方

コミットメッセージの先頭に何らかの文字をつけること

feat:xxx という機能を追加

fix:yyy で発生するバグを修正

refactor:zzz の機能をリファクタ

4.1　プルリクエストのレビューのやり方

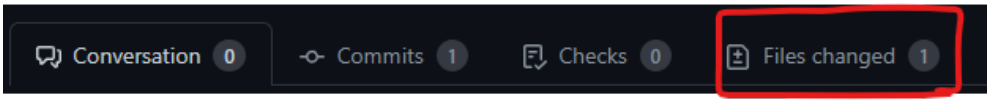


図 12　Files changed を押す

+を押すと 1 行を選択。

Shift して選択していくと複数行

選択した状態で下でコメントを書く。

Finish your review を押す。

Submit review

すると、さきほど選択した行に対して行ったコメントが Conversation の画面に出てくる。

pull requests と issues を結びつける。

ただ pull request をしてはいけない。プルリクになんの issues を解決したかを書くこと。issues はハッシュタグで書ける (#)。補完もできる。issue の URL でも指定ができる。

New issues ボタンで新しい issues が作れる。

アサインとラベルを見る。

図 13　プルリク作成時のテンプレート例

4.2　issues のチーム実習

省略

4.3　issues の意味・使い方

issue はミッションの内容を当てはめるうことができる。実現したいことを小分けに issue にして、それをチームで共有して作業することで、作業が被ることがない。それが issue を使う意味。

チーム同士で確認ができるのがメリット。

issue をプルリクで誰かがやってくれる。

Assignees をすると、相手を指定することができるので、余分なコミュニケーションコストを削減できる。

編集時の追記

以下推測。
issue を相手を指定しないでどんどん出しておいて、やれる人が issue を解決する。というのが実際の OSS 開発で行われることかもしれない。つまり、issue には 2 つの利用方法があることになる。

1. 相手を指定して issue を発行。指定された者が解決する。
2. 相手を指定せず issue を発行。だれかできそうな人間が解決を行う。