Rubyエコシステム 開発入門 ワークショップ

アイスブレイク：チャットに入ろう！

- ruby-jp Slack #oss_gate
  https://ruby-jp.github.io/
  - 最初に↑の説明を読もう！
  - すでに入っている人はまわりの人を助けよう！
- 今後もみんなと助け合える場所！
- 入ったら挨拶しよう！
- 終わったら今日の成果を自慢しよう！

目的

- ✓ Rubyエコシステムの健全性をキープ
  - ✓ 今が悪いというわけではない
- ✓ 健全？
  - ✓ たくさんのユーザー・それなりの新規開発者
  - ✓ Ruby・gem開発者を増やしたい！

内容

- ✓ 🤦開発ノウハウを伝授
  - ✓ こう実装するといいよとか
- ✓ 🤦開発を体験
  - ✓ 初めてのバグレポート・機能提案とか
  - ✓ 初めてのバグ修正とか

開発未経験の理由

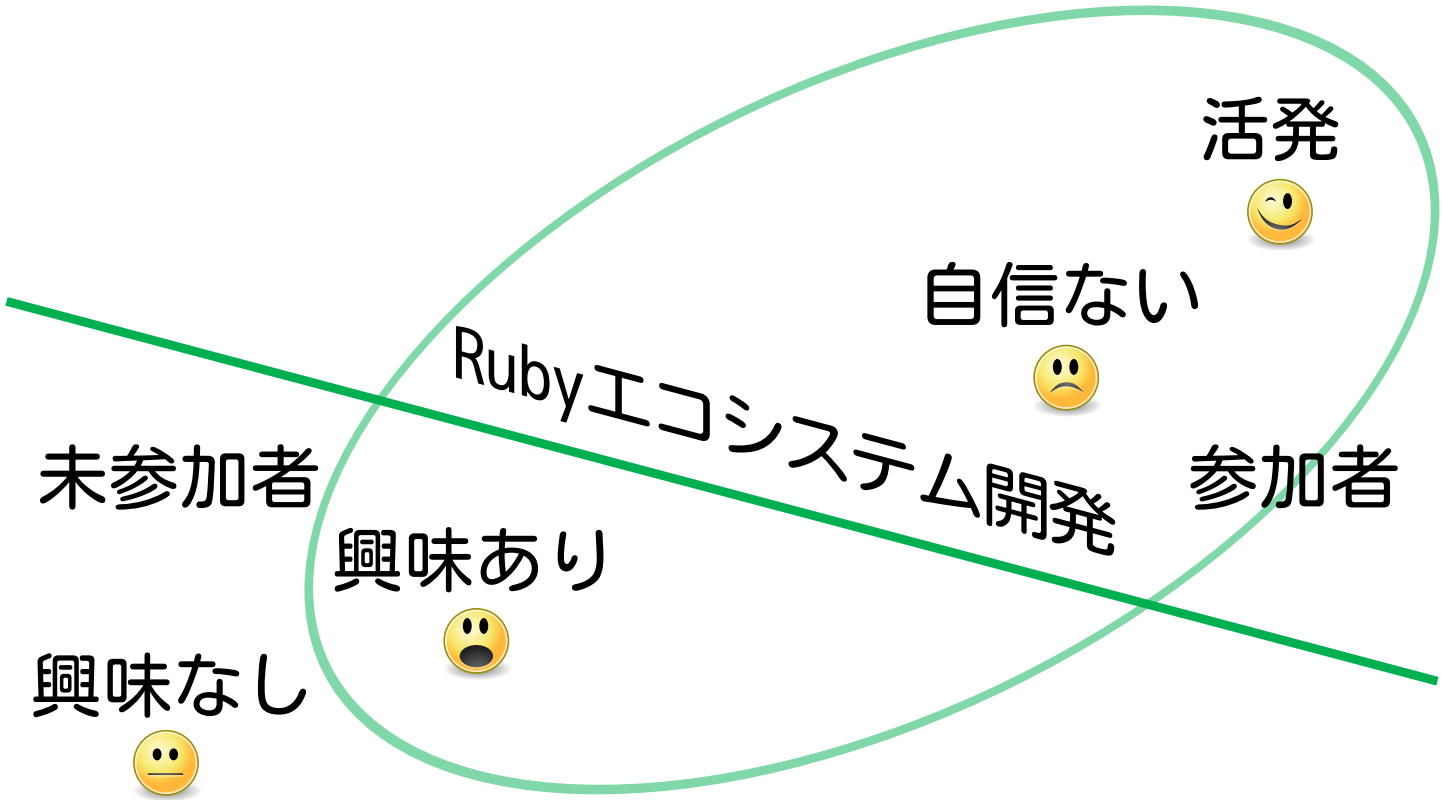
- （数人の参加者に聞く）
- 予想：
  - やったことがないから
    なんとなく敷居が高いと感じる

このワークショップでの「入門」

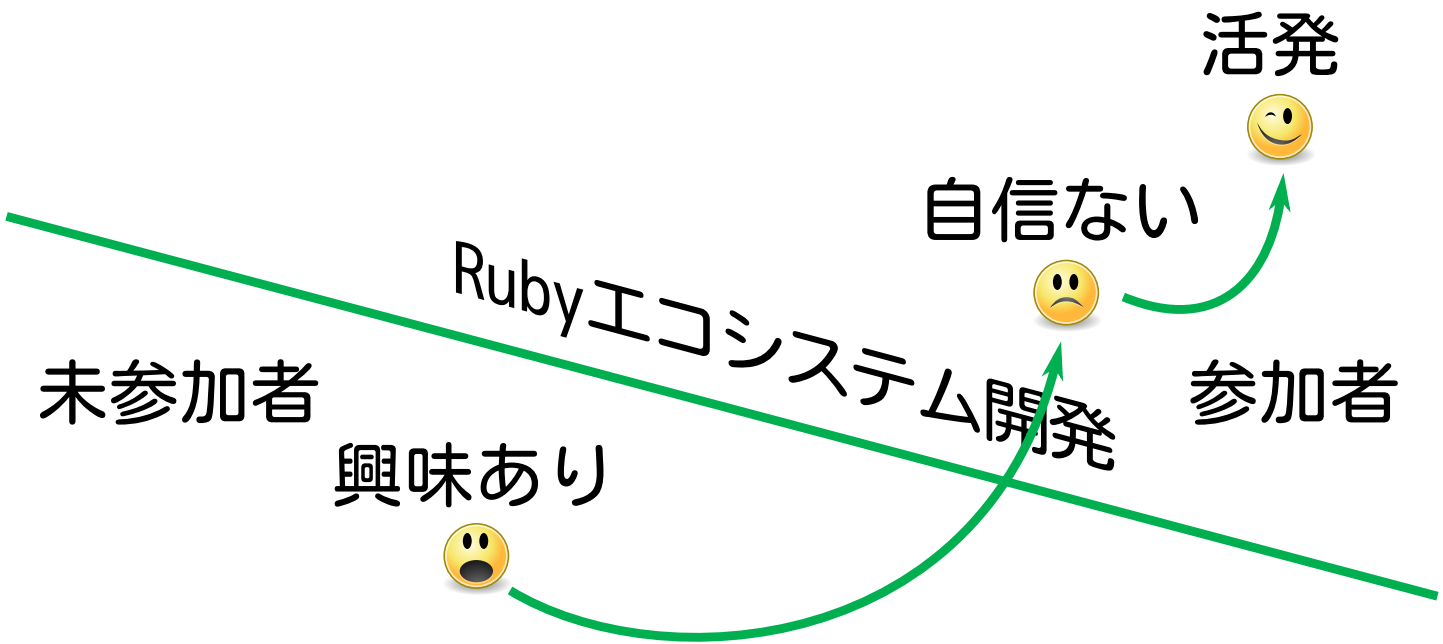
体験

体験してたいしたことはないとわかる→敷居が下がる

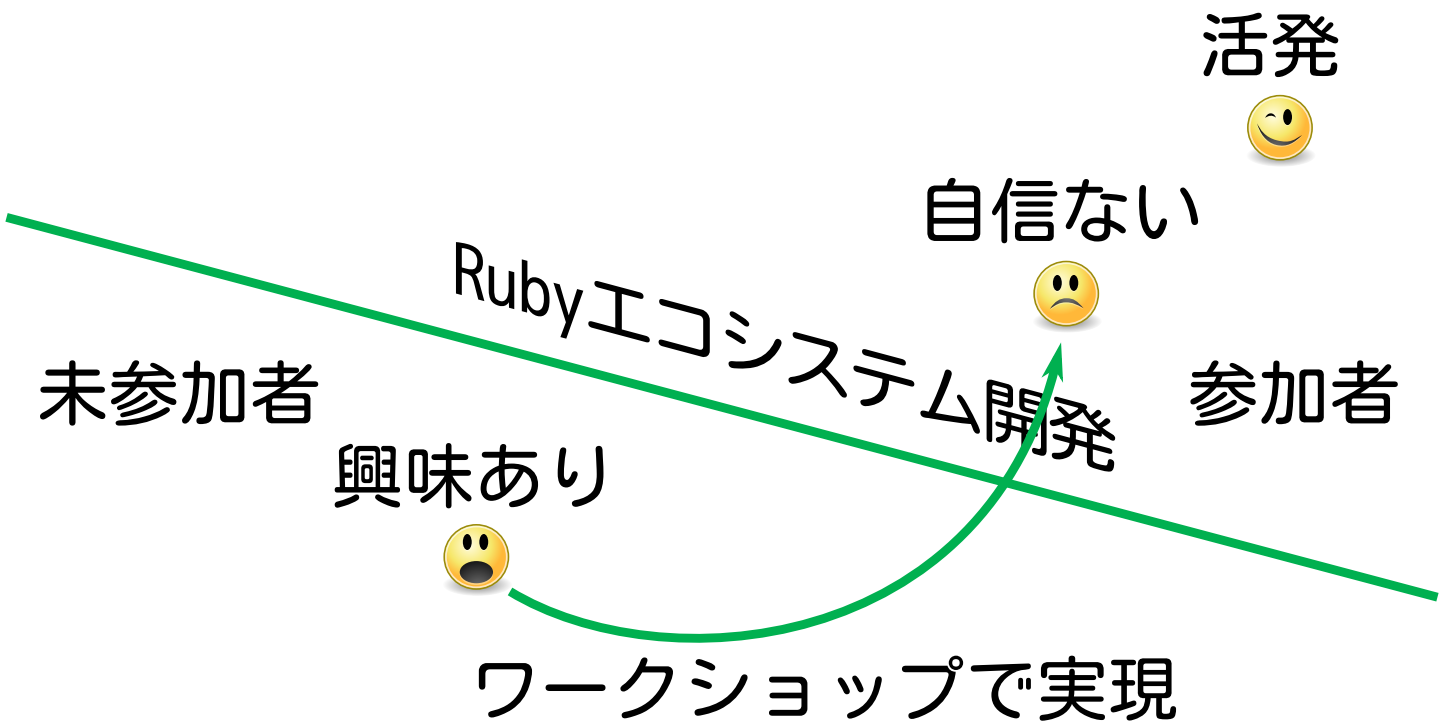
ターゲット



やりたいこと



未参加者→参加者



ワークショップの内容1

参加者のこと

立場一覧

- ✓ ビギナー
- ✓ サポーター
- ✓ サポートメンター
- ✓ 進行役

ビギナー

- ✓ Rubyエコシステムの開発に参加したい
  - ✓ でも参加したことはない
- ✓ Rubyエコシステムの開発に参加経験あり
  - ✓ でもまだ自信がない

サポーター

- ビギナーのサポート係
- Rubyエコシステム開発経験者
- 初参加でも大丈夫！
  - 例：進行役がやることを随時説明
  - 例：サポートメンターがサポート

サポートメンター

- ✓ サポーターのサポート係
- ✓ サポーター経験者
- ✓ 会場各地でスポットサポート
- ✓ サポート例：
  - ✓ うまくサポートできていない感…
    →相談しよう！
    サポーター1人で完璧にサポートしなくてよい！

進行役

- ✓ 進行と全体を気にかける係

ワークショップの内容2

流れ

今日の流れのポイント

- ✓ 未経験者の最初の1歩に最適化
  - ✓ ※Rubyエコシステムの開発方法はいろいろある
  - ✓ ※やりたい事がある人は応相談
  - ✓ ※基本的にこのやり方でやろう！

流れ

1. 動かす
   - ✓ Ruby本体の場合：
     公式ドキュメント通りRubyをビルド・テスト
   - ✓ gemの場合：**ユーザーとして**gemを動かす
     （README とかのドキュメント通り動かす）
2. ↑で気づいた事を開発元に
   フィードバック

期待

- ✓ 普段は気づいていないだけで
  実はフィードバックポイントが
  あったことを**体験**して！
  - ✓ ※ググったり生成AIに聞いて回避していない？
    そんなときどうしたらよいかはワークショップ内で！
- ✓ フィードバックを**体験**して！

ワークショップの内容3

動かす

動かす流れ

（詳細は後述）

1. 対象プロダクトを決める
2. 作業メモを書く場所を用意
3. 作業メモを書きながら
   公式ドキュメント・README通り
   動かす

対象プロダクト

- ✓ Ruby本体
- ✓ OSSのgem

OSSとは

- オープンソースライセンスを設定したソフトウェア
- オープンソースライセンス
  - OSIがオープンソースを定義
    OSI: Open Source Initiative
  - OSIが定義を満たしたライセンスを承認
- 承認されたライセンスのリスト
  - https://opensource.org/licenses/

OSSかどうかの判断方法

- ✓ ライセンスを確認するだけ
- ✓ 承認されたライセンスを使っていればOSS
- ✓ OSS「っぽい」は存在しない

例：Ruby本体

- https://github.com/ruby/ruby/blob/master/COPYING.ja
  - Rubyライセンス
- https://opensource.org/licenses/で検索
  - ない！
- RubyはOSSじゃない！？

Rubyライセンス

- ✓ オープンソースの定義は満たし…そう？
  - ✓ ベースのArtisticライセンスは承認されている
    いわくつきだけど
- ✓ でも、承認されていない
  - ✓ 申請していないから
    [OSS]RubyライセンスとOSI承認 - Matzにっき(2003-06-07)
    https://matz.rubyist.net/20030607.html#p06
- ✓ 2条項BSDライセンス（承認されている）とのデュアルライセンスなのでOSS！
  前はGPLとのデュアルライセンスだった

対象プロダクト決め

- ✓ ビギナーが決める
  - ✓ Ruby本体or使っているgemから選ぶ
    Gemfileを眺めてみて！
  - ✓ 難易度は気にしなくてよい！
    サポーターがサポートするから！
- ✓ サポーターは↑をサポート
  - ✓ 自分の知らないgemでもよい
    ビギナーと一緒に悩んであげよう！

対象gem決めデモ

デモ

- ✓ 最近使っているgemは？
  - ✓ ライセンス確認→OK！
- ✓ その中で一番ときめくのは？
- ✓ ではそれにしましょう！

動かすときのポイント

- ✓ 作業メモを書く
  - ✓  メモを書く場所はこのあと作る
- ✓ なにかする毎に書く
  - ✓ 例：ドキュメントを読み始めた
  - ✓ 例：次のドキュメントを読み始めた

作業メモを書く場所を作る

デモ

1. GitHub: oss-gate/workshop
2. ↑にissueを作る
3. 周囲のビギナーの人たちが
   作ったissueにコメント

作業メモの例

ドキュメント通りインストールしたけど失敗した。

よりよい作業メモの例

```
https://... のインストール手順をなぞろう！
 (↑後から再度参照できるようにURLも書く)
gem installでインストールできるはずなのに失敗した
 (↑期待する結果)

  $ gem install XXX (←なにをしたか)
   (...コマンドの実行結果...)
   (↑実際の結果)
  XXX is not found
↑というようにパッケージがないと言われる
```

ユーザーとして動かす

デモ

1. 公式サイトを開く
2. 作業メモを書く
3. 概要を読む
4. 作業メモを書く
5. ...

作業開始！

●時▲分まで！

1. 公式サイトを開く
2. 作業メモを書く
3. 概要を読む
4. 作業メモを書く
5. ...

ふりかえり1

・・・●時▲分！

- ✓ これまでの活動を見直す機会
- ✓ 目的：
  - ✓ 他の人の視点での考え方を知る
  - ✓ 作業ログが役に立つことを実感

ふりかえり1：デモ

デモ

- ビギナー：
  - 作業メモを読む
- サポーター：
  - 気になることをビギナーに質問
  - フィードバックポイントを確認
  - 完了→issueにコメント

ふりかえり1：進め方

- ✓ サポーターを他の人に交代
- ✓ 対象ビギナーの作業ログを
  ディスプレイに映す
- ✓ ビギナーが作業メモを読む
- ✓ 時間が余ったら：
  - ✓ 近くの他のビギナーにも説明

休憩

●時▲分まで！

現状確認

1. ~~ユーザーとして動かす~~
2. ~~ふりかえり1~~
3. ~~フィードバックポイントを発見！~~
4. ↑を**フィードバック**

フィードバック

- upstream（開発元）に
  うまくいかなかったことを報告
  - ここで詰まった、を伝える
  - こうだったらよかった、を伝える

報告方法

1. 整理する
   - ✓ 自分の考えが文章になればOK
2. **開発者にとって**
   わかりやすくなるように編集
3. 適切な場所に報告
   - ✓ GitHubのissueとか
   - ✓ Ruby本体ならhttps://bugs.ruby-lang.org/

1. 整理する

- ✓ 自分で自分の気持ちを理解
  - ✓ 自分が読んで理解できる文章にまとめられれば理解できている
  - ✓ 自分が理解できていないことは開発者にも伝えられない！
  - ✓ 作業メモに追記→サポーター確認

サポーターへ：メモ（断片）の文書化を手伝って
例：考えを整理できるような質問をする

整理方法

デモ

- ✓ 作業メモを開く
- ✓ フィードバック対象を決める
- ✓ 自分の気持ちを作業メモに追記
- ✓ サポーターに確認依頼

2. 編集する

- ✓ **開発者にとって**
  わかりやすくなるように編集
  - ✓ 報告方針をまとめているgemもある
    例：GitHubにあるCONTRIBUTING.md
  - ✓ 作業メモに追記→サポーターに確認

サポーターへ：リーダブル化を手伝って
例：自分が開発者ならこう読めると開発者視点を伝える

編集の仕方

- ポイント
  - **相手が**わかるように書く
  - 例：省略しない（具体的に書く）

省略例

インストールしました。
動きませんでした。
どうしたらいいでしょうか？

省略しない例

↓でインストール

```
$ gem install ...
(...実行結果...)
```

↑のように失敗しました。

環境：Ubuntu 26.04 amd64

なぜ省略しないか

- ✓ 相手は私を知らないから
  - ✓ 省略すると**想像**しないといけない
  - ✓ だいたい**想像は外れる**
  - ✓ 話が噛み合わない！

省略しないとは

- ✓ 詳細を書く
  - ✓ 実行したコマンド・実行結果
- ✓ やったことを書く
- ✓ **やっていないこと**を書く
- ✓ 期待した結果を書く

編集方法

デモ

- ✓ 作業メモを開く
- ✓ 自分の気持ちを開発者に伝わるように
  まとめて作業メモに追記
- ✓ サポーターに確認依頼

3. 報告する

- ✓ 適切な場所に報告
  - ✓ プロダクトによって報告場所は違う
- ✓ サポーターへ
  - ✓ 報告に二の足を踏んでいる人の背中を押してあげて
    例：開発者視点を伝える：自分ならこの報告をもらったら助かる

報告方法

デモ

- ✓ 報告方法を探す
- ✓ サポーターに後押ししてもらう
- ✓ まとめた報告内容を報告

報告
●時▲分まで！

1. 整理する
2. 開発者にとってわかりやすくなるように編集する
3. 適切な場所に報告する
4. 報告したことをチャットで自慢！

   ✓ ruby-jp Slackの#oss_gate

ふりかえり2：デモ

デモ

- ビギナー：
  - 作業メモを読む
- サポーター：
  - **よかったことをよい！と言う**
  - 気になることをビギナーに質問
  - 完了→issueにコメント

ふりかえり2：進め方

- ✓ サポーターを他の人に交代
- ✓ 対象ビギナーの作業ログを
  ディスプレイに映す
- ✓ ビギナーが作業メモを読む
- ✓ 時間が余ったら：
  - ✓ 近くの他のビギナーにも説明

Rubyエコシステム開発関連情報

- ✓ 生成AI

生成AIとRubyエコシステム開発

- ✓ Ruby開発者は結構使っている
  - ✓ Ruby開発者以外が使ってPRするのはあまり見ない
- ✓ gemはそれぞれでいろいろありそう
  - ✓ よい付き合い方はまだ探り探り
- ✓ これは困るというのは見えてきている
- ✓ ポイント：
  自分も開発チームの1人という振る舞い

生成AIで報告

- ✓ 生成された報告は冗長なことが多い
  - ✓ 無用な冗長さは開発者を疲弊させる
  - ✓ 開発チームの1人としてどうしたい？
- ✓ 必須：報告前に報告者がレビュー！
  - ✓ 報告者がそもそも理解できていること
    - ✓ 開発者に聞かれたら自分で答えられるくらい
  - ✓ 報告者が不要な情報を減らす

生成AIでプルリクエスト

- ✓ 大きなプルリクエストになりがち
  - ✓ 大きなプルリクエストは開発者を疲弊させる
  - ✓ 開発チームの1人としてどうしたい？
- ✓ 必須：作成前に作成者がレビュー！
  - ✓ 詳細を理解する
  - ✓ レビュー可能なサイズにする

プルリクエストとレビュー

- ✓ 狙いの1つ：関係者間での知識の共有
  - ✓ 作成者が未理解→知識の共有は進まない
  - ✓ 作成者抜きで開発者が直接生成AIを使えばよい
- ✓ 開発チームの1人としてどうしたい？
  - ✓ 学んで次のプルリクエストに活かせる？
  - ✓ 学んだことを他の人に伝えられる？

生成AIとライセンス

- ✓ 機械が生成したものに著作権は発生しない
- ✓ 著作権があるコードの複製を生成しうる
  - ✓ そのコードのライセンスが大事
  - ✓ ライセンスを守れれば再利用可能
- ✓ ライセンスの問題がないことは
プルリクエスト作成者が保証する

生成AIとコミット

- ✓ どのツールを使ったかの情報を残す
  - ✓ GitならGenerated-By: ...などで記録
  - ✓ なにか問題があったときに調べられるように

生成AIとRubyエコシステム開発

- ✓ 生成したものを丸投げはダメ
- ✓ 生成したものを詳細まで理解した上で活用

まとめ

- ✓ 今日やったことを再確認
- ✓ 明日からのことを確認

目的の確認

Rubyエコシステム開発**未**経験者

↓

Rubyエコシステム開発　経験者

やったこと

Rubyエコシステム開発を体験する

1. 動かす
2. フィードバック

体験時のポイント

常にメモ

常にメモの理由

- ✓ 詰まったところに気づくため
  - ✓ いつもはスルーしていない？
  - ✓ 実はフィードバックポイント！

詰まったところ

- ✓ Rubyエコシステム開発参加の**チャンス**！
  - ✓ ポジティブに捉えてみよう
  - ✓ 実際に参加して楽しかった？
- ✓ 直ると次の人は**うまくいく**
  - ✓ 気分がいいね！

気づいた？

- ✓ コードを書くだけが
  Rubyエコシステム開発参加方法じゃない
  - ✓ 使いはじめのユーザーだから
    できることもある
- ✓ やり方を知ればやれる
  - ✓ 明日からもやってみよう！

明日からのやり方

- ✓ 自分が使っている他のgemでも
  やってみよう
  a. 動かす
  b. 気になったことをまとめる
  c. フィードバック
- ✓ ↑失敗が怖い？

Rubyエコシステムと失敗

- ✓ そもそも失敗と認識されない
  - ✓ 少なくとも1発アウト！はほぼない
  - ✓ 新規開発者は基本的にWelcome
- ✓ 失敗しても根に持たれない
  - ✓ 失敗→改善：改善後を評価

↓

明日からオススメ方法をTry！

メッセージ

不安がらずに
Rubyエコシステムの
開発を
楽しんで！

来てよかった！と思ったら

- ✓ 次回はサポーターとして参加！
- ✓ 次回をまわりに継続宣伝！
- ✓ #oss_gateで他の人をサポート

おねがい

- ✓ 今日のフィードバックを！
  - ✓ 次に活かしたい
- ✓ この後すぐ
  - ✓ アンケート記入
  - ✓ アンケート結果をみんなで確認

アンケートの回答方法

1. github.com/oss-gate/workshop をfork
2. git clone git@github.com:#{GITHUB_ID}/workshop.git
3. tutorial/retrospectives/#{YYYY}-#{MM}-#{DD}-ruby-#{LOCATION}
   - ✓ cp beginner.yaml beginner-#{GITHUB_ID}.yaml
   - ✓ cp supporter.yaml supporter-#{GITHUB_ID}.yaml
4. git switch -c retrospective
5. git add → git commit → git push
6. github.com/#{GITHUB_ID}/workshopを開いて「Pull request」