OSS 開発という視点から見た Godot・GitHub との付き合い方

Silc Lizard (Tokage) Renew

in Godot 勉強会 #7

1. Godot Engine という組織

自己紹介

Godot Core Contributor (Member)

Silc Lizard (Tokage) Renew

Godot Engine の
アニメーションチーム※に所属してます

レビュー可能範囲：
Animation / 3D Asset-pipeline / 3D Editor

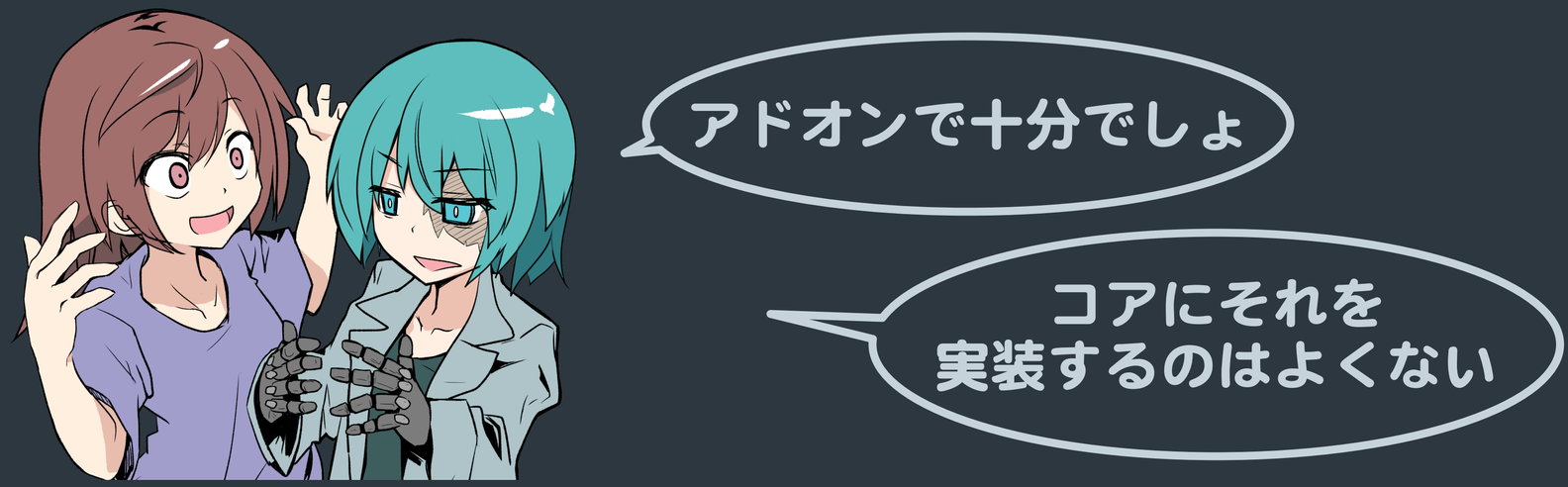
TokageITLab

※ 参考：https://godotengine.org/teams/

Godot Core Contributor とは

GitHub の godotengine/godot リポジトリの
master ブランチにコミット※ した事がある人

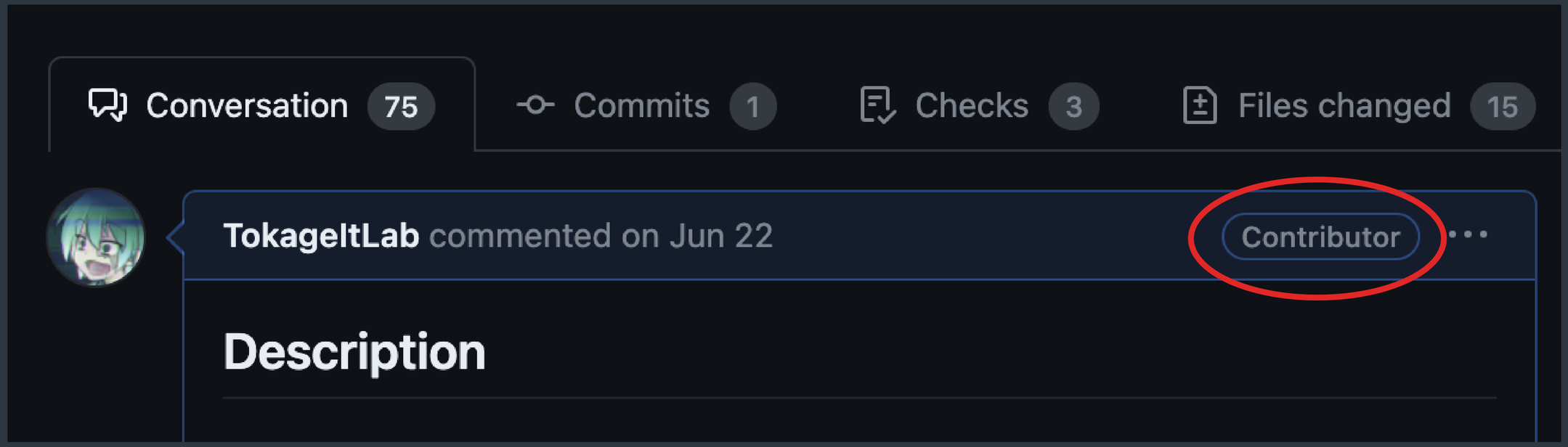
コア＝Godot Engine そのもの
コアに実装する＝Godot Engine の次のバージョン以降にそれが含まれる



※ Pull Request がマージされるだけでなく直接の Push 等も含まれるが、基本的にその権限はない

Godot Core Contributor とは

一度でもコミットする（Pull Request がマージされる）と
当該リポジトリのスレッドなどで名前の横に Contributor バッジが付く



Godot Core Contributor とは

一応、10 コミット以上でクレジットされるという目安がある

Godot Engine authors

Godot Engine is developed by a community of voluntary contributors who contribute code, bug reports, documentation, artwork, support, etc.

It is impossible to list them all; nevertheless, this file aims at listing the developers who contributed significant patches to this MIT licensed source code. "Significant" is arbitrarily decided, but should be fair :)

GitHub usernames are indicated in parentheses, or as sole entry when no other name is available.

Project Founders

Juan Linietsky (reduz)
Ariel Manzur (punto-)

Lead Developer

Juan Linietsky (reduz)

Project Manager

Rémi Verschelde (akien-mga)

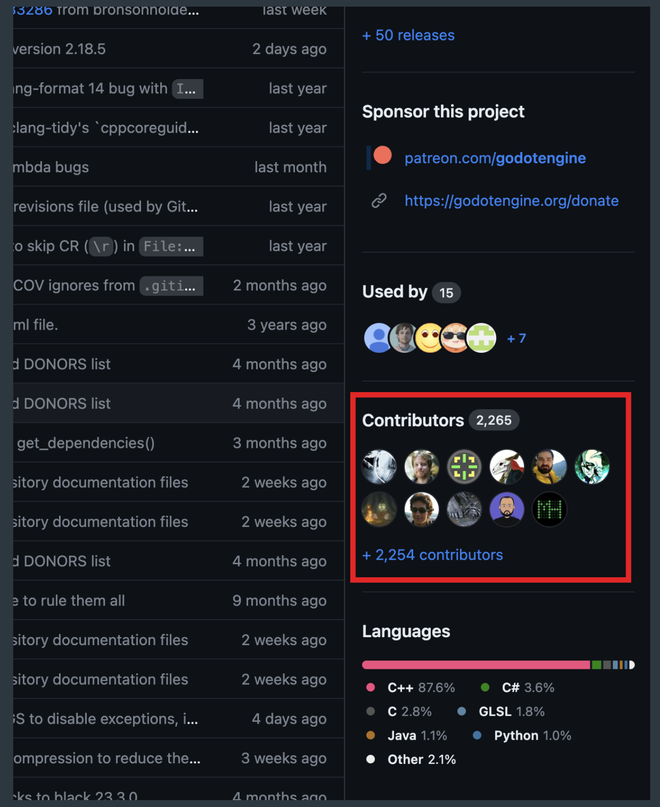
Developers

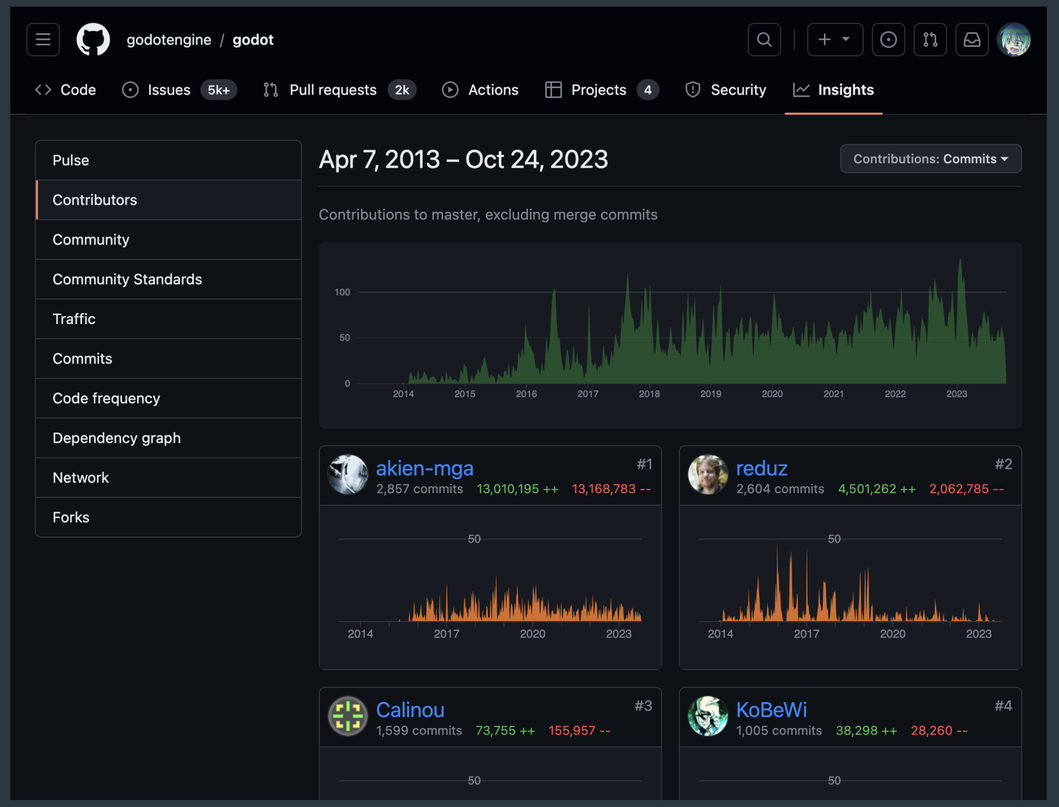
(in alphabetical order, with over 10 commits excluding merges)

with over 10 commits excluding merges)

Godot Core Contributor とは

更にコミット数上位者※はリポジトリのランキングに載る





※ 追加・削除行数ランキングもあるが、まずコミット数で 100 件フィルタリングされた後にソートされる

Member とは

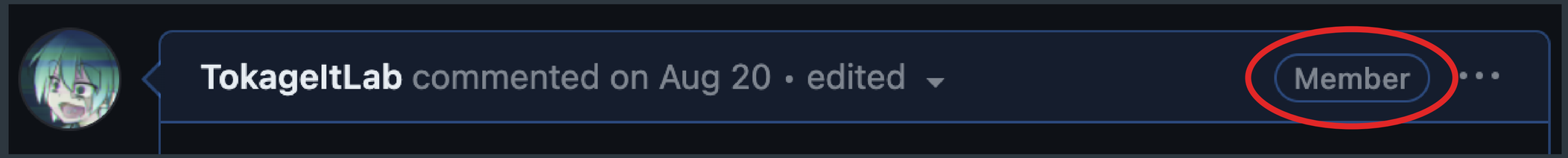
継続的に Pull Request を送り、言動に問題などが無ければ
Member すなわちコラボレーターとしてスカウトされる可能性がある

Godot Engine 所有のリポジトリでの権限が強くなる

- Issue や Pull Request にラベルを付けられる
- 他人の Issue や Pull Request を閉じることができる
  → 重複してる Issue や Pull Request を整理するため
- Pull Request を Approve してマージ可能な状態にできる
  → マージするかどうかの判断はもっと偉い人がする
  → 勝手にマージしていい訳ではない

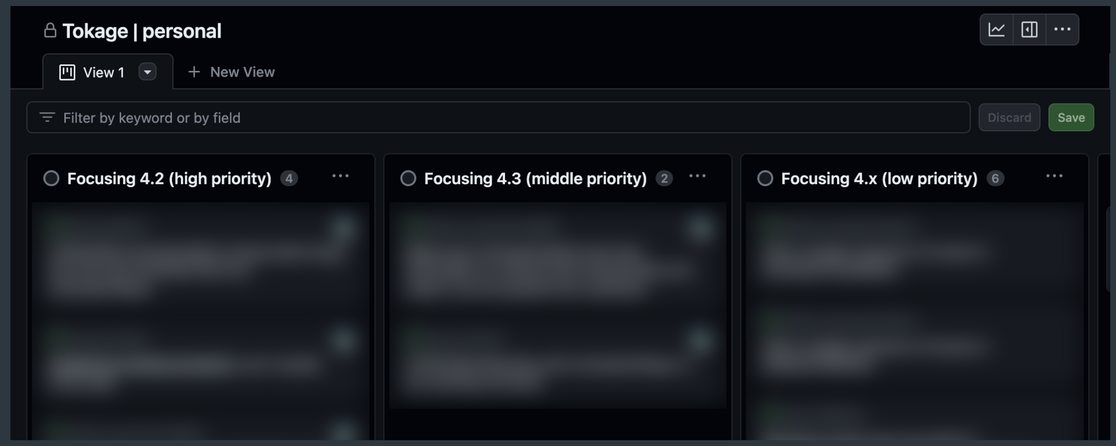
Member とは

Contributor ではなく Member バッジ※になる



GitHub で Organization のプロジェクト管理機能が使えるようになる

- タスク管理
- ロードマップの編集



※ GitHub 上の Organization の Members の設定で外部向けに表示するかどうか自分で決められる

Member とは

GDC や内部カンファレンスのようなイベントに
Godot Engine のメンバーとして参加できる

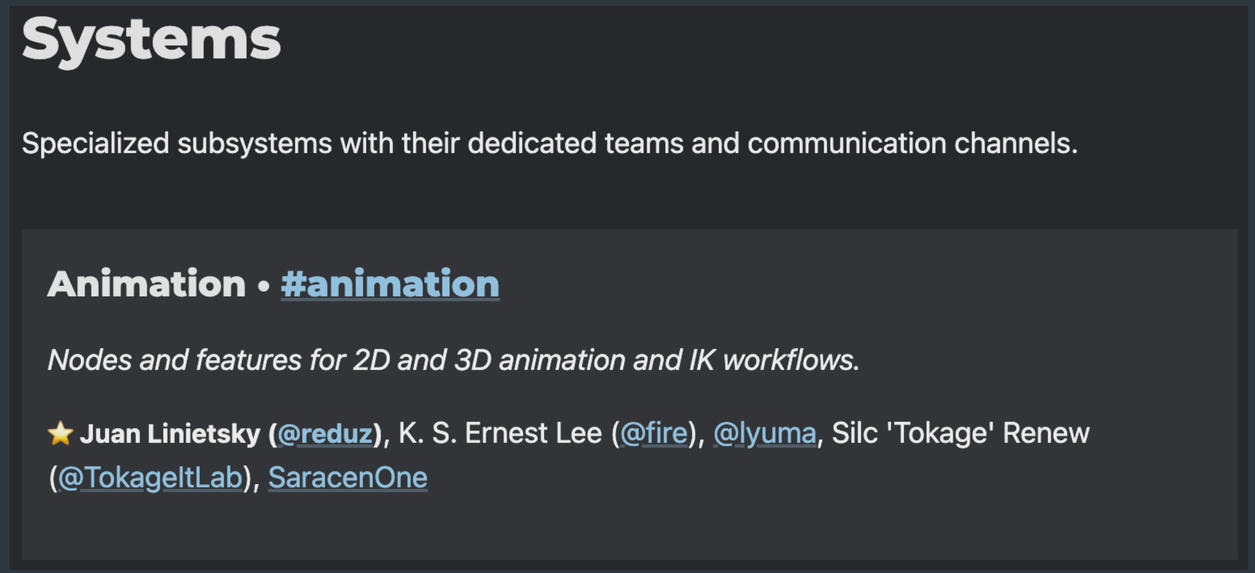
https://twitter.com/reduzio/status/1638699964867170305


https://twitter.com/reduzio/status/1532306676719697921

Godot 開発チーム

機能ごとにチームが存在し、メンバーが割り振られている

チームが存在する機能に対する Pull Request は、
当該チームのメンバーが Approve すると
星付きのチームリーダーかプロジェクトマネージャーによってマージされる



Systems

Specialized subsystems with their dedicated teams and communication channels.

Animation • #animation

Nodes and features for 2D and 3D animation and IK workflows.

⭐ **Juan Linietsky (@reduz)**, K. S. Ernest Lee (@fire), @lyuma, Silc 'Tokage' Renew (@TokageItLab), SaracenOne

Godot 開発チーム

基本的にメンバーはお金で雇われていない

- 常に増減したりアクティブ・非アクティブになったりする
- チームに割り振られていないメンバーも存在する
  → 未所属のメンバーの Approve により PR がマージされる事もある

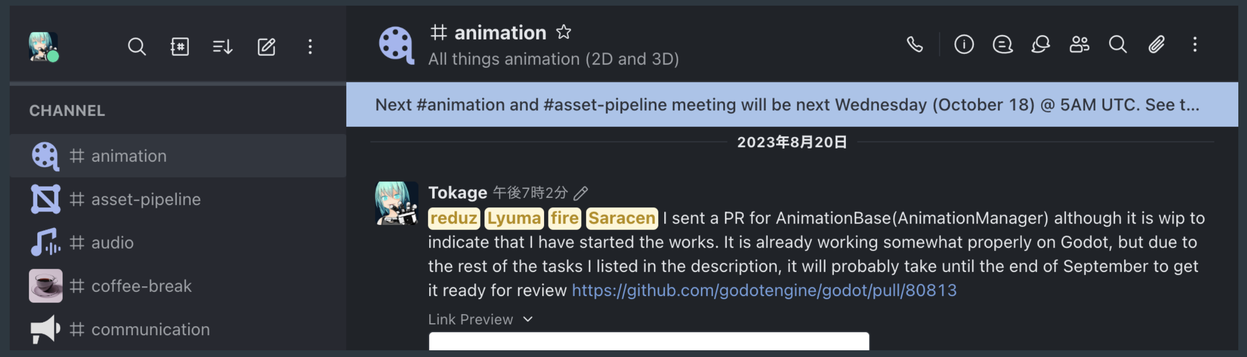
誰が何をしているかの最新情報は
Contributors Chat（Rocket Chat）※で手にいれよう

※ 参考：https://chat.godotengine.org/

Contributors Chat

コントリビューター同士がやりとりするためのチャット

- ホスト可能な IRC ライクでログが残せるチャットとして採用
- Issue や PR の質問をするとコントリビューターが答えてくれるかも…
- ゲームの作り方や実装方法※を質問する場所ではないので注意
  → Reddit や Discord にいこう



※ Godot Engine の仕様上の制約で困っているというようなケースでの質問は OK

2. Godot Engine との付き合い方

OSS を過信してはいけない

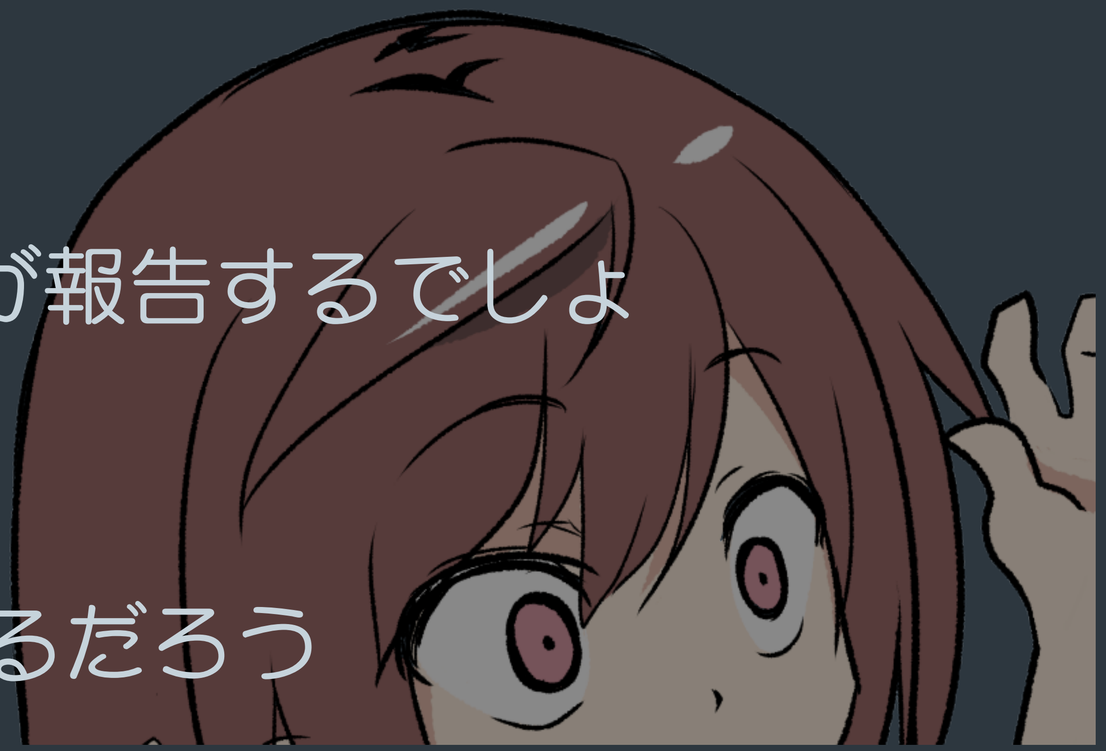
こんなとき…

・コアの仕様上の制約により実装ができない
・バグに遭遇した
・フリーズ・クラッシュした

誰かが報告するでしょ

SNS で愚痴ろう

放っておけばいつか直るだろう



OSS を過信してはいけない

ゲーム開発者としてそれでいいのか？

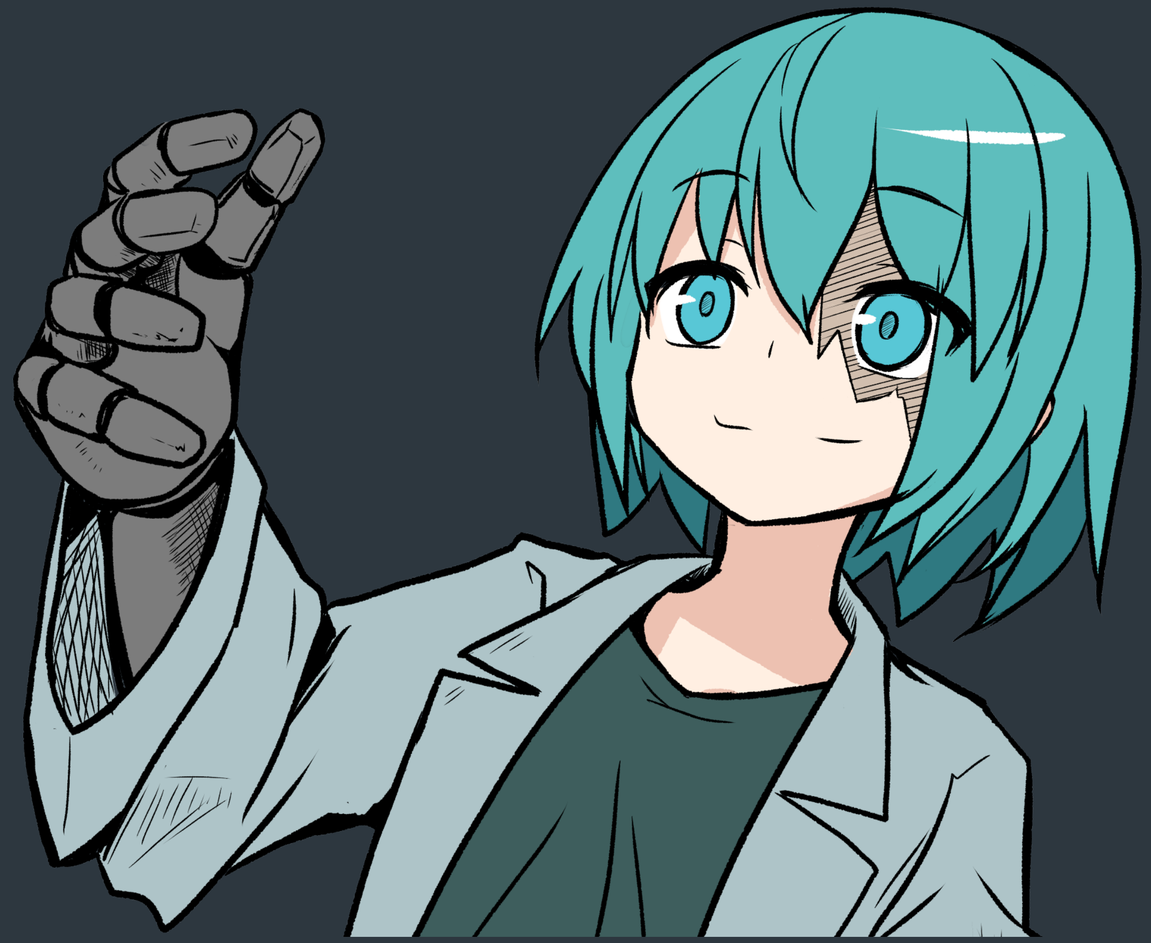
OSS に何も支払わずにタダ乗りしていないか？

Godot Engine はオープンソース故に
ソースコードが公開されていて
自分でビルドする事が十分に可能である

ゲームの企画者(プランナー)や設計者(デザイナー)ではなく開発者(デベロッパー)を名乗るなら
自力で解決できるように手を動かすべきではないのか？

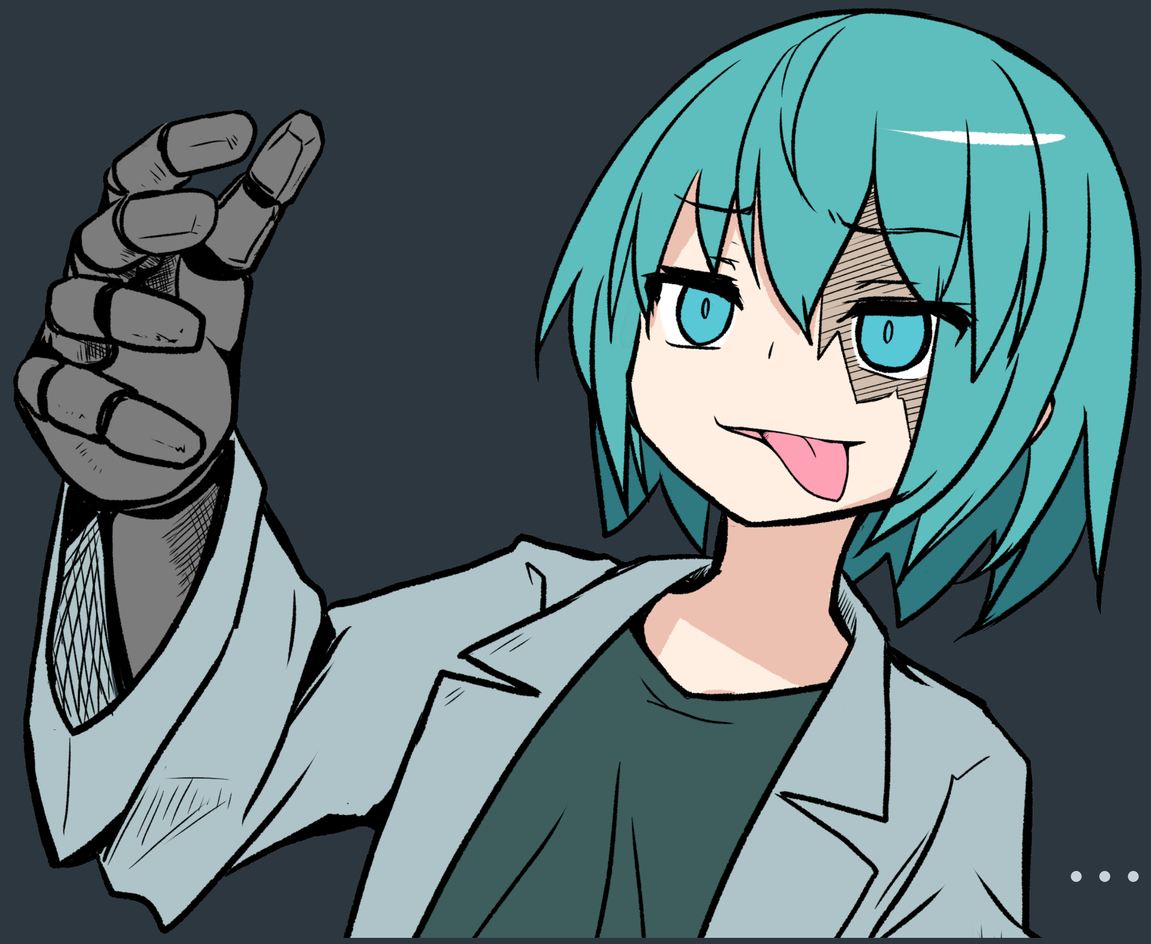
なぜコントリビュートするのか

コアにマージされればそれが万人のためになります！



なぜコントリビュートするのか

コアにマージされればそれが万人のためになります！



…というのは建前

なぜコントリビュートするのか

forkして独自バージョンのGodot Engineを作ることもできる

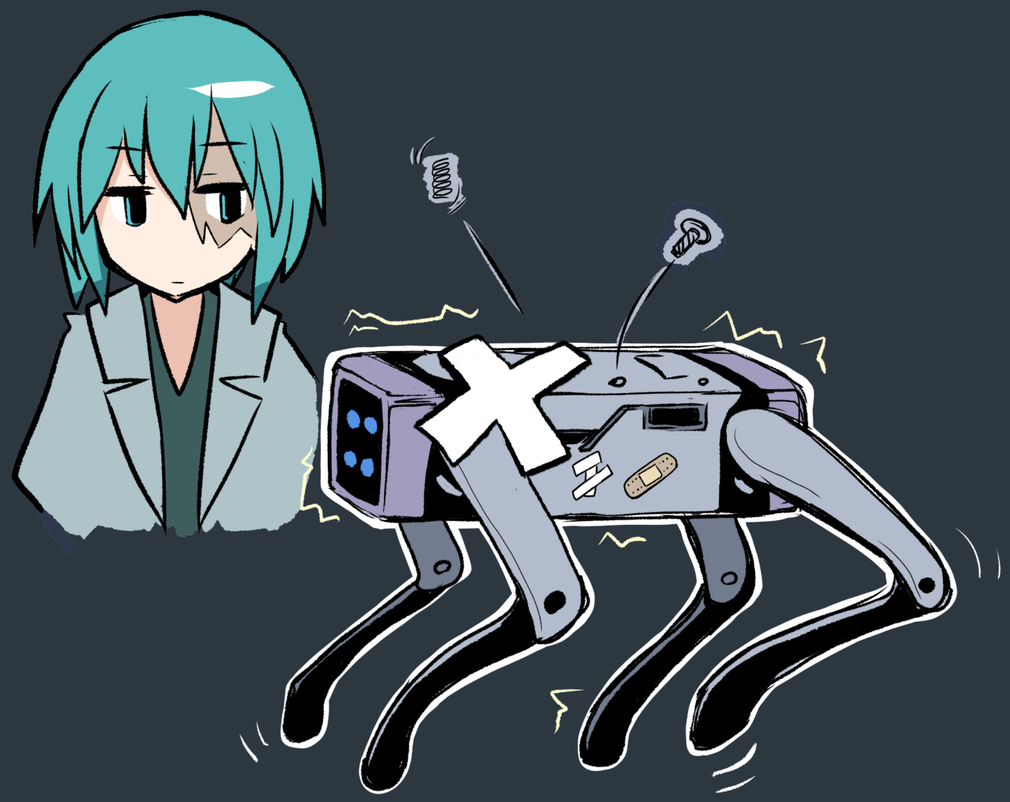
アドオンや少しHackyな実装で解決できる問題もある

しかし…

独自の回避策は将来のGodot Engineの
アップデートにより無意味になったり壊れたりする
可能性がある！

なぜコントリビュートするのか

独自の回避策は対症療法のようなもの



きちんと根本的な治療（コアの修正）を行うべし

なぜコントリビュートするのか

もしマージされれば…

次のアップデートに修正が含まれる
↓
誰かがその機能に触れる
↓
フィードバックやバグ報告が上がる
↓
誰かが更に改善や修正をしてくれる（かも）

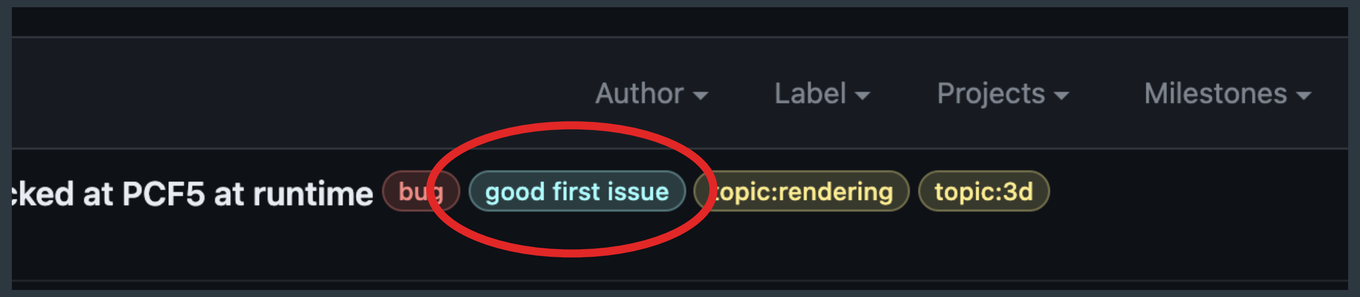
他人をアテにするならまずは自分から手を動かそう

3. Pull Request マナー講座

コントリビューターを目指す人へ

コントリビューターになりたいけど、
特に実装したいものや直したいバグがない…

メンバーの手が空いてなかったり面倒だったりする時、
新規コントリビューターのための入り口として
バグの原因が単純で修正が簡単な Issue に付けられるラベル



good first issue ラベル※を探してみよう

※ 参考：https://github.com/godotengine/godot/issues?q=is%3Aissue+is%3Aopen+label%3A%22good+first+issue%22

コントリビューターを目指す人へ

身につける必要があるスキル

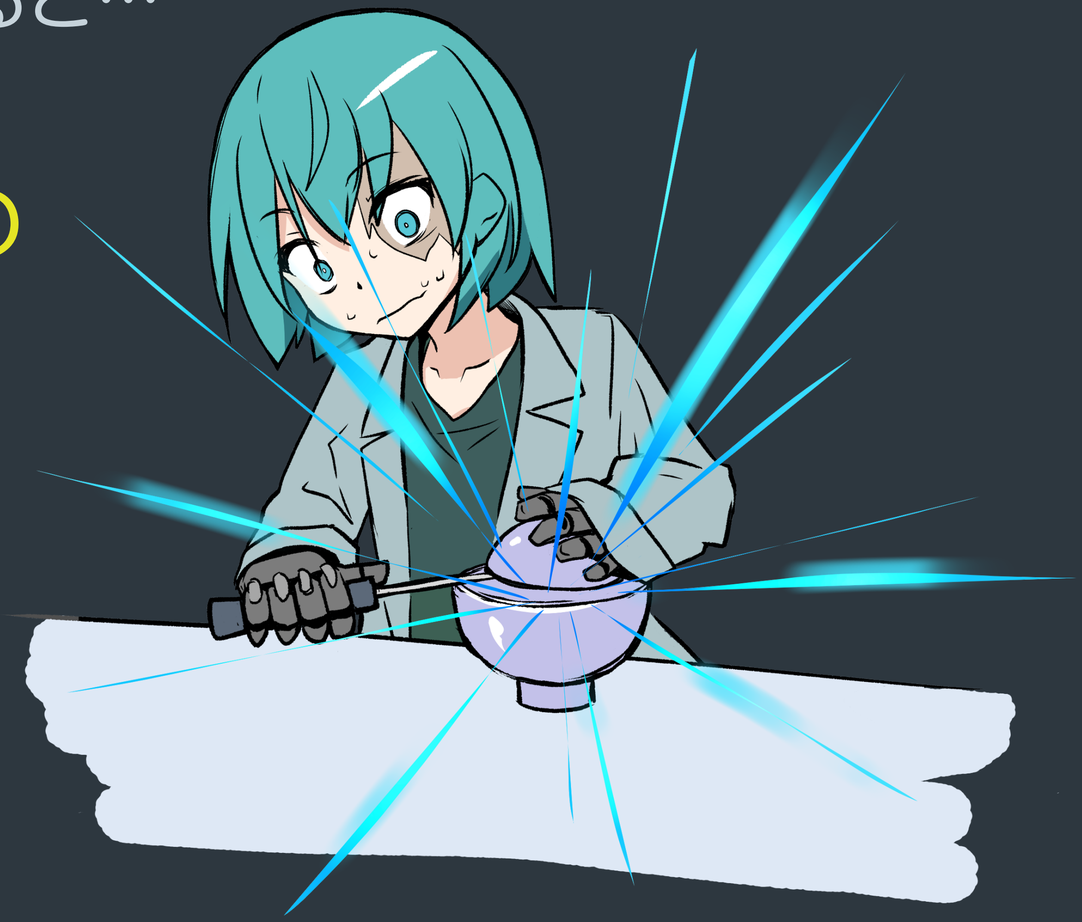
- 開発環境の構築※に慣れよう
  - → scoop や brew といったパッケージマネージャの導入
- C++ の書き方をある程度知っておこう
  - → C++ は色々な書き方ができてしまう（C++ の闇）
  - → コーディングスタイルを揃えよう（Godot Engine なりの書き方がある）
- Git の仕組みや使い方を少し詳しく知っておこう
  - → OSS コントリビュートにおいてはかなり重要
  - → 適当にやると他人に迷惑をかける可能性がある

※ 参考：https://docs.godotengine.org/en/latest/contributing/development/compiling/index.html

なぜマナーが必要なのか

例えば、Pull Request の手順を間違えると…

通知メールが Godot Engine の
コントリビューター全員に
飛ぶ可能性があります※



Nuke（核爆撃）と呼ばれることも…

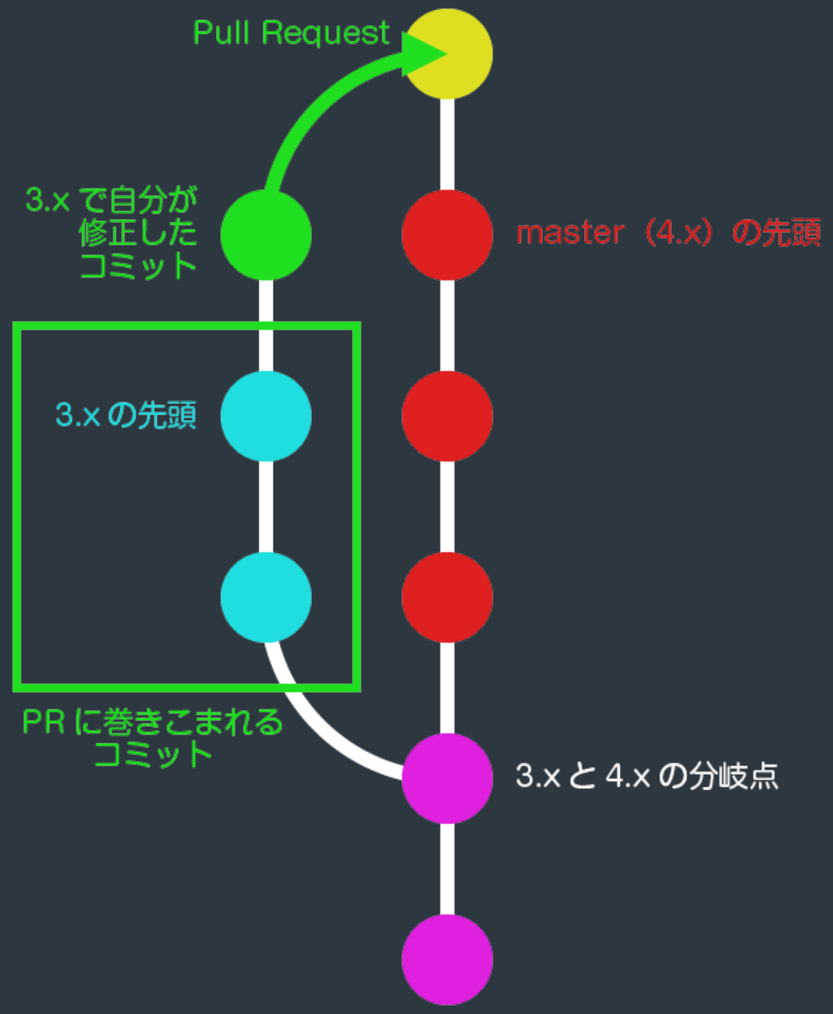
※ 受信者の設定次第ではあるが、大抵の場合はデフォルトで通知されるようになっている

なぜマナーが必要なのか

禁止されているわけではないが、

Pull Request を送るなら
間違った Git の使い方をしないように
気をつけよう

× 間違ったブランチに Pull Request を送る



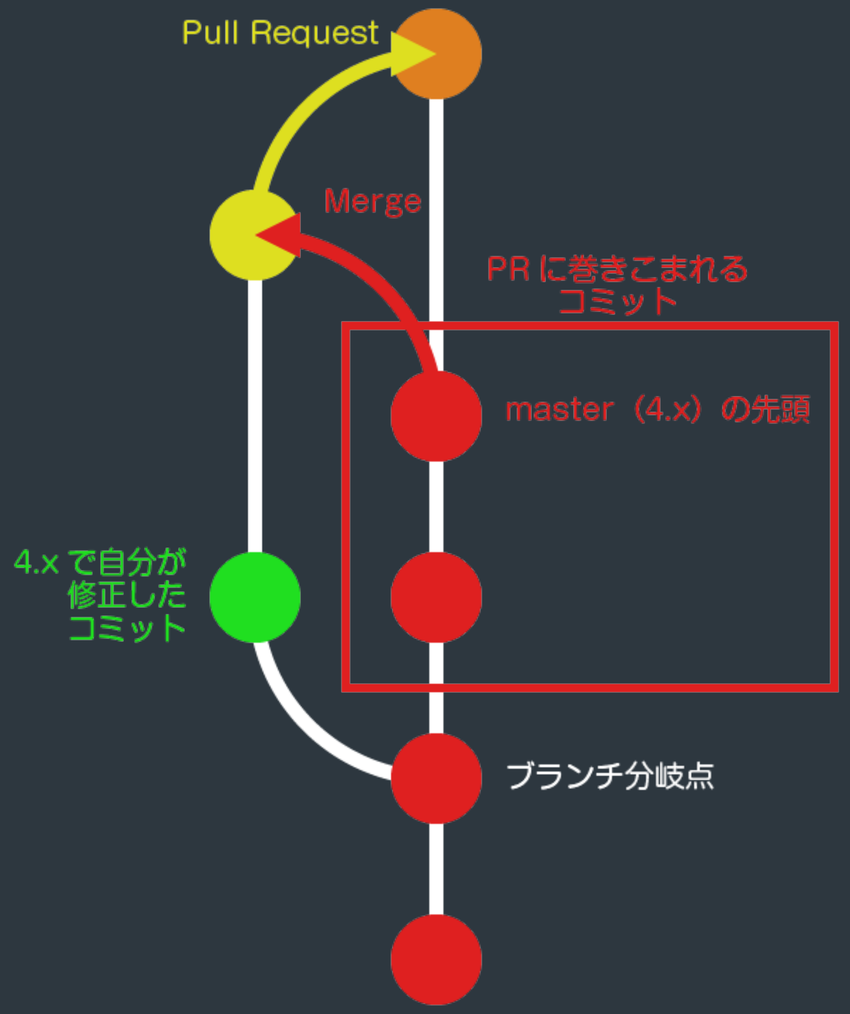
Godot Engine は 3.x と master（4.x）で
開発ブランチを分けている

3.x ブランチの修正を master に
Pull Request として送ると…

3.x と 4.x の分岐点から PR までの
3.x ブランチの全コミットが
自分の PR に含まれてしまう

PR に巻きこまれたコミットの著者
全員に通知メールが飛びます

× コンフリクト解消のために master を merge する



自分のブランチで作業していると
時折 master とのコンフリクトが発生する

コンフリクトの解消のために master を
自分のブランチにマージすると…

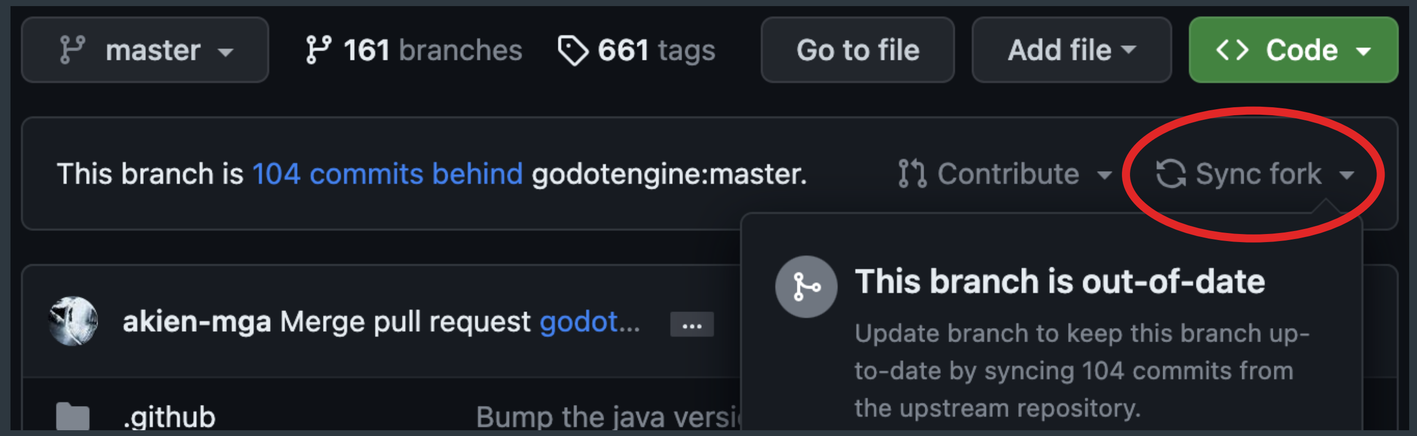
自分のブランチと master の分岐点以降の
master の全コミットが
自分の PR に含まれてしまう

PR に巻きこまれたコミットの著者
全員に通知メールが飛びます

× コンフリクト解消のために master を merge する

Pull Request を送るためには Godot Engine を fork する必要があるが、
fork したリポジトリは本家（upstream）と同期されていない

GitHub 上には Sync fork 機能があるが、
これを利用するとマージによって同期しようとするので
使ってはいけない※



※ 少なくとも Godot Engine を開発する場合においては

× コンフリクト解消のために master を merge する

Godot Engine 的に正しいのは…

・同期
　→ Fork に本家のブランチを追加して手動で同期する※1
・コンフリクト解消
　→ interactive rebase と force push を使う※2

※1 参考：https://zenn.dev/tetsu_koba/articles/77347ef2c283ac
※2 参考：https://docs.godotengine.org/en/latest/contributing/workflow/pr_workflow.html#the-interactive-rebase

コミットは1つにまとめよう

ブランチが複雑にならないように
原則として１つの Pull Request につきコミットは１つしか認められない

squash（fixup）と force push でコミットを１つに押し潰そう※

force push を正しく使いこなそう

※ 参考：https://chaika.hatenablog.com/entry/2019/02/25/170000

Pull Request の前に Issue や Proposal を送ろう

Issue がないのに PR を送られるとメンバーの人が優先度の判断に困る

- バグ修正の Pull Request
  - → godotengine/godot の Issue が必要
  - → バグ修正は優先度が高い
  - → Patch Update（ex. 4.2.1 → 4.2.2）に含めることが可能※
- 機能改善・新規実装の Pull Request
  - → godotengine/godot-proposals の Issue が必要
  - → Minor Update（ex. 4.2 stable → 4.3 beta）にしか含められない
  - → stable が出た直後の alpha 時期にレビュー要求すると通りやすい

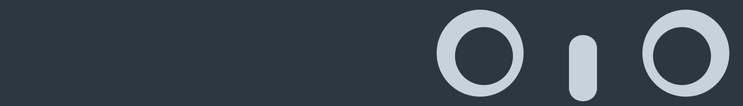
※ beta 以降は互換性の破壊がないことが前提

ディベートではなくディスカッションに強くなろう

厳しいレビューや否定的なコメントがくるかもしれないが
それらに負けない強い心を持とう

・技術的な問題についてはとことん話し合って妥協点を見つけよう
　→ ディベートと違って折り合いをつける事が重要

・ただし、反発心を持ったり反抗的な態度をとったりしないようにしよう
　→ 意外と Godot Engine の中の人たちはユーザーの言動を見ている…



4. 終わりに

コントリビュートして得られるもの

「Godot Engine は Godot Engine で構成されている」の本当の意味…

C++ を使う必要はあるが、
Node・Resource・Signal という概念は GDScript での開発と変わらない
Godot Engine でゲームを作る時と同じ感覚でコアの開発ができる

つまり、

Godot Engine のコアを開発している内に
Godot Engine の使い方・ゲームの作り方に詳しくなれる

OSS との付き合い方

アップデートを蔑ろにするな
alpha/beta のテストも蔑ろにするな
Issue を送れ
Pull Request を送れ
出来る範囲で構わないから

OSS は何もしていないと壊れます

Godot Engine 案件

絶賛大募集中！

ご静聴ありがとうございました