

# GH-900

## GitHub の基礎

本講義は以下のMicrosoft Learn教材に準拠しています。

<https://learn.microsoft.com/ja-jp/collections/66w2a7txpy78r1>



A screenshot of a Microsoft Learn course page. At the top left is a small profile icon of a person with short brown hair. To its right, the word "COURSE" is written in a light blue font. Below that, the title "GitHub Copilot" is displayed in a large, bold, dark blue font. Underneath the title, the text "Course GH-300T00-A: GitHub Copilot" is shown in a smaller, dark blue font.



# 本コースについて

- ・「GitHubをこれから使っていきたい」という初心者向けのコースです
- ・「**1日でサッと学べる GitHub 入門**」といったレベル感と内容です
- ・GitHubの基本的な利用方法（リポジトリ、Issues、Gitによる操作、プルリクエスト）や、GitHub Codespaces、複数人での共同開発におけるGitHubの使い方、GitHub Projectsなどについて学びます。

# 関連コースのご紹介

- GitHub Actions、GitHub Copilot、GitHub Advanced Securityについては別コースで詳しく解説しています。
- GH-200 GitHub Actions
- GH-300 GitHub Copilot
- GH-500 GitHub Advanced Security
- 本コースと合わせて、ご受講をご検討いただければと思います。

# 目次

- 1 GitHub の概要
- 2 GitHub Codespaces でのコーディング
- 3 Markdown を使用して GitHub で効果的にコミュニケーションする
- 4 GitHub を利用し、リポジトリ履歴を検索し、整理する
- 5 GitHub で pull request を使用してリポジトリの変更を管理する
- 6 GitHub のベストプラクティスを使用してセキュリティで保護されたり ポジトリを維持する
- 7 GitHub のオープンソースプロジェクトに貢献する
- 8 GitHub Projects で仕事を管理する

# 目次

- 1 GitHub の概要
- 2 GitHub Codespaces でのコーディング
- 3 Markdown を使用して GitHub で効果的にコミュニケーションする
- 4 GitHub を利用し、リポジトリ履歴を検索し、整理する
- 5 GitHub で pull request を使用してリポジトリの変更を管理する
- 6 GitHub のベストプラクティスを使用してセキュリティで保護されたり ポジトリを維持する
- 7 GitHub のオープンソースプロジェクトに貢献する
- 8 GitHub Projects で仕事を管理する

# モジュール1 GitHubの概要

- GitHubとは？
- よくあるご質問
- GitHubに含まれる機能
- GitHubのアカウント
- GitHubのリポジトリ
- Gitの操作
- GitHubの組織

# モジュール1 GitHubの概要

- GitHubとは？
- よくあるご質問
- GitHubに含まれる機能
- GitHubのアカウント
- GitHubのリポジトリ
- Gitの操作
- GitHubの組織



# GitHub (ギットハブ) とは？

- ・ソフトウェア開発のプラットフォーム
- ・インターネット上での多数の開発者とのコラボレーションが可能
- ・個人でも使えるが、エンタープライズ（企業）向けの機能もある
- ・Git（バージョン管理ツール）を使用してソースコードのバージョン管理を行う
- ・2018/6/4, マイクロソフトはGitHubの買収を発表、同10月に買収を完了。現在GitHubはマイクロソフト傘下となっている。

[マイクロソフト、GitHubの買収を完了 - ZDNET Japan](#)

<https://japan.cnet.com/article/35120250/>

# モジュール1 GitHubの概要

- GitHubとは？
- よくあるご質問
- GitHubに含まれる機能
- GitHubのアカウント
- GitHubのリポジトリ
- Gitの操作
- GitHubの組織

# よくあるご質問

- ・マイクロソフトのDevOps関連サービスとしては、 Azure DevOps ServiceとGitHubの2つがあるということになります。どちらを採用すべきですか？
- ・→簡単に言えば、手軽で使いやすいDevOpsツールがほしい場合は GitHub、企業向けの豊富な機能を利用したい場合は Azure DevOps Service という選択になります。
- ・どちらを採用しても似たようなDevOps機能（プロジェクト管理、バージョン管理、CI/CDなど）が利用できますが、多くの場合、GitHubのほうがより簡単に使え、 Azure DevOps Serviceのほうはより高度で複雑な機能を利用できる、というパターンが多いように思います。
- ・ Azure DevOps Serviceにしかない企業向けの機能があります。たとえばEntra IDを使用したユーザー管理・アクセス制御など。
- ・ Azure DevOps ServiceとGitHubを組み合わせて運用することも可能です
- ・ぜひ両方を検討してみてください！

# よくあるご質問

- ・マイクロソフトがGitHubを買収したことにより、今後、Azure DevOpsやGitHubが統合されたり、あるいはどちらかが廃止されたり、といった予定はありますか？
- ・→特にそのような予定はありません。
- ・GitHubに関して、マイクロソフトは買収時に「何も変えない」「独立したオープンなプラットフォームであり続ける」ことを約束しています。
- ・GitHubもAzure DevOpsも、どちらも引き続きご利用いただけます。

# モジュール1 GitHubの概要

- GitHubとは？
- よくあるご質問
- GitHubに含まれる機能
- GitHubのアカウント
- GitHubのリポジトリ
- Gitの操作
- GitHubの組織

# GitHubに含まれる機能

- **GitHub Actions:**
  - 各コミットやプルリクエスト時などに自動的にビルド・テスト・デプロイを実行。
  - → 詳しくは、別コース **「GH-200 GitHub Actions」** で解説しています。

# GitHubに含まれる機能

- **GitHub Copilot:**

- Visual Studio Code、PyCharm、Eclipseなどの開発ツールに組み込んで使用するAIツール
- 開発者のコード記述、コメント記述、テスト開発などをAIが支援
- →詳しくは、別コース **「GH-300 GitHub Copilot」** で解説しています。

# GitHubに含まれる機能

- **GitHub Advanced Security** (GHAS)
  - コードの脆弱性を発見する「Code Scanning」を全てのプルリクエストに対して実施する
  - →詳しくは、別コース **「GH-500: GitHub Advanced Security」** で解説しています。

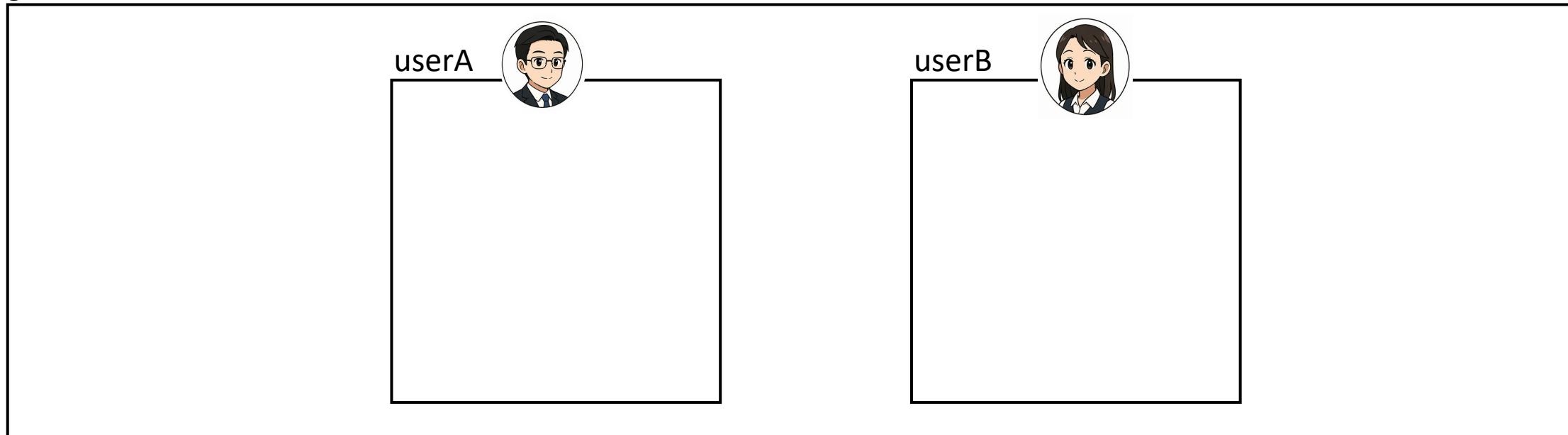
# モジュール1 GitHubの概要

- GitHubとは？
- よくあるご質問
- GitHubに含まれる機能
- GitHubのアカウント
- GitHubのリポジトリ
- Gitの操作
- GitHubの組織

# GitHubのアカウント

- GitHubを利用したいユーザーは、自分のGitHubアカウントを作成する
  - ・企業での利用の場合は組織から払い出されるアカウントを使用
- 個人用のアカウントは無料で作成でき、GitHubの基本機能は無料で利用できる(GitHub Freeプラン)
  - ・より多くの機能を使用するための有料プランもある (GitHub Pro、Team、Enterprise)

github.com

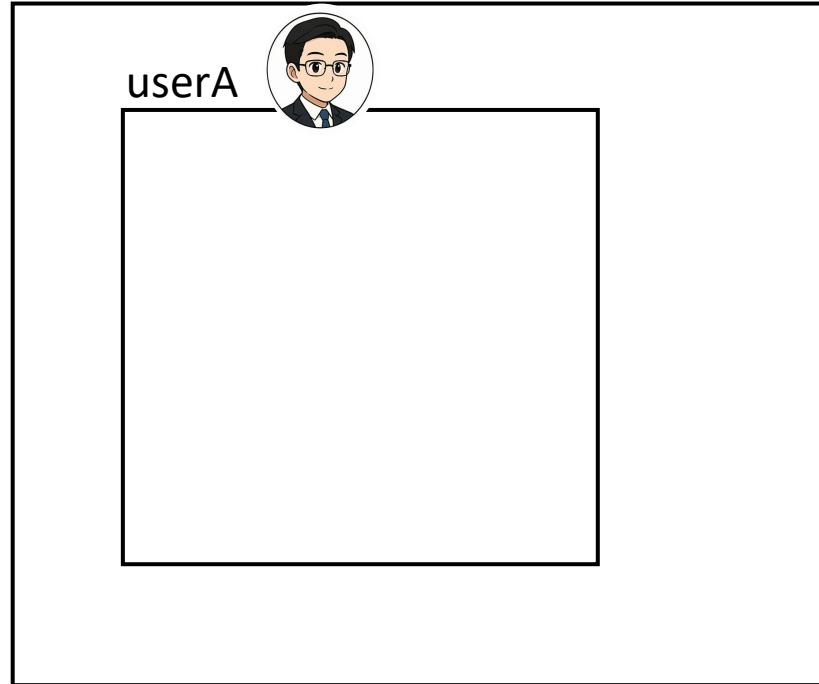


[GitHub アカウントの種類 - GitHub Docs](#)

[GitHub のプラン - GitHub Docs](#)

- GitHubアカウントを作成
- Windowsの場合は、パソコンにGitをインストール

github.com



userA のパソコン



Git for Windows

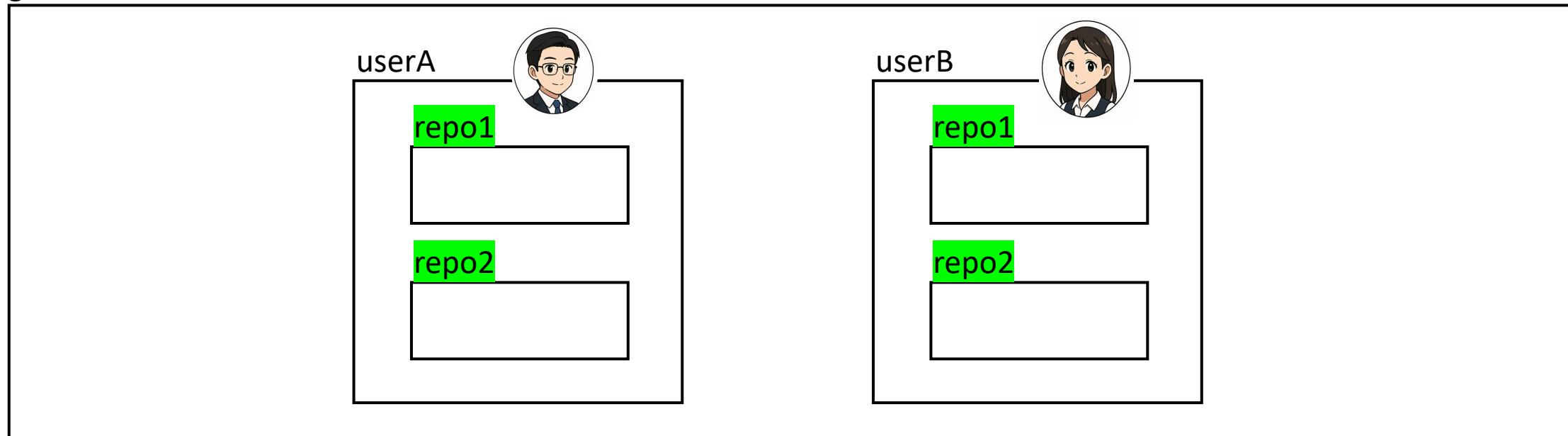
# モジュール1 GitHubの概要

- GitHubとは？
- よくあるご質問
- GitHubに含まれる機能
- GitHubのアカウント
- GitHubのリポジトリ
- Gitの操作
- GitHubの組織

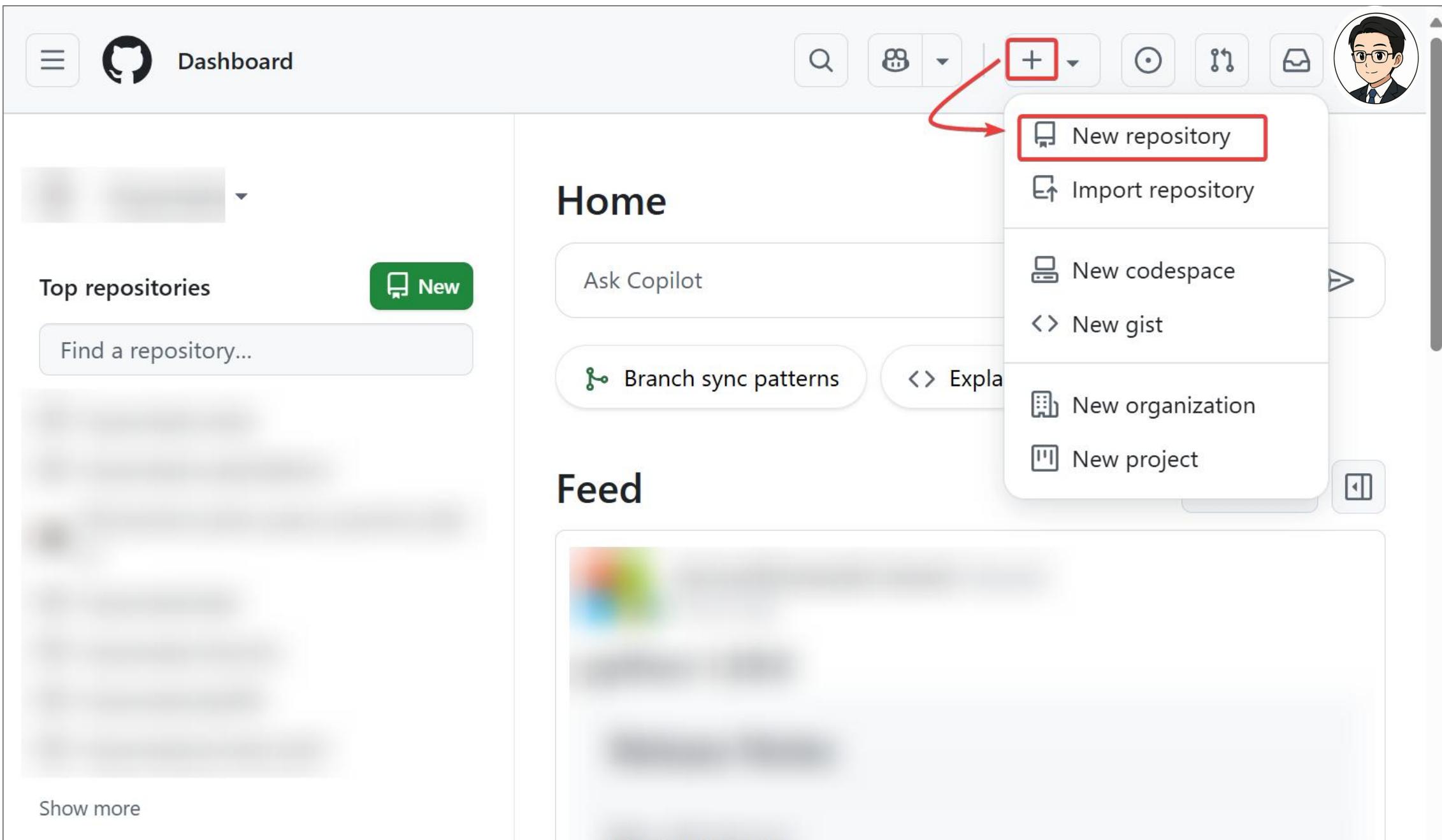
# GitHubのリポジトリ (Repositories)

- ・ソースコードなどを管理するための「入れ物」
- ・GitHubの最も基本的な構造
- ・複数のリポジトリを作成して利用できる
- ・パブリック（公開）またはプライベート（非公開）に設定可能

github.com



■GitHub (<https://github.com/>) にサインインし、画面右上の「+」をクリック、「New repository」をクリック



## ■リポジトリ名を入力して「Create repository」をクリック

New repository

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?  
[Import a repository.](#)

Required fields are marked with an asterisk (\*).

Repository template

No template ▾

Start your repository with a template repository's contents.

Owner \* / Repository name \*

repo4

repo4 is available.

Great repository names are short and memorable. Need inspiration? How about [bookish-doodle](#) ?

Description (optional)

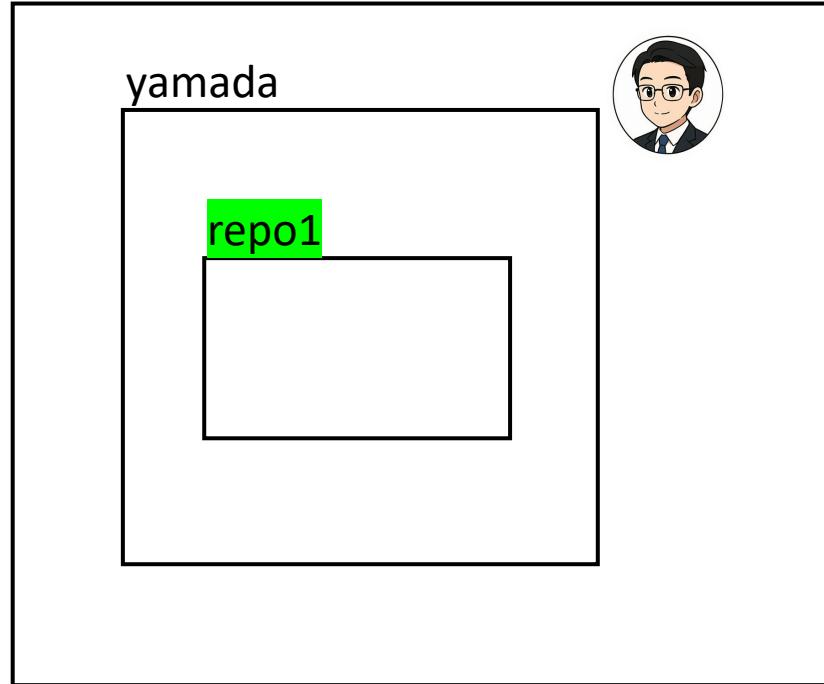
Create repository

# モジュール1 GitHubの概要

- GitHubとは？
- よくあるご質問
- GitHubに含まれる機能
- GitHubのアカウント
- GitHubのリポジトリ
- Gitの操作
- GitHubの組織

- リポジトリ repo1 を新規作成した状態

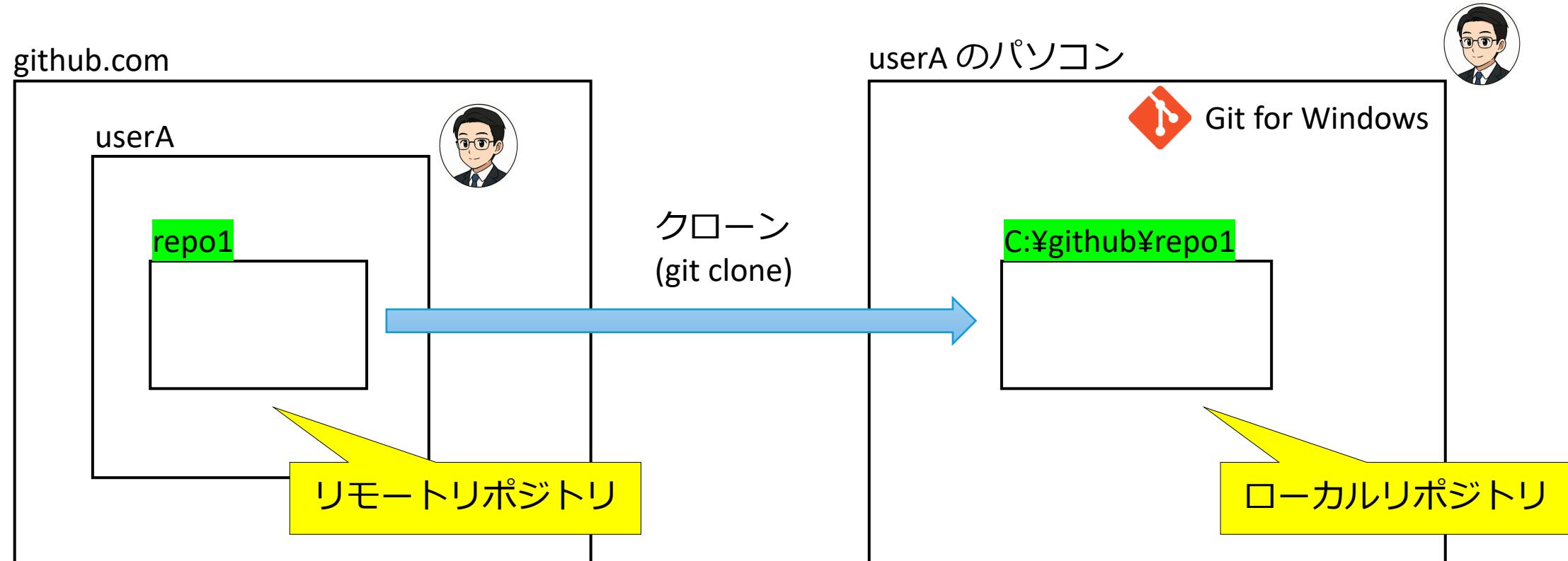
github.com



yamada のパソコン



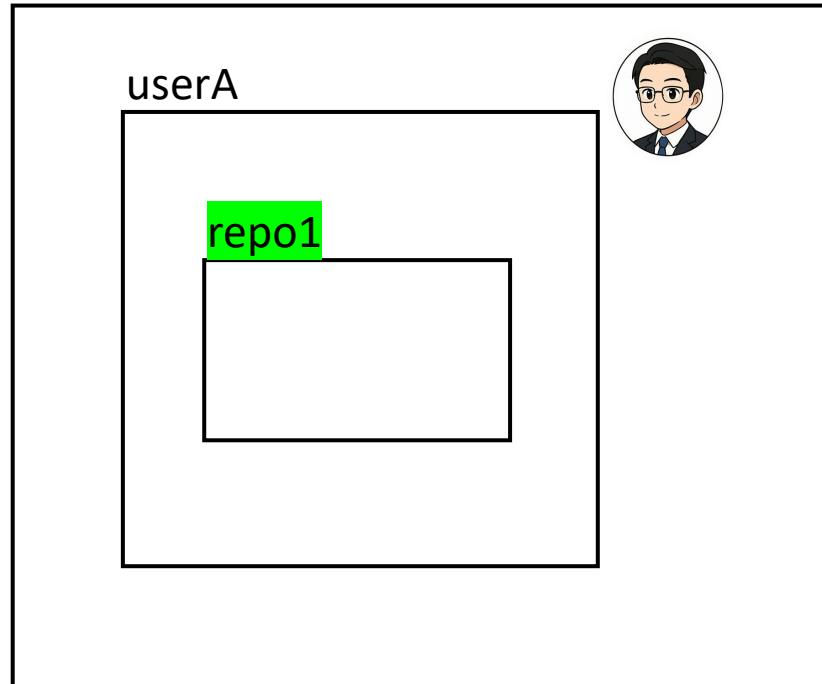
- リポジトリを「クローン」して、ローカルリポジトリを作成



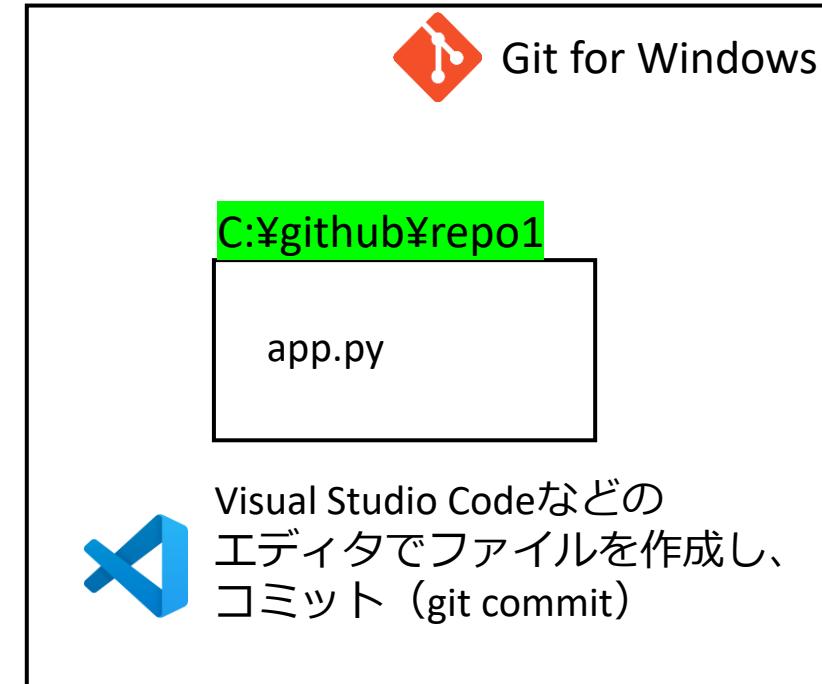
```
git clone https://github.com/userA/repo1
```

- ・ローカルリポジトリでファイルを作成し、コミット (Gitで変更を記録)

github.com

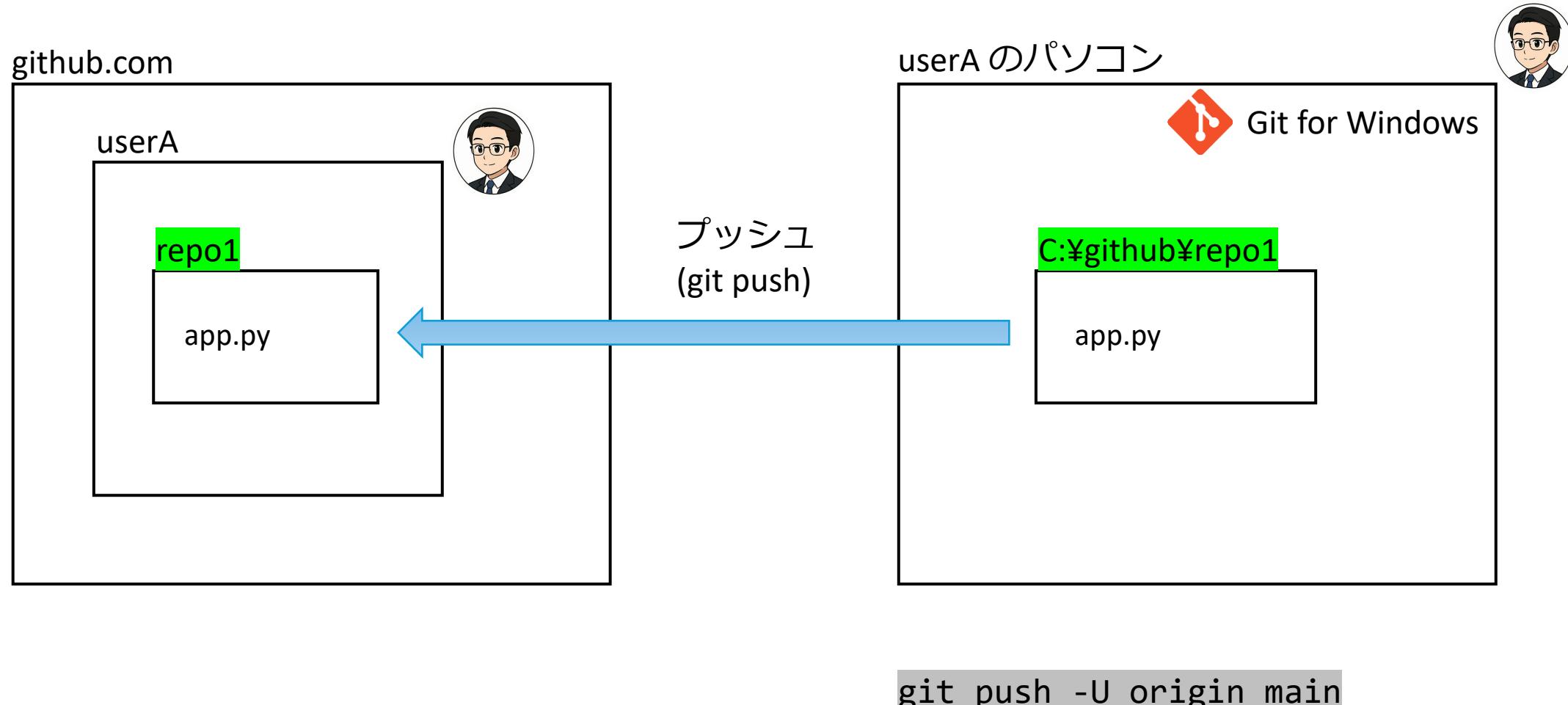


userA のパソコン



```
git add -A  
git commit -m '(commit message)'
```

- ローカルリポジトリの変更をリモートリポジトリ (GitHub) へとプッシュ



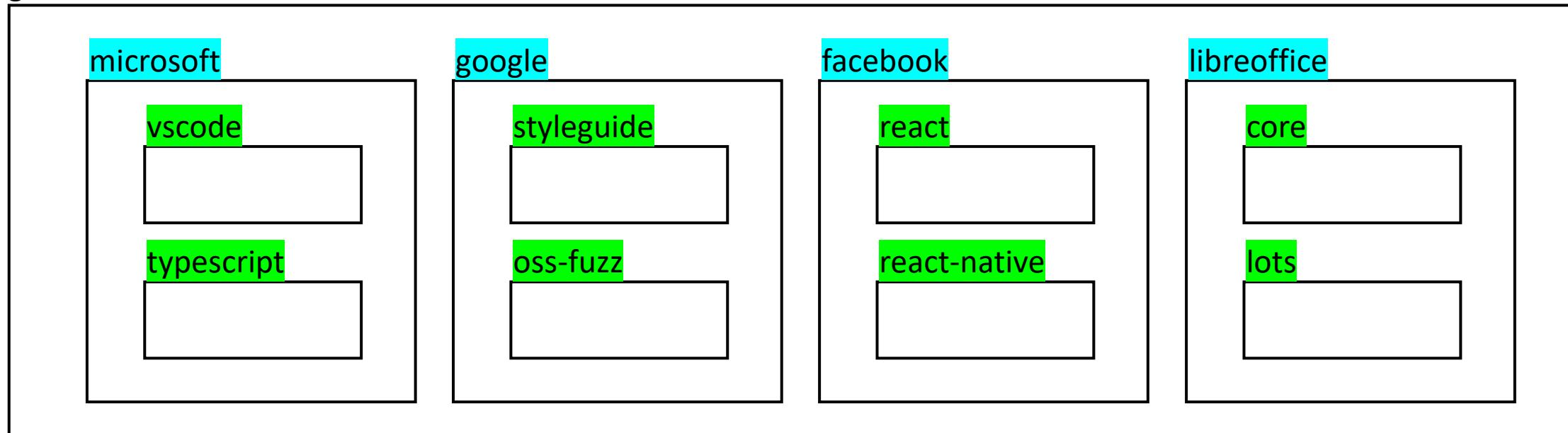
# モジュール1 GitHubの概要

- GitHubとは？
- よくあるご質問
- GitHubに含まれる機能
- GitHubのアカウント
- GitHubのリポジトリ
- Gitの操作
- GitHubの組織

# GitHubの「組織」 (Organizations)

- GitHubのアカウントの一種
- 複数のリポジトリを束ねるためのしくみ
- 企業や、オープンソースプロジェクトの団体などが利用
- 以下の図の青の部分が「組織」、緑の部分が「リポジトリ」

github.com



[GitHub アカウントの種類 - GitHub Docs](#)

[Organizationについて - GitHub Docs](#)

# まとめ

- GitHubとは？
- ソフトウェア開発のプラットフォーム。インターネット上での多数の開発者とのコラボレーションが可能。エンタープライズ（企業）向けの機能もある。Gitを使用してバージョン管理を行う。自動化機能（GitHub Actions）、セキュリティ機能（GitHub Advanced Security）、AIによる開発者支援機能（GitHub Copilot）などを利用できる。GitHubアカウントを作り、リポジトリ内で、Gitを使用してソースコードを管理する。

# 目次

- 1 GitHub の概要
- 2 GitHub Codespaces でのコーディング
- 3 Markdown を使用して GitHub で効果的にコミュニケーションする
- 4 GitHub を利用し、リポジトリ履歴を検索し、整理する
- 5 GitHub で pull request を使用してリポジトリの変更を管理する
- 6 GitHub のベストプラクティスを使用してセキュリティで保護されたり ポジトリを維持する
- 7 GitHub のオープンソースプロジェクトに貢献する
- 8 GitHub Projects で仕事を管理する

# GitHub Codespacesとは？

- Webブラウザーからアクセスして利用できる、クラウドベースの開発環境
- Visual Studio Codeがベースとなっている
- 2020/5/7 発表、2021/8/11リリース
- 2022/11/10 個人ユーザーの場合は月60時間まで無料で利用可能に

[What's new with Codespaces from GitHub Universe 2022 - The GitHub Blog](#)

[Codespaces is generally available for Team and Enterprise - GitHub Changelog](#)

[GitHub、WebIDEの「Codespaces」を発表。GitHubからワンクリックで開発環境へ。GitHub Satellite 2020 – Publickey](#)

[GitHub Codespaces とは - GitHub Docs](#)

# <講師デモ> GitHub Codespaces

- GitHubリポジトリを開く
- GitHub Codespacesを起動
- GitHub Codespaces内でコードを実行

# まとめ

- GitHub Codespacesとは？
- Webブラウザーからアクセスして利用できる、クラウドベースの開発環境。Visual Studio Codeがベースとなっている。コードの開発・実行が可能。無料でも利用できる。

# 目次

- 1 GitHub の概要
- 2 GitHub Codespaces でのコーディング
- 3 Markdown を使用して GitHub で効果的にコミュニケーションする
- 4 GitHub を利用し、リポジトリ履歴を検索し、整理する
- 5 GitHub で pull request を使用してリポジトリの変更を管理する
- 6 GitHub のベストプラクティスを使用してセキュリティで保護されたり ポジトリを維持する
- 7 GitHub のオープンソースプロジェクトに貢献する
- 8 GitHub Projects で仕事を管理する

# モジュール3

- Markdownとは？
- Markdownが利用できるところ
- GFM（GitHub Flavored Markdown）とは？
- Mermaidとは？
- IssuesでのMarkdownの利用例

# モジュール3

- Markdownとは？
- Markdownが利用できるところ
- GFM（GitHub Flavored Markdown）とは？
- Mermaidとは？
- IssuesでのMarkdownの利用例

# Markdownとは？

- ・構造化されたドキュメントを作成するためのプレーンテキスト形式
  - ・例:
    - # 見出し
    - - 箇条書き
    - \*\*太字\*\*
- ・シンプルな文法で、人間にも読みやすく、書きやすい

# モジュール3

- Markdownとは？
- Markdownが利用できるところ
- GFM（GitHub Flavored Markdown）とは？
- Mermaidとは？
- IssuesでのMarkdownの利用例

# Markdownが利用できるところ

- README.md などのファイル
- Issues、プルリクエストなどのコメント
- Wiki
- GitHub Pages
- GitHub Discussions

# モジュール3

- Markdownとは？
- Markdownが利用できるところ
- GFM（GitHub Flavored Markdown）とは？
- Mermaidとは？
- IssuesでのMarkdownの利用例

# GFM (GitHub Flavored Markdown) とは？

- GitHub.com で利用可能な Markdown 記法は GitHub Flavored Markdown (GFM) と呼ばれる。
- GitHub 独自の記法の例:
  - 「:thumbsup:」 → 、 「:octocat:」 → 
  - @ユーザー名 (メンション、そのユーザーに通知が飛ぶ)
  - #番号 (IssueやPull requestの番号、それらへのリンクとなる)
  - - [ ] 未完了のタスク
  - - [x] 完了のタスク

[GitHub Flavored Markdown Spec](#)

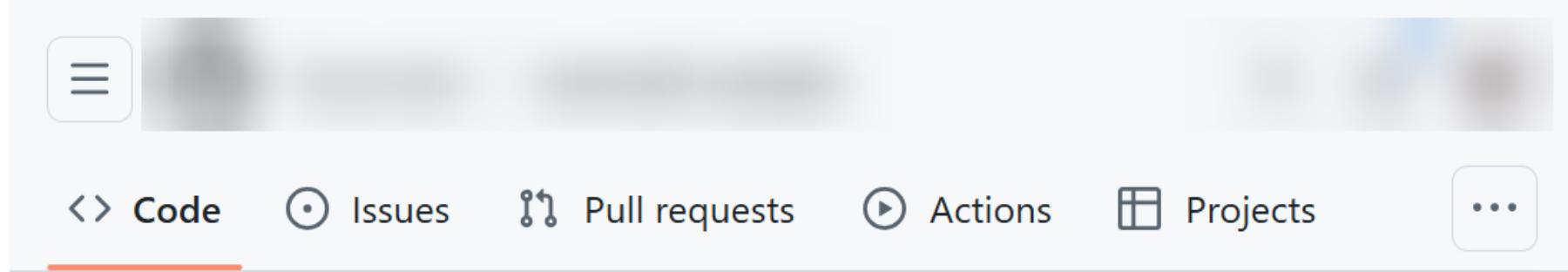
[Markdown - Wikipedia](#)

# モジュール3

- Markdownとは？
- Markdownが利用できるところ
- GFM（GitHub Flavored Markdown）とは？
- Mermaidとは？
- IssuesでのMarkdownの利用例

# Mermaidとは？

- Mermaid（マーメイド）は、フローチャート、シーケンス図、クラス図、ER（Entity-Relationship）図などを簡単なテキストで記述するための文法、およびそれを処理するためのJavaScriptライブラリ
- GitHubのMarkdown内ではMermaid記法も利用できる



← Files / **mermaid-samples** **README.md** in **main** **Cancel changes** **Commit changes...**

Edit Preview

Spaces

2

Soft wrap

```
1 # mermaid-samples
2
3 ````mermaid
4 stateDiagram-v2
5 [*] --> Idle
6 Idle --> Running : start
7 Running --> Paused : pause
8 Paused --> Running : resume
9 Running --> Stopped : stop
10 ````
```

Code Issues Pull requests Actions Projects Wiki Security Insights ...

main / README.md

Update README.md 4c3cf4a · now

10 lines (9 loc) · 190 Bytes

Preview Code Blame

Raw

mermaid-samples

```
graph TD; start(( )) --> Idle[Idle]; Idle -- start --> Running[Running]; Running -- pause --> Paused[Paused]; Running -- resume --> Running; Running -- stop --> Stopped[Stopped];
```

# モジュール3

- Markdownとは？
- Markdownが利用できるところ
- GFM（GitHub Flavored Markdown）とは？
- Mermaidとは？
- IssuesでのMarkdownの利用例

# GitHub Markdownの利用例

- Issue の作成

- Issue へのコメント
- 担当者 (Assignee) の割り当て
- Issue に対応する修正作業の実施
- Issue のコメントでの「コミットID」や「メンション」の使用
- Issue のクローズ



なおここでは、この二人のユーザーが、リポジトリに対して書き込み権限を持っている場合について説明します。  
書き込み権限がない場合の共同作業についてはモジュール7で説明します。

## ■リポジトリの「Issues」をクリック

The screenshot shows a GitHub repository page for 'repo4'. The top navigation bar includes links for Code, Issues (which is highlighted with a red box), Pull requests, Actions, Projects, Security, Insights, and more. Below the navigation is a header with a user profile picture, the repository name 'repo4' (labeled as Private), and buttons for Unwatch (1), Fork (0), and Star (0). The main content area features sections for 'Start coding with Codespaces' (with a 'Create a codespace' button) and 'Add collaborators to this repository' (with a 'Invite collaborators' button). At the bottom, there's a light blue box for quick setup with options to 'Set up in Desktop' (using HTTPS or SSH), a copy link icon, and instructions to start by creating a new file or uploading an existing file, recommending README, LICENSE, and .gitignore.

/ repo4

Code Issues Pull requests Actions Projects Security Insights

repo4 Private

Unwatch 1 Fork 0 Star 0

Start coding with Codespaces

Add a README file and start coding in a secure, configurable, and dedicated development environment.

Create a codespace

Add collaborators to this repository

Search for people using their GitHub username or email address.

Invite collaborators

Quick setup — if you've done this kind of thing before

Set up in Desktop or HTTPS SSH git@github.com: /repo4.git

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

## ■ 「New issue」をクリック

The screenshot shows the GitHub Issues page interface. At the top, there is a navigation bar with links: Code, Issues (which is selected), Pull requests, Actions, Projects, Security, Insights, and a three-dot menu. Below the navigation bar is a search bar containing the query "is:issue state:open". To the right of the search bar are buttons for Labels, Milestones, and a prominent green "New issue" button, which is highlighted with a red box. Underneath the search bar are filters for Open (0) and Closed (0) issues, and dropdown menus for Author, Labels, Projects, Milestones, and a three-dot menu. On the left side of the main content area is a circular profile picture of a person with glasses and a suit. The central part of the page displays the message "No results" and the sub-instruction "Try adjusting your search filters."

## ■ Issueの「タイトル」 「説明文」を記述し、「Create」をクリック

Create new issue

Add a title \*

プロジェクトに README ファイルがない

Add a description

Write Preview

リポジトリに `README.md` ファイルを作る。

Paste, drop, or click to add files

Create more

Cancel

Create

Assignees  
No one - [Assign yourself](#)

Labels  
No labels

Projects  
No projects

Milestone  
No milestone

#### ■新しいIssueが作成された。

Code Issues Pull requests Actions Projects Security Insights Settings

# プロジェクトに README ファイルがない #1

Open

opened now

リポジトリに README.md ファイルを作る。

Create sub-issue

Add a comment

Write Preview

Use Markdown to format your comment

Paste, drop, or click to add files

Close issue  Comment

Assignees  
No one - Assign yourself

Labels  
No labels

Projects  
No projects

Milestone  
No milestone

Relationships  
None yet

Development

Code with Copilot Agent Mode

Create a branch for this issue or link a pull request.

# GitHub Markdownの利用例

- Issue の作成
- Issue へのコメント
- 担当者（Assignee）の割り当て
- Issue に対応する修正作業の実施
- Issue のコメントでの「コミットID」や「メンション」の使用
- Issue のクローズ



■ Issueにコメントを追加することができる。コメント本文を書いて「Comment」ボタンをクリック。

# プロジェクトに README ファイルがない #1

Open



opened 19 minutes ago ...

リポジトリに README.md ファイルを作る。

Create sub-issue ... Smile icon



Add a comment

Write Preview H B I ≡ <> ↗ ≡ 1/≡ ≡ @ ↵ ↺ ↻

この件は私が担当します

Paste, drop, or click to add files

Close with comment Comment

■コメントが追加された。

# プロジェクトに README ファイルがない #1

Open



opened 21 minutes ago

リポジトリに README.md ファイルを作る。

Create sub-issue ⏺ ☺



now

この件は私が担当します

☺



# GitHub Markdownの利用例

- Issue の作成
- Issue へのコメント
- 担当者（Assignee）の割り当て
- Issue に対応する修正作業の実施
- Issue のコメントでの「コミットID」や「メンション」の使用
- Issue のクローズ



■ 「Assignees」の歯車をクリックして、このIssueの担当者を選択。（ここでは自分自身にIssueを割り当てている）

プロジェクトに READMEファイルがない #1

Open

opened 47 minutes ago

リポジトリに README.md ファイルを作る。

Create sub-issue

25 minutes ago

この件は私が担当します

Assignees

Assign up to 10 people to this issue

Filter assignees

No projects

Milestone

No milestone

■担当者が設定された。担当者の設定は任意だが、設定しておくと、誰がこのIssueを担当しているのかがわかりやすくなる。また、Issueを検索する際の条件としても利用できる。

# プロジェクトに READMEファイルがない #1

Open

 opened 49 minutes ago ...

リポジトリに README.md ファイルを作る。

Create sub-issue Smile icon

 27 minutes ago ...

この件は私が担当します

Smile icon

 self-assigned this now

Edit New issue Print icon

**Assignees**  ⚙️

**Labels** No labels ⚙️

**Projects** No projects ⚙️

**Milestone** No milestone ⚙️

**Relationships** None yet ⚙️

# GitHub Markdownの利用例

- Issue の作成
- Issue へのコメント
- 担当者 (Assignee) の割り当て
- Issue に対応する修正作業の実施
- Issue のコメントでの「コミットID」や「メンション」の使用
- Issue のクローズ



## ■新しいファイルの作成を行う。リポジトリのトップページに移動し、Creating a new file をクリック

The screenshot shows a GitHub repository page for 'repo4'. The top navigation bar includes links for Code, Issues (1), Pull requests, Actions, Projects, Security, and Insights. The repository name 'repo4' is displayed with a 'Private' status. On the right, there are buttons for 'Unwatch' and a dropdown menu showing '2' notifications. Below the header, there are sections for 'Set up GitHub Copilot' and 'Add collaborators to this repository'. A large blue banner in the center provides 'Quick setup' instructions, mentioning 'Set up in Desktop', 'HTTPS', 'SSH', and a URL 'git@github.com:hiryamada/repo4.git'. It also suggests starting by creating a new file or uploading an existing one, which is highlighted with a red box. Below this, a command-line section shows the steps to initialize a repository: 'echo "# repo4" >> README.md', 'git init', 'git add README.md', 'git commit -m "first commit"', 'git branch -M main', 'git remote add origin git@github.com:hiryamada/repo4.git', and 'git push -u origin main'. A small profile picture of a person with dark hair and glasses is visible on the right side of the page.

■ファイル名と、ファイルの内容を入力。Commit changes... をクリック。

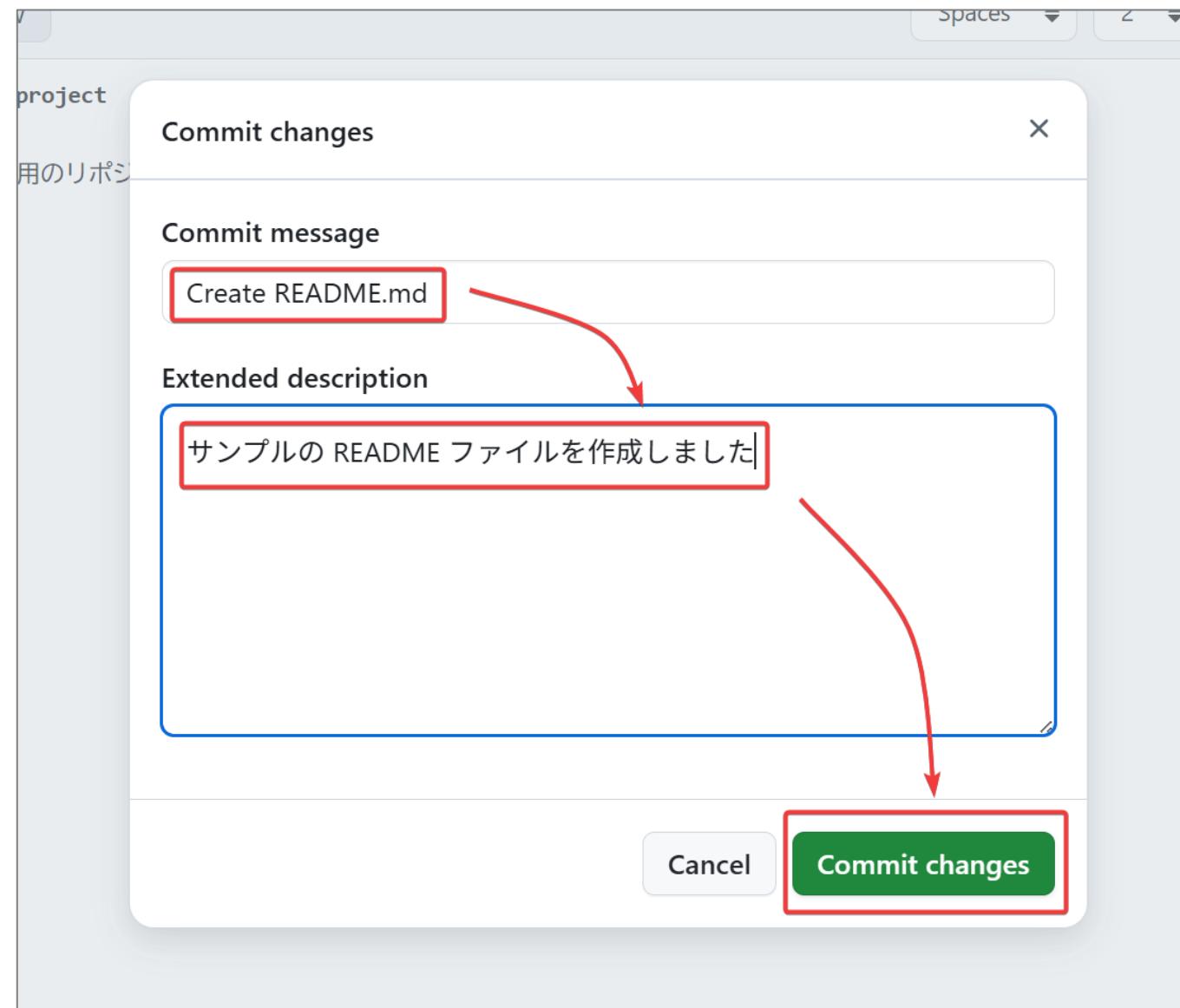
The screenshot shows a GitHub repository interface for a repository named 'repo4'. The 'Code' tab is selected. A file named 'README.md' is open for editing. The content of the file is:

```
1 # Sample project
2
3 これは練習用のリポジトリの `README.md` ファイルです。
4 |
```

The file name 'README.md' is highlighted with a red box. The 'Commit changes...' button in the top right corner of the editor is also highlighted with a red box and has a red arrow pointing to it from the bottom left.

Other visible elements include the GitHub logo in the top left, a search bar, and various navigation links like 'Issues', 'Pull requests', 'Actions', 'Projects', 'Security', and 'Insights'.

- 変更をコミットする（変更内容の確定・記録を行う）。  
Commit message（コミットの内容を説明する一文）を入力。  
オプションでExtended descriptionを入力。Commit changesをクリック。



■コミットが実行された（変更内容が確定・記録された）。README.md ファイルが作成された。

repo4 Private

Unwatch 2 Fork 0 Star 0

Help us improve GitHub Codespaces  
Tell us how to make GitHub Codespaces work better for you with three quick questions.

Give feedback ×

main Go to file + <> Code

Merge pull request #3 from hir... · 0c155aa · 1 hour ago

README.md Update README.md 1 hour ago

README

Sample project

これは練習用のリポジトリの README.md ファイルです。

About

No description, website, or topics provided.

Readme Activity 0 stars 2 watching 0 forks

Releases

No releases published Create a new release



# GitHub Markdownの利用例

- Issue の作成
- Issue へのコメント
- 担当者（Assignee）の割り当て
- Issue に対応する修正作業の実施
- Issue のコメントでの「コミットID」や「メンション」の使用
- Issue のクローズ



## ■ 時計のアイコン (commits) をクリック

repo4 Private

Unwatch 2 Fork 0 Star 0

Help us improve GitHub Codespaces  
Tell us how to make GitHub Codespaces work better for you with three quick questions.

Give feedback X

main Go to file + Code

Merge pull request #3 from hir... · 0c155aa · 1 hour ago

README.md Update README.md 1 hour ago

README

Sample project

これは練習用のリポジトリの README.md ファイルです。

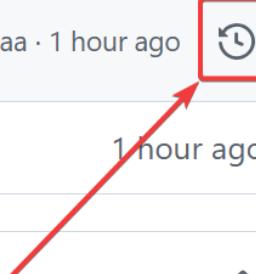
About

No description, website, or topics provided.

Readme Activity 0 stars 2 watching 0 forks

Releases

No releases published Create a new release



- すべてのコミットを確認できる。各コミットには「コミットID」（コミットを識別するチェックサムハッシュ）が付与される。これを使うことで、たとえばIssueのコメントの中で、各コミットを指示示すことができる。コミットIDをコピー。

## Commits

main ▾

All time ▾

-o Commits on Apr 29, 2025

Update README.md  
author 1 hour ago

Verified b9c7173 ↗ <>

- さきほどのIssueに戻り、修正を報告するコメントを書く。コメントにはコミットIDを付与したり、「@ユーザー名」で指定ユーザーに「メンション」（通知）したりできる。



## Add a comment

H B I | 三 <> ⌂ | 三 二 三 ⌂ ⌂ ←

Write

Preview



コミットID

修正しました。

9d6833a

「@ユーザー名」  
でユーザーを「メ  
ンション」できる  
(そのユーザーに  
通知される)

@



確認をお願いします。|

Add files

Close with comment

Comment



## ■メンションされたユーザーに送られてきた通知の例（メール）



 <notifications@github.com>  ⤵ ⤲ ⤳ | 📄 🌐 ...

宛先:  <repo4@noreply.github.com> 2025/04/30 (水) 0:04

Cc:   他 +1 人

  left a comment ([/repo4#1](#)).

修正しました。  
[9d6833a](#)

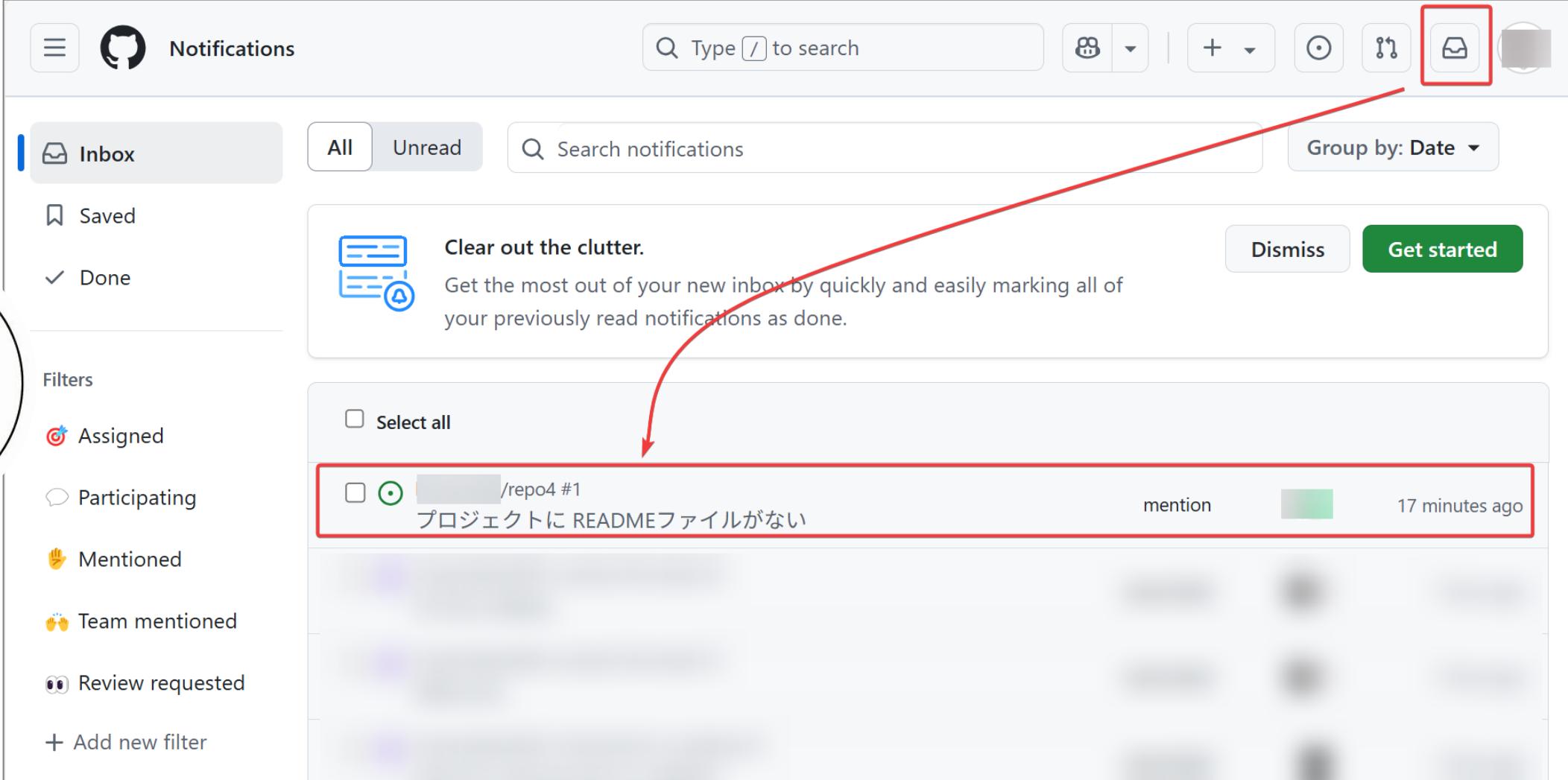
 @!  確認をお願いします。

—

Reply to this email directly, [view it on GitHub](#), or [unsubscribe](#).  
You are receiving this because you were mentioned.

 [返信](#)  [全員に返信](#)  [転送](#)

■メンションされたユーザーは、  
画面上部の notifications アイコンをクリックして、一覧形式でもメンションを確認できる



The screenshot shows the GitHub Notifications interface. On the left, there's a sidebar with a user profile picture and various filter options: Inbox (selected), All, Unread, Saved, Done, Filters, Assigned, Participating, Mentioned, Team mentioned, Review requested, and Add new filter. The main area displays notifications. At the top right of the main area, there are several icons: a search bar, a user icon, a plus sign, a circle with a dot, a gear, and an envelope icon (which is highlighted with a red box). Below these are two buttons: Dismiss and Get started. A large red arrow points from the 'Mentioned' filter in the sidebar down to the notification card for user '/repo4 #1'. The notification card itself has a red border around it. It contains the text 'Clear out the clutter.' followed by a description: 'Get the most out of your new inbox by quickly and easily marking all of your previously read notifications as done.' At the bottom of the notification card, there's a checkbox labeled 'Select all', another checkbox next to a green circle, the user handle '/repo4 #1', the text 'mention', and the timestamp '17 minutes ago'. The rest of the notifications in the list are blurred.

■コメントのコミットID（リンク）をクリックして、コミットの内容を確認できる。

The image shows a GitHub interface. On the left, a user profile picture of a woman is shown next to a comment card. The card contains the text "修正しました。" (Fixed), the commit ID "9d6833a", and a message "@ [redacted] 確認をお願いします。". A large red arrow points from the commit ID "9d6833a" in the comment to the detailed commit view on the right. The detailed view shows the commit "Commit 9d6833a" was authored 6 hours ago and is verified. It lists the file "Create README.md" was created, with the note "サンプルの README ファイルを作成しました". The commit has 0 parents and the commit ID is 9d6833a. The commit details show "1 file changed" with "README.md" having "+3 -0" lines changed. The diff shows the following changes:

...	@@ -0,0 +1,3 @@
1	+ # Sample project
2	+
3	+ これは練習用のリポジトリの `README.md` ファイルです。

At the bottom, there are "Comments 0" and a "Lock conversation" button.

# GitHub Markdownの利用例

- Issue の作成
- Issue へのコメント
- 担当者（Assignee）の割り当て
- Issue に対応する修正作業の実施
- Issue のコメントでの「コミットID」や「メンション」の使用
- Issue のクローズ



■このIssueの作業が完了したので、コメントを書いて、Issueをクローズする。

The screenshot shows a GitHub interface for managing issues. At the top, a comment from a user named 'あいこ' is visible, posted 21 minutes ago. The comment reads: "修正しました。" (Fixed), followed by a link to a commit: "9d6833a" and a message: "@確認をお願いします。" (Please check). Below this, there is a reply from another user, 'かずさ', with the message: "Add a comment".

The main focus is on the 'Write' tab of the comment editor, which contains the text: "確認しました！ありがとうございます。このIssueをクローズします。". This text is highlighted with a red box.

At the bottom right of the editor, there are two buttons: "Close with comment" (with a checked checkbox) and "Comment". The "Close with comment" button is also highlighted with a red box.

# まとめ(1/2)

- Markdownとは？
- 構造化されたドキュメントを作成するためのプレーンテキスト形式。シンプルな文法で、人間にも読みやすく、書きやすい。GitHubでは追加の文法（GitHub flavored Markdown, GFM）やMermaid（技術的な図を表現するしくみ）も利用できる。

# まとめ(2/2)

- GitHub Issuesとは？
- リポジトリにバグや機能追加要望などの「イシュー（問題）」を記録するためのしくみ。イシューを登録し、担当者を割り当てる。イシューのコメントではMarkdownを利用でき、「@ユーザー名」でユーザーへのメンション（通知）ができ、「#番号」でイシュー や プルリクエストへのリンクを参照できる。イシューが解決したらクローズする。

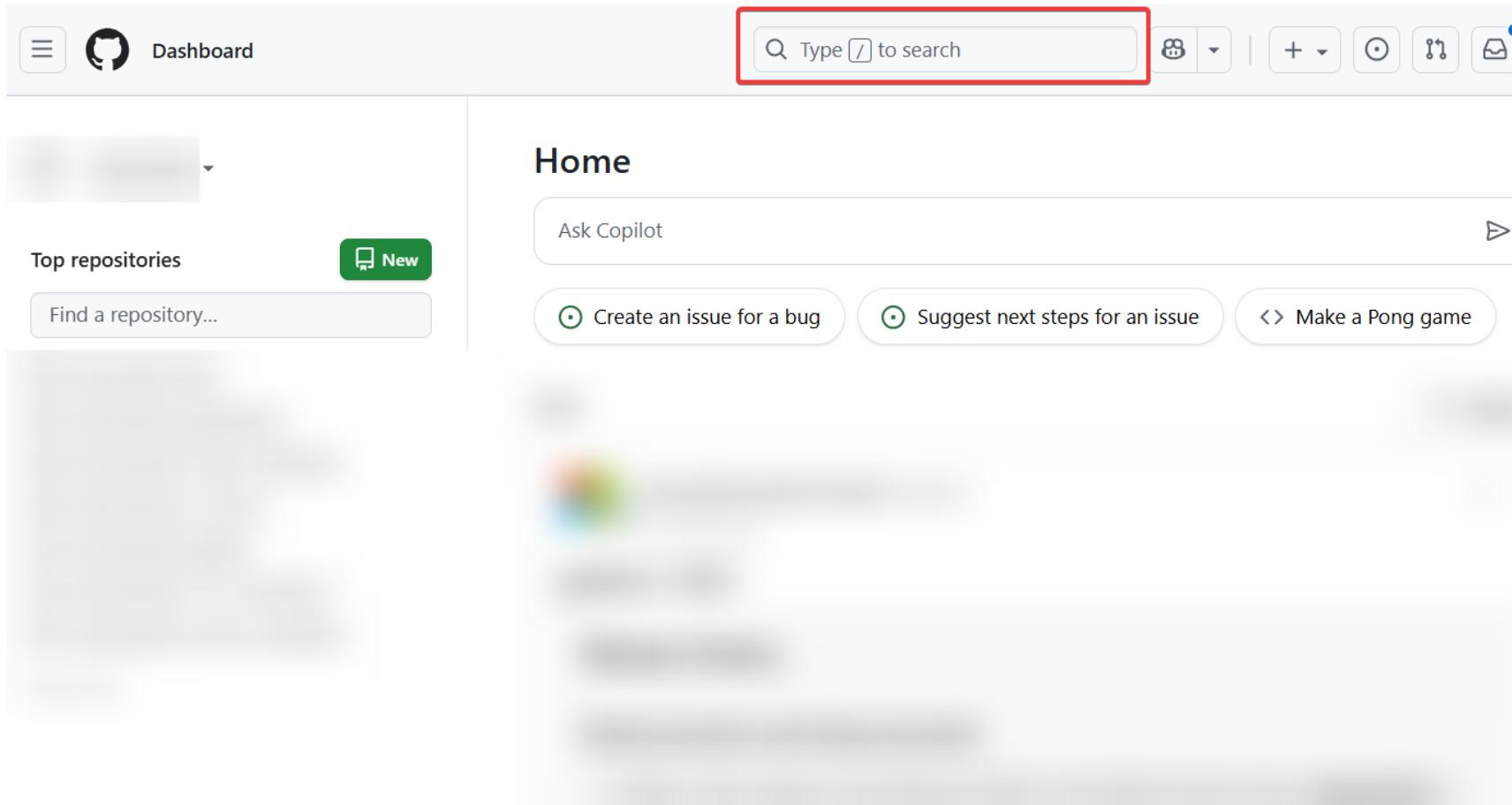
# 目次

- 1 GitHub の概要
- 2 GitHub Codespaces でのコーディング
- 3 Markdown を使用して GitHub で効果的にコミュニケーションする
- 4 GitHub を利用し、リポジトリ履歴を検索し、整理する
- 5 GitHub で pull request を使用してリポジトリの変更を管理する
- 6 GitHub のベストプラクティスを使用してセキュリティで保護されたり ポジトリを維持する
- 7 GitHub のオープンソースプロジェクトに貢献する
- 8 GitHub Projectsで仕事を管理する

# モジュール4

- GitHubの検索機能
- グローバル検索の例
- 特定のリポジトリでの検索の例

# GitHubの検索機能

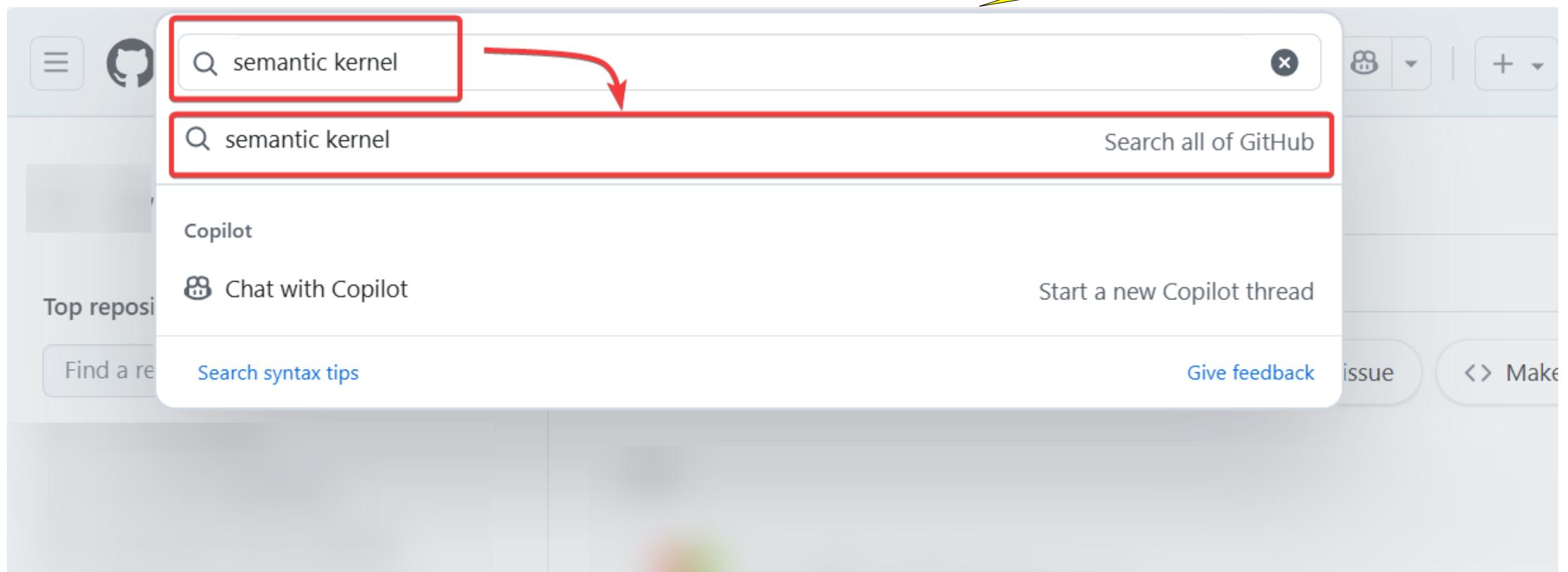


# GitHubの検索機能

- ・グローバル検索
  - ・GitHub 全体を検索する
  - ・GitHub.com に移動して、検索バーにキーワードを入力し、「Search all of GitHub」を選択
- ・スコープ検索
  - ・特定のリポジトリや組織内を検索する
  - ・リポジトリや組織のページに移動して、検索バーにキーワードを入力。

# グローバル検索の例

GitHubのトップページ  
(github.com)



# グローバル検索の結果の例

The screenshot shows a GitHub search interface with the query "semantic kernel" entered in the search bar. The results are filtered by "Repositories" and show 1.9k results found in 142 ms.

**Filter by:**

- Code: 1.7M
- Repositories: 1.9k** (selected)
- Issues: 15k
- Pull requests: 70k
- Discussions: 1k
- Users: 23
- More

**Languages:**

- C#: 1.9k
- Python: 1.7M
- Jupyter Notebook: 1.5M
- HTML: 1.2M
- Java: 1.0M
- TypeScript: 800k
- JavaScript: 700k
- Bicep: 500k
- C++: 300k

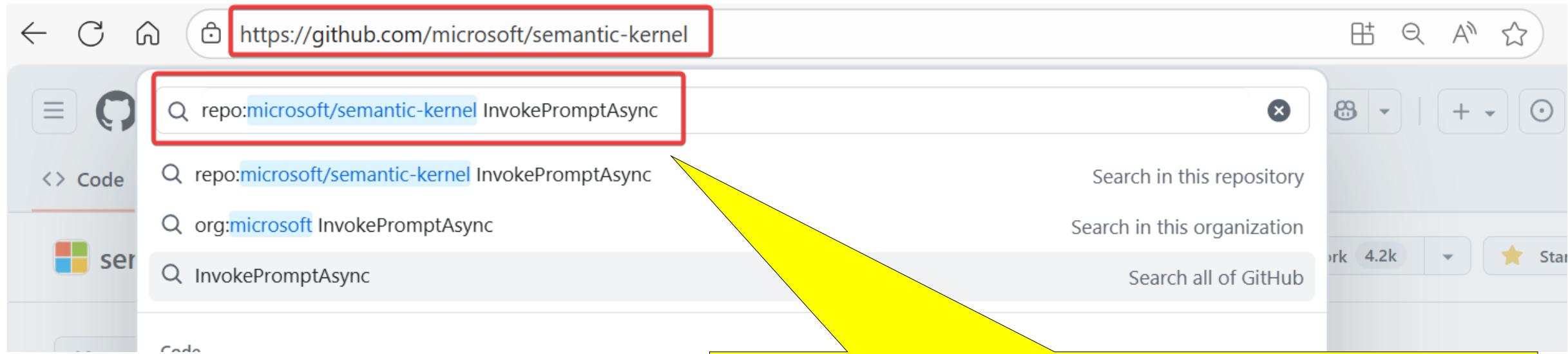
**Search Results:**

- microsoft/semantic-kernel**  
Integrate cutting-edge LLM technology quickly and easily into your apps  
sdk ai artificial-intelligence openai llm  
C# · ⭐ 26k · Updated 1 hour ago
- microsoft/semantic-kernel-starters**  
Starter Projects for Semantic Kernel  
C# · ⭐ 403 · Updated on 5月18日
- MicrosoftDocs/semantic-kernel-docs**  
Semantic Kernel (SK) is a lightweight SDK enabling integration of AI Large Language Models (LLMs) with conventional programming languages.  
Mermaid · ⭐ 233 · Updated 5 days ago
- microsoft/SemanticKernelCookBook**  
This is a Semantic Kernel's book for beginners  
Jupyter Notebook · ⭐ 307 · Updated on 2024年8月6日

**Right sidebar:**

- Sponsor open source projects you depend on**  
Contributors are working behind the scenes to make open source better for everyone—give them the help and recognition they deserve.  
[Explore sponsorable projects →](#)
- How can we improve search?**  
[Give feedback](#)
- ProTip!** Press the **/** key to activate the search input again and adjust your query.

# 特定のリポジトリでの検索の例



特定のリポジトリで検索を行う場合、  
「repo: リポジトリ名」という  
検索修飾子（search qualifier）が検索バーに入力される。  
それに続いて検索キーワード  
(リポジトリ内で検索したいもの) を入力。

# 特定のリポジトリでの検索の例

The screenshot shows a GitHub search interface with the following details:

- Search Query:** repo:microsoft/semantic-kernel InvokePromptAsync
- Results Count:** 110 files (218 ms)
- Filter by:** Code (selected), Issues (71), Pull requests (18), Discussions (21), Commits (7), Packages (0), Wikis (0).
- Languages:** C# (selected), Markdown, Jupyter Notebook, XML.
- Paths:** dotnet/, dotnet/src/, dotnet/samples/, dotnet/samples/Concepts/.

The results are displayed in two sections:

- dotnet/src/SemanticKernel.Core/KernelExtensions.cs:**

```
1216     #region InvokePromptAsync
1238         public static Task<FunctionResult> InvokePromptAsync(
1252             functionName: KernelFunctionFromPrompt.CreateRandomFunctionName(nameof(InvokePromptAsync)),
1282         public static Task<FunctionResult> InvokePromptAsync(
1298             functionName: KernelFunctionFromPrompt.CreateRandomFunctionName(nameof(InvokePromptAsync)),
1328         public static Task<T?> InvokePromptAsync<T>(
1342             functionName: KernelFunctionFromPrompt.CreateRandomFunctionName(nameof(InvokePromptAsync)),
1372         public static Task<T?> InvokePromptAsync<T>(

```

Show -1 more matches
- dotnet/src/SemanticKernel.UnitTests/KernelTests.cs:**

```
378     }
379
380     [Fact]
381     public async Task ValidateInvokePromptAsync()
382     {
383         // Arrange
384         IKernelBuilder builder = Kernel.CreateBuilder();

```

Show 1 more match

# まとめ

- GitHubの検索機能とは？
- GitHub全体、特定のリポジトリ内、などをすばやく検索できる。キーワードを指定して、それを含むリポジトリ（コード）、issues、プルリクエストなどを検索できる。

# 目次

- 1 GitHub の概要
- 2 GitHub Codespaces でのコーディング
- 3 Markdown を使用して GitHub で効果的にコミュニケーションする
- 4 GitHub を利用し、リポジトリ履歴を検索し、整理する
- 5 GitHub で pull request を使用してリポジトリの変更を管理する
- 6 GitHub のベストプラクティスを使用してセキュリティで保護されたリポジトリを維持する
- 7 GitHub のオープンソースプロジェクトに貢献する
- 8 GitHub Projects で仕事を管理する

# モジュール5

- GitHubでのGitの操作
- フォーク
- プルリクエスト

# GitHubでのGitの操作

- ・フォークする
- ・作業用のブランチを作成する
- ・プルリクエストをオープンする
- ・プルリクエストがクローズ（マージ）される
- ・フェッチ＆マージする
- ・作業用ブランチを削除する



- ・ユーザーAが作成した、塩を使った卵焼きのレシピが書かれた紙があった。
- ・ユーザーBがそのレシピを見て「塩じゃなくて砂糖を使ったらどうかな？」と思った。
- ・ただし、ユーザーBはユーザーAと特に面識はなく、
- ・ユーザーAのレシピを勝手に書き換えることはできない。



- ・そこで、ユーザーBはいったんそのレシピを自分の手元にコピーした。
- ・コピーしたレシピの紙は、あとで参考にできるように、そのままにしておき、
- ・「砂糖を使った卵焼き」のレシピは別の紙に書いた。
- ・実際に砂糖を使った卵焼きを作って食べてみたところ美味しかったので、
- ・ユーザーBは思い切って、この変更をユーザーAに提案することにした。



- ・ユーザーAがその変更を試したところ美味しかったので、
- ・ユーザーAはその変更提案を受け入れて、自分のレシピを更新した。



- ・ユーザーBがあとでユーザーAのレシピを確認したところ、
- ・「ユーザーBが塩を砂糖に変更した」「この変更をユーザーAが受け入れた」という記録とともに、レシピが変更されていることがわかった。



- ・ユーザーAが作成した、塩を使った卵焼きのレシピが書かれた紙があった。
- ・ユーザーBがそのレシピを見て「塩じゃなくて砂糖を使ったらどうかな？」と思った。
- ・ただし、ユーザーBはユーザーAと特に面識はなく、
- ・ユーザーAのレシピを勝手に書き換えることはできない。

元のリポジトリの  
mainブランチ



### 自分のリポジトリ

- そこで、ユーザーBはいったんそのレシピを自分の手元にコピーした。
- コピーしたレシピの紙は、あとで参考にできるように、そのままにしておき、
- 「砂糖を使った卵焼き」のレシピは別の紙に書いた。
- 実際に砂糖を使った卵焼きを作って食べてみたところ美味しかったので、
- ユーザーBは思い切って、この変更をユーザーAに提案することにした。

フォーク

mainブランチでは  
作業しない

作業用の  
ブランチを作成

プルリクエストを  
作成



- ユーザーAがその変更を試したところ美味しかったので、
- ユーザーAはその変更提案を受け入れて、自分のレシピを更新した。

プルリクエストを  
マージ



- ユーザーBがあとでユーザーAのレシピを確認したところ、
- 「ユーザーBが塩を砂糖に変更した」「この変更をユーザーAが受け入れた」という記録とともに、レシピが変更されていることがわかった。

2つのコミットが  
記録される

# フォーク (Fork)

- ・他人のGitHubリポジトリを自分のGitHubアカウントにコピーする操作。
- ・主に、バグ修正をしたり機能を追加したりしたい場合に、フォークを行う
- ・他人のGitHubリポジトリに対する変更の権限がなくても、フォークによってコピーされた自分のリポジトリでは、自由に変更ができる。





userA



Fork 3.8k



userA/tamagoyaki

フォーク

userB/tamagoyaki



userB

元リポジトリの「Fork」ボタンを使用。  
元リポジトリのコピーが作られる。  
元リポジトリとのつながりは維持される

元リポジトリをアップストリーム（上流）、  
フォークしてできたりポジトリをダウンストリーム（下流）という。

# プルリクエスト

- ・フォークしたリポジトリでの変更部分は「プルリクエスト」という形で、元のリポジトリに提出することができる。
- ・プルリクエストを作るかどうかはオプション。
- ・元のリポジトリ側では、プルリクエストの内容を確認し、マージする（その変更を受け入れる）か、却下する



userA

userA/tamagoyaki  
main ブランチ

README.md

塩

Pull requests 1

Merge pull request

Confirm merge

(7)userAがプルリクエストをマージする  
(8)プルリクエストがクローズされる

README.md

砂糖

main  
ブランチに  
userBの  
コミットと  
userAの  
マージ  
コミットの  
2つの  
コミットが  
記録される

(2)userBがリポジトリを  
フォークする

(3)userBがブランチを作成する

userB

userB/tamagoyaki  
main ブランチ

README.md

塩

Create pull request

userB/tamagoyaki  
patch-1 ブランチ

README.md

塩

塩  
砂糖

(4)userBが  
編集を行う

README.md

砂糖

(5) プルリクエストを  
作成する  
※(8)の後

This branch is 1 commit  
**behind** userA/repo1



(10)userBが  
patch-1ブランチを  
削除する

※(8)の後

This branch is 2 commit **behind** userA/repo1

(9)userBが  
フェッチ&  
マージする

Fetch and merge

README.md

砂糖

# まとめ

- ・プルリクエスト
- ・多数のユーザーでの共同開発を行うための、GitHubの最も重要な機能。リポジトリをフォーク（コピー）し、新しいブランチを作成して、そこでコードの修正作業などを行う。変更部分をプルリクエストとして元のリポジトリに送信。元のリポジトリのオーナーはプルリクエストの内容を確認し、それをマージ（取り込み）できる。

# 目次

- 1 GitHub の概要
- 2 GitHub Codespaces でのコーディング
- 3 Markdown を使用して GitHub で効果的にコミュニケーションする
- 4 GitHub を利用し、リポジトリ履歴を検索し、整理する
- 5 GitHub で pull request を使用してリポジトリの変更を管理する
- 6 GitHub のベストプラクティスを使用してセキュリティで保護されたリポジトリを維持する
- 7 GitHub のオープンソースプロジェクトに貢献する
- 8 GitHub Projects で仕事を管理する

# モジュール6

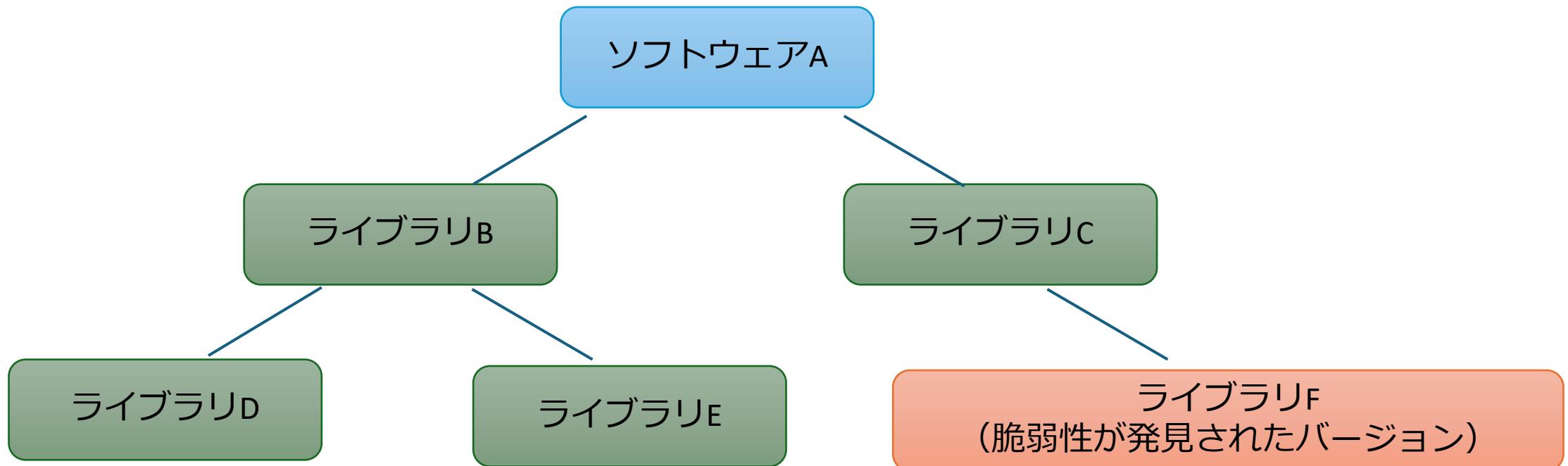
- Dependabotによる脆弱性の検査
- .gitignoreの使用
- シークレットスキャン
- SECURITY.mdへのセキュリティポリシーの記載
- GitHub Code Security

# モジュール6

- Dependabotによる脆弱性の検査
- .gitignoreの使用
- シークレットスキャン
- SECURITY.mdへのセキュリティポリシーの記載
- GitHub Code Security

# Dependabotによる脆弱性の検査

- Dependabot は、**依存関係の脆弱性をスキャンし、修正提案を行うことで、ソフトウェアのセキュリティを向上させるしくみ**
- 依存関係 (Dependency) : ソフトウェアで使用しているライブラリなど。





# GitHub security alert digest

's repository security updates from  
the week of Aug 26 - Sep 2

's personal account



## Known security vulnerabilities detected

Dependency	Version	Upgrade to
org.apache.logging.log4j:log4j-j-core	>= 2.13.0	~> 2.16.0
	< 2.16.0	

Defined in

[pom.xml](#)

### Vulnerabilities

- CVE-2021-45046 Critical severity
- CVE-2021-45046 Critical severity
- CVE-2021-44228 Critical severity
- CVE-2021-44228 Critical severity
- CVE-2021-45105 High severity

[View 3 more](#)

[Review all vulnerable dependencies](#)

# モジュール6

- Dependabotによる脆弱性の検査
- .gitignoreの使用
- シークレットスキャン
- SECURITY.mdへのセキュリティポリシーの記載
- GitHub Code Security

# .gitignoreの使用

- ・.gitignore は Git を使ったバージョン管理において、**追跡対象から除外したいファイルやディレクトリを指定するための設定ファイル**。
- ・Git では通常、以下のようなファイルはバージョン管理する必要はない
  - ・ビルド成果物（例：\*.class, \*.exe, dist/）
  - ・一時ファイル（例：\*.log, \*.tmp）
  - ・IDEやエディタの設定ファイル（例：.vscode/, .idea/）
  - ・**秘密情報（例：.env, config/secrets.yml）**
- ・これらをGitのバージョン管理から除外するように設定する。

# .gitignore ファイルの設定例

```
# Python のキャッシュファイル  
__pycache__/  
*.pyc  
  
# Node.js の依存パッケージ  
node_modules/  
  
# 環境変数ファイル  
.env  
  
# ログファイル  
*.log  
  
# OSごとの不要ファイル  
.DS_Store
```

# モジュール6

- Dependabotによる脆弱性の検査
- .gitignoreの使用
- シークレットスキヤン
- SECURITY.mdへのセキュリティポリシーの記載
- GitHub Code Security

# シークレットスキャン

- ・シークレットスキャンは、API キー、パスワード、トークン、他のシークレットなどの機密情報が誤ってリポジトリに含まれることを検出して防ぐのに役立つセキュリティ機能
- ・リポジトリ内のコミットをスキャンする
- ・シークレットを検出すると、シークレットスキャンアラートが生成される
- ・アラートは、GitHub のリポジトリの **[Security]** タブで確認できる（リポジトリの管理者のみこのタブの内容を確認でき、第三者が見ることはできない）

# モジュール6

- Dependabotによる脆弱性の検査
- .gitignoreの使用
- シークレットスキャン
- SECURITY.mdへのセキュリティポリシーの記載
- GitHub Code Security

# SECURITY.mdへのセキュリティポリシーの記載

- SECURITY.md は、GitHubリポジトリでホスティングされているソフトウェアのセキュリティ関連情報を記載するファイル
- 各GitHubリポジトリの直下に作成する
- 脆弱性を発見した際の報告先、報告のガイドライン（報告に含めるべき情報など）、報告を受けた際の対応についてのポリシーなどが記載される
- 必ずしも作成する必要はないが、あったほうがよい

# 安全

---

Microsoft は、[Microsoft](#)、[Azure](#)、[DotNet](#)、[AspNet](#)、[Xamarin](#)、[GitHub 組織](#)を含む GitHub 組織を通じて管理されるすべてのソース コード リポジトリを含む、ソフトウェア製品とサービスのセキュリティを真剣に受け止めています。

Microsoft が所有するリポジトリで、[Microsoft のセキュリティ脆弱性の定義](#)を満たすセキュリティの脆弱性が見つかったと思われる場合は、以下に説明するように報告してください。

## セキュリティ問題の報告

---

公開されている GitHub の問題を通じてセキュリティの脆弱性を報告しないでください。

代わりに、<https://msrc.microsoft.com/create-report> の Microsoft セキュリティ対応センター (MSRC) に報告してください。

ログインせずに送信する場合は、[secure@microsoft.com](mailto:secure@microsoft.com) にメールを送信してください。可能であれば、PGPキーでメッセージを暗号化してください。[Microsoft Security Response Center の PGP キー ページ](#)からダウンロードしてください。

24 時間以内に応答が届きます。何らかの理由で通知されない場合は、電子メールでフォローアップして、元のメッセージが届いたことを確認してください。詳細については、[microsoft.com/msrc](https://microsoft.com/msrc) をご覧ください。

考えられる問題の性質と範囲をよりよく理解できるように、以下にリストされている要求された情報を(提供できる限り)含めてください。

- 問題の種類 (バッファ オーバーフロー、SQL インジェクション、クロスサイト スクリプティングなど)
- 問題の発生に関連するソースファイルのフルパス
- 影響を受けるソースコードの場所(タグ/ブランチ/コミットまたは直接URL)
- 問題を再現するために必要な特別な構成
- 問題を再現するためのステップバイステップの手順
- 概念実証またはエクスプロイト コード(可能な場合)
- 攻撃者が問題を悪用する方法など、問題の影響

この情報は、レポートをより迅速にトリアージするのに役立ちます。

バグ報奨金を報告する場合は、より完全な報告がより高い報奨金に貢献できます。アクティブなプログラムの詳細については、[Microsoft バグ報奨金プログラム](#)のページをご覧ください。

## 優先言語

---

私たちは、すべてのコミュニケーションを英語で行うことを望んでいます。

## 政策

---

Microsoft は、[調整された脆弱性開示](#)の原則に従います。

# モジュール6

- Dependabotによる脆弱性の検査
- .gitignoreの使用
- シークレットスキャン
- SECURITY.mdへのセキュリティポリシーの記載
- GitHub Code Security

# GitHub Code Securityの 「Code scanning」と「Copilot Autofix」

- Code scanning（コードスキャン）
  - コードのセキュリティ脆弱性とコーディングエラーを自動的に検出
  - コードに潜在的な脆弱性またはエラーを見つけた場合、リポジトリにアラートが表示される
- GitHub Copilot Autofix
  - code scanning 分析で検出されたアラートに対する修正を生成して提案
  - GitHub Copilot Autofixが修正を提案できる場合は「Generate fix」ボタンが出現。それをクリックすると修正がPull requestとして作成される。

# まとめ

- GitHubのセキュリティ機能
- (1) 「Dependabot」は、依存関係（リポジトリで使用しているライブラリなど）をスキャンし、脆弱性があるライブラリを使用している場合はDependabotから通知が飛ぶ。
- (2) 「.gitignore」ファイルを使用して、Gitバージョン管理から除外する（つまりGitHubリポジトリに入れないと）ファイルを指定する。
- (3) 「シークレットスキャン」では、リポジトリにキーやパスワードが送信された場合、アラートを発報してリポジトリの管理者に知らせる。
- (4) 「SECURITY.md」は、GitHubリポジトリのセキュリティ関連情報を記載するファイルで、脆弱性発見時の報告先や報告方法などの情報を含める。
- (5) 「GitHub Code Security」では、リポジトリのコードのセキュリティ脆弱性とコーディングエラーを自動的に検出し、アラートを発報してしてリポジトリの管理者に知らせる。GitHub Copilot Autofixを使用するとその脆弱性を修正する提案が生成される。

# 目次

- 1 GitHub の概要
- 2 GitHub Codespaces でのコーディング
- 3 Markdown を使用して GitHub で効果的に通信する
- 4 GitHub を利用し、リポジトリ履歴を検索し、整理する
- 5 GitHub で pull request を使用してリポジトリの変更を管理する
- 6 GitHub のベストプラクティスを使用してセキュリティで保護されたり  
ポジトリを維持する
- 7 GitHub のオープンソースプロジェクトに貢献する
- 8 GitHub Projectsで仕事を管理する

# モジュール7

- ・オープンソースプロジェクトでの共同開発
- ・クローン
- ・コミット
- ・プッシュ
- ・プル
- ・GitHub flow
- ・オープンソースプロジェクトの推奨プラクティス

# モジュール7

- ・オープンソースプロジェクトでの共同開発
- ・クローン
- ・コミット
- ・プッシュ
- ・プル
- ・GitHub flow
- ・オープンソースプロジェクトの推奨プラクティス

# オープンソースプロジェクトでの共同開発

- ・オープンソースプロジェクト開発で使用されるGitコマンド
  - ・クローン (git clone)
  - ・コミット (git commit)
  - ・プッシュ (git push)
  - ・プル (git pull)

# モジュール7

## ・オープンソースプロジェクトでの共同開発

- ・**クローン**

- ・コミット

- ・プッシュ

- ・プル

- ・GitHub flow

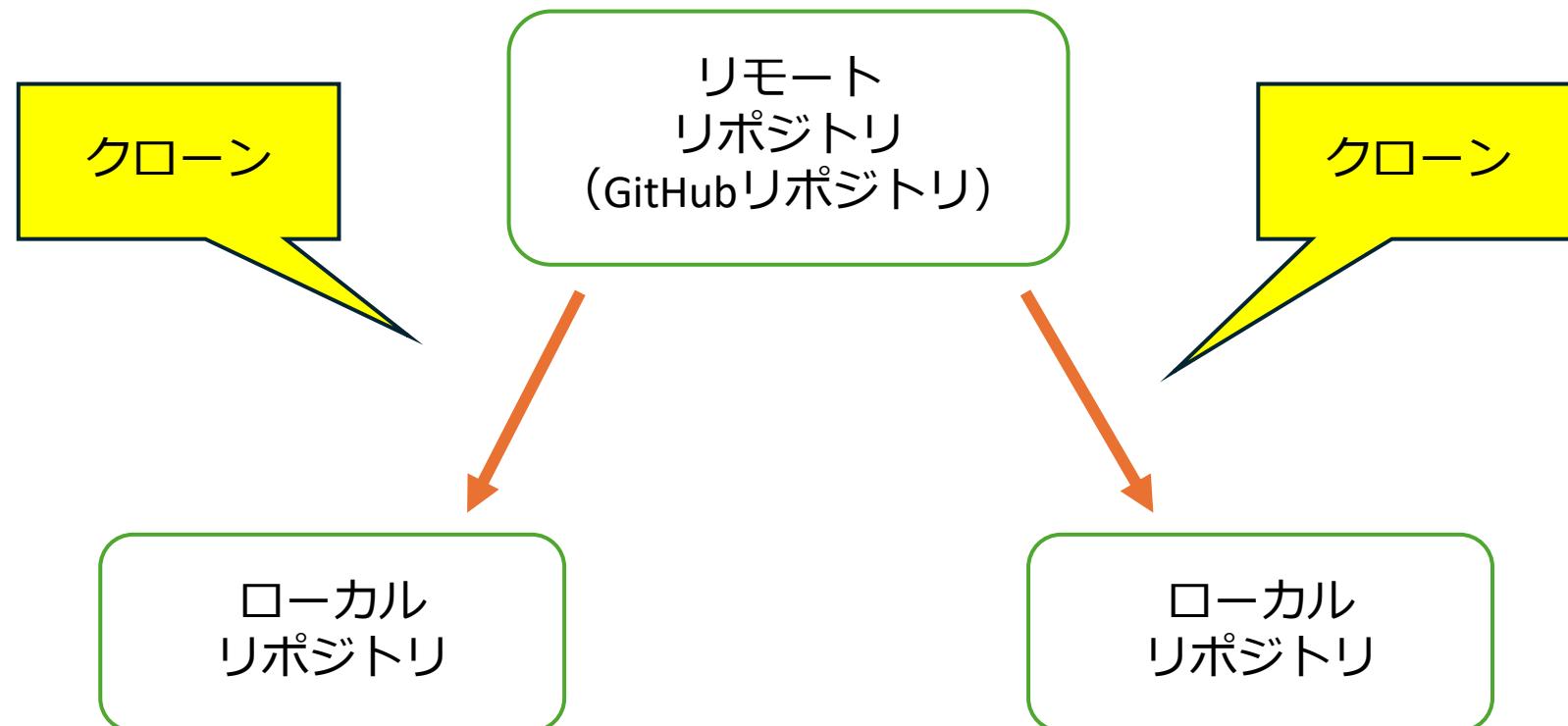
- ・オープンソースプロジェクトの推奨プラクティス

リポジトリへの書き込み権限がある場合、クローン（git clone）を使用。たとえばオープンソースのプロジェクトに直接参加している開発者メンバーはクローンを使用する。

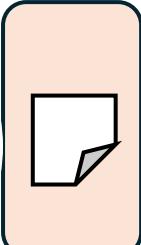
リポジトリへの書き込み権限がない場合、フォーク（GitHubの他人のリポジトリを自分のGitHubアカウント内にコピー）を使用。たとえばオープンソースのソフトウェアに対し、バグ修正や機能追加などの提案を行いたい場合は、フォークを使用する。

# クローン (git clone)

- ・リモートリポジトリのコピーを作成する
- ・作業用のローカルリポジトリができる



- ・変更作業中



リモート  
リポジトリ  
(GitHubリポジトリ)

ローカル  
リポジトリ

ローカル  
リポジトリ

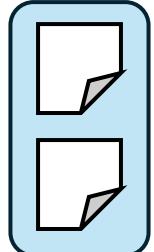
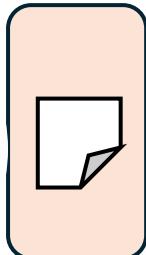


- ・変更作業中

リモート  
リポジトリ  
(GitHubリポジトリ)

ローカル  
リポジトリ

ローカル  
リポジトリ

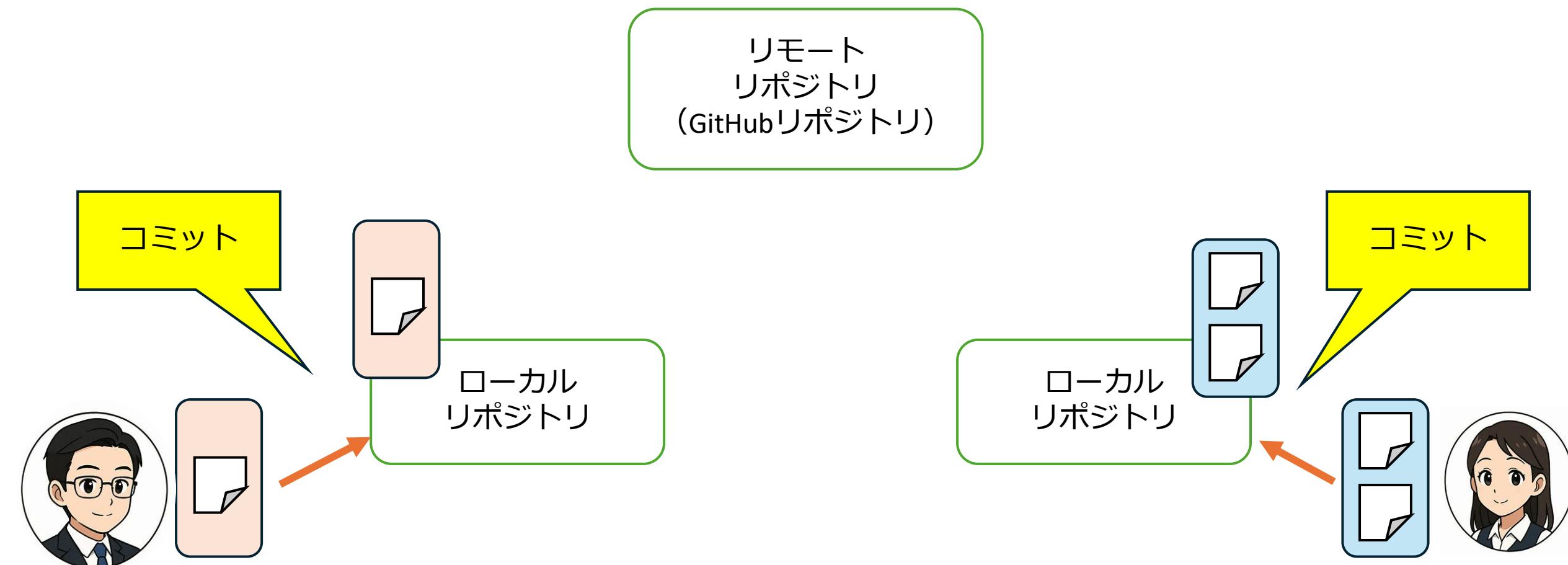


# モジュール7

- ・オープンソースプロジェクトでの共同開発
- ・クローン
- ・コミット
- ・プッシュ
- ・プル
- ・GitHub flow
- ・オープンソースプロジェクトの推奨プラクティス

# コミット (git commit)

- ファイルの追加・変更・削除をローカルリポジトリで登録・確定する処理

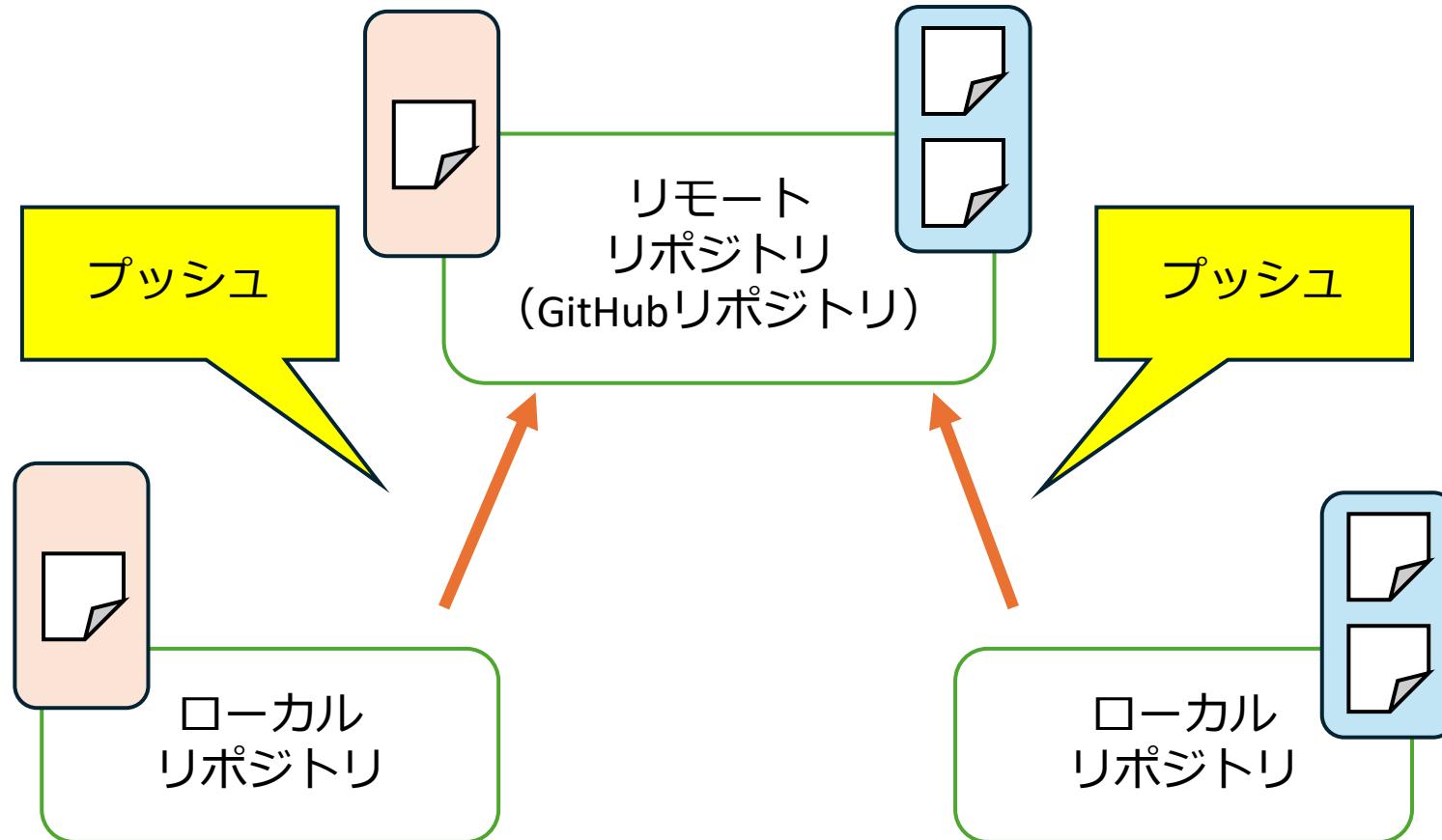


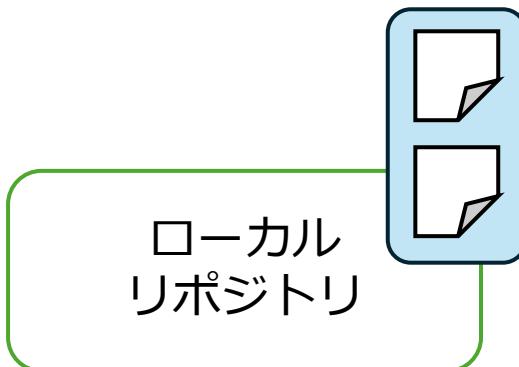
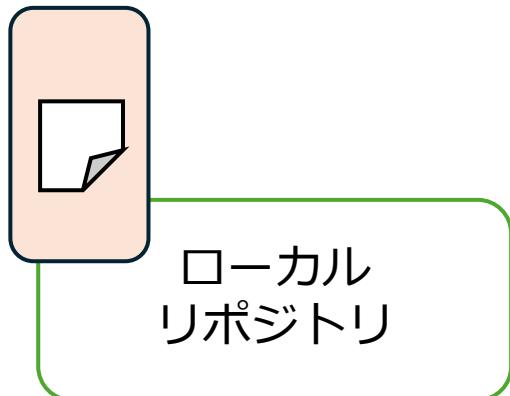
# モジュール7

- ・オープンソースプロジェクトでの共同開発
- ・クローン
- ・コミット
- ・プッシュ (Red Box)
- ・プル
- ・GitHub flow
- ・オープンソースプロジェクトの推奨プラクティス

# プッシュ (git push)

- ・ローカルのコミット（群）を、リモートへ送信する



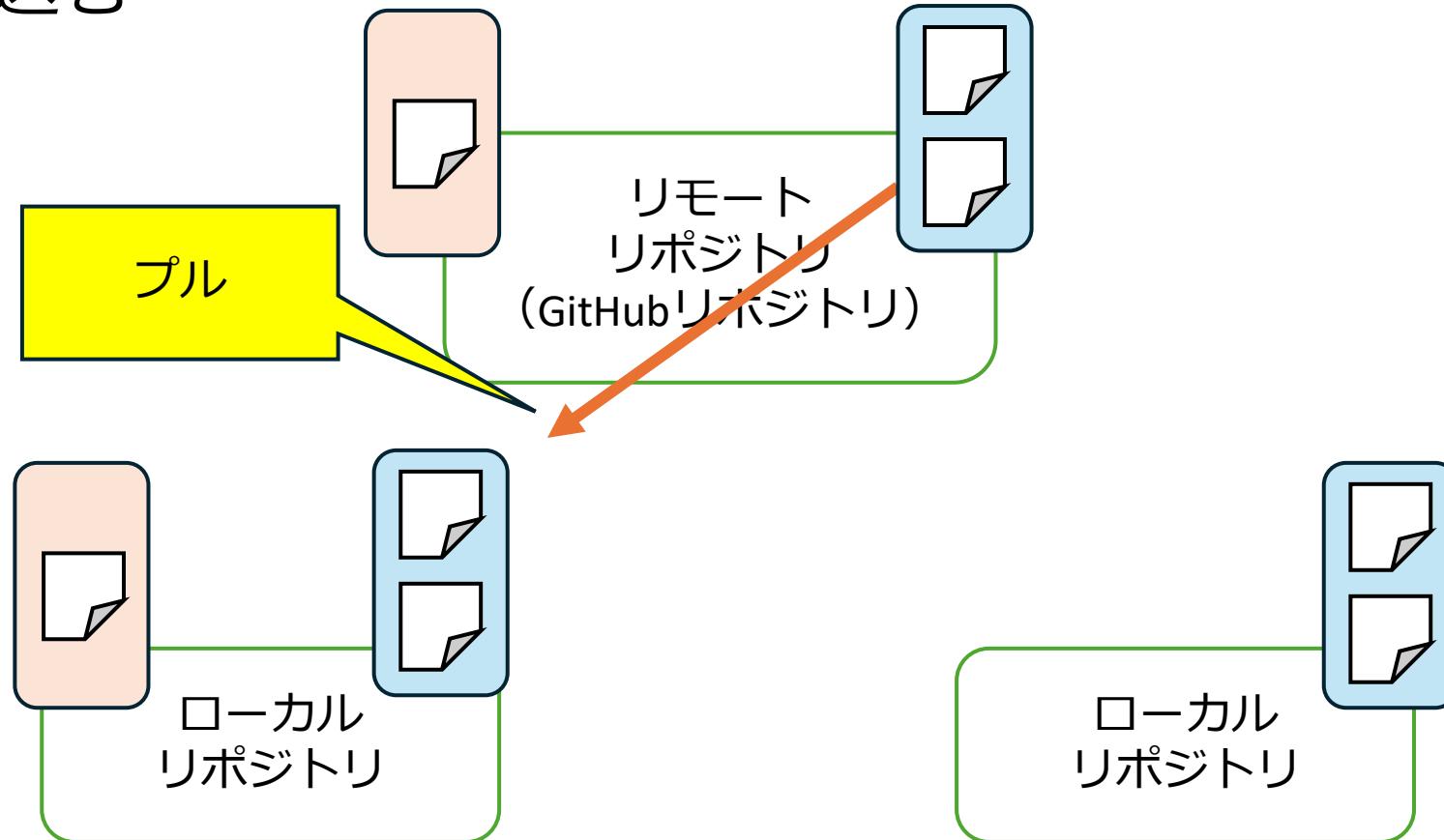


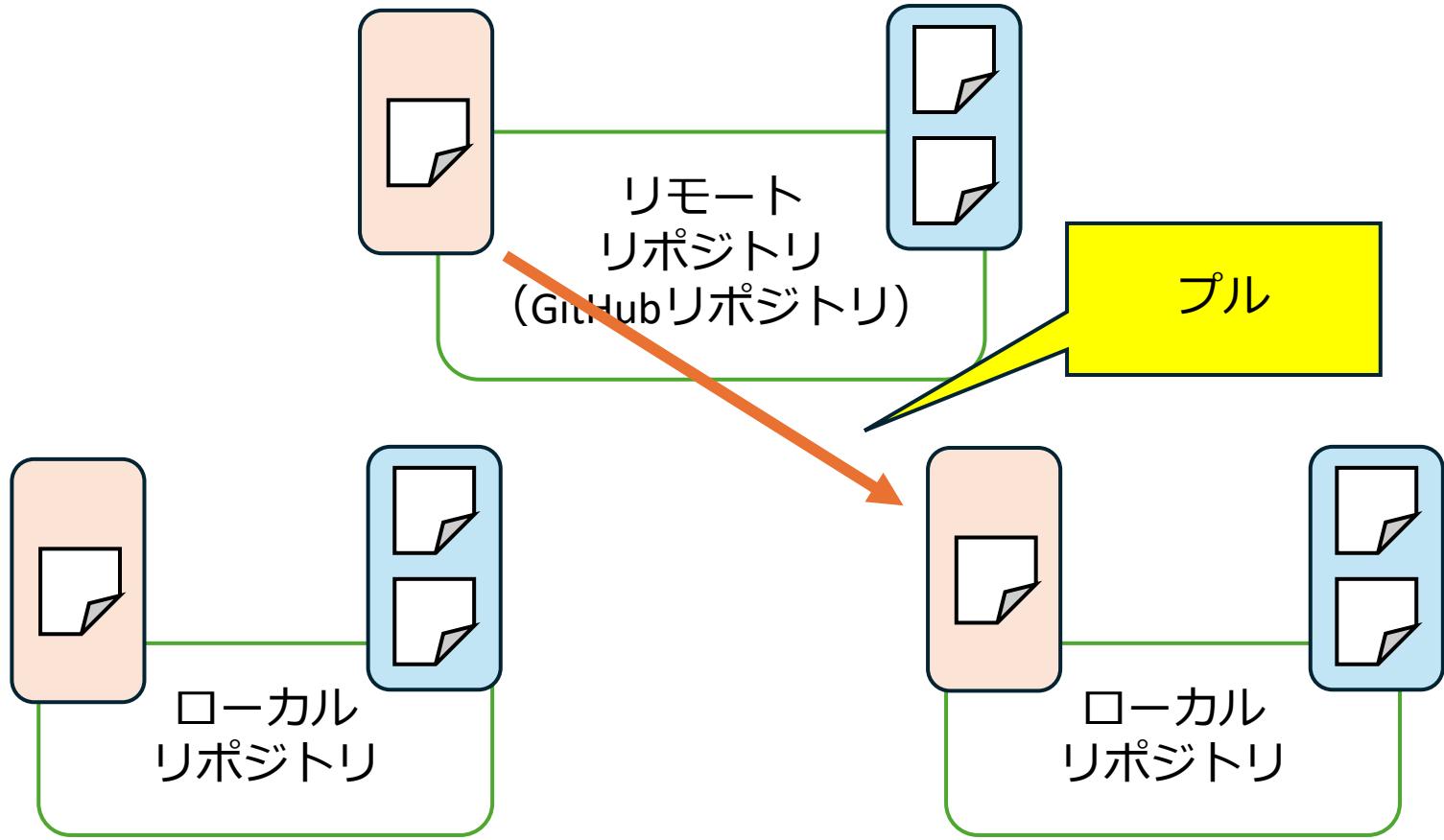
# モジュール7

- ・オープンソースプロジェクトでの共同開発
- ・クローン
- ・コミット
- ・プッシュ
- ・プル
- ・GitHub flow
- ・オープンソースプロジェクトの推奨プラクティス

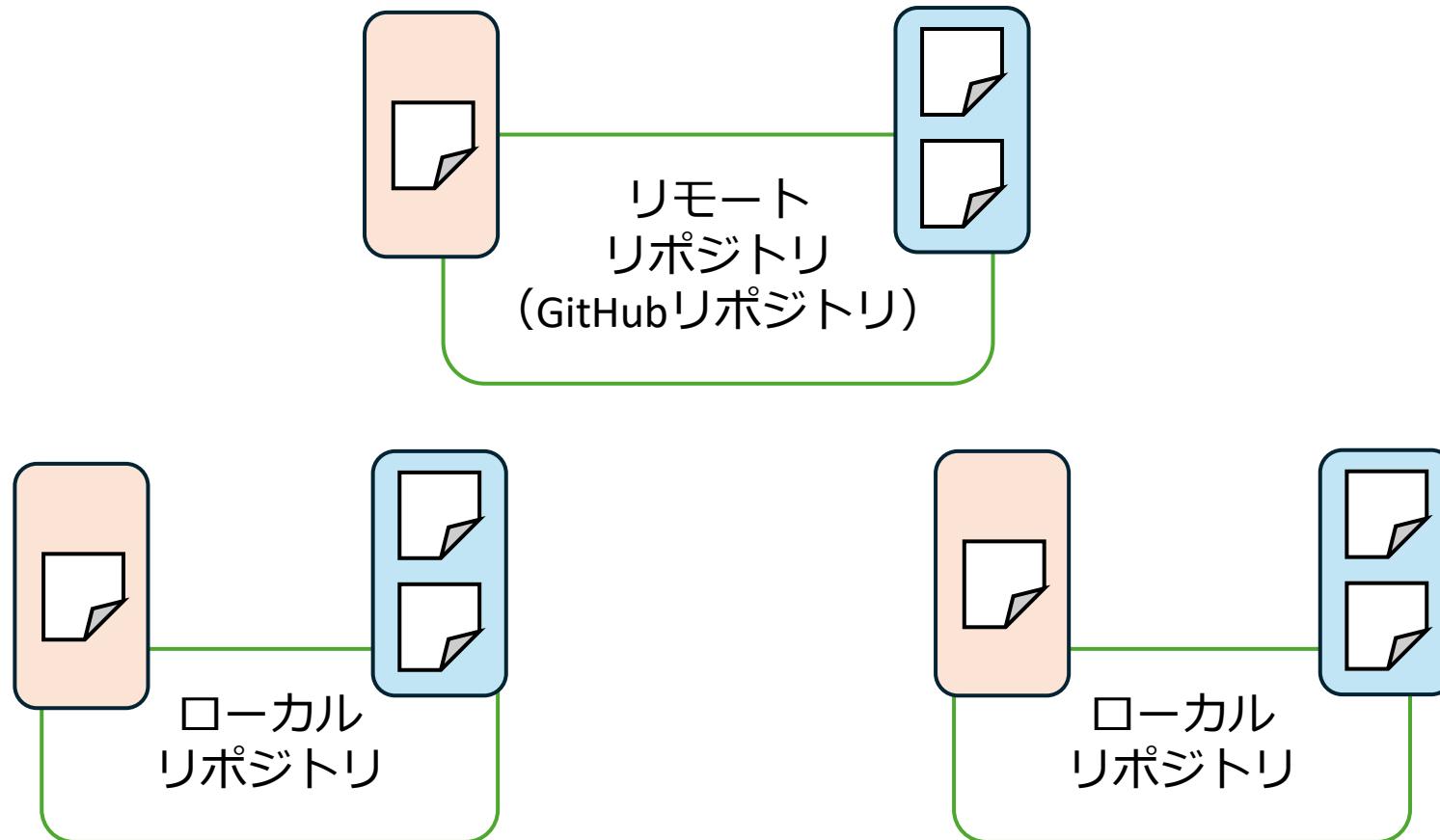
# プル (git pull)

- リモートのコミット（群）をまとめて自分のローカルリポジトリに取り込む





- ・最新化された！



# モジュール7

- ・オープンソースプロジェクトでの共同開発
- ・クローン
- ・コミット
- ・プッシュ
- ・プル
- ・GitHub flow
- ・オープンソースプロジェクトの推奨プラクティス

# GitHub flow

- GitHub社によって開発されたワークフロー（Gitを使った作業の手順）
- 軽量（lightweight、シンプルであるという意味）
- 比較的理解しやすい

# GitHub flow の基本操作

操作	意味
フォーク	元のユーザーが所有するリポジトリのコピーを作成する
ブランチ作成	フォークしたリポジトリ内で新しい作業用のブランチを作成する
変更・コミット	ブランチ上で必要な変更を行い、コミットする
プッシュ	コミット（群）を、フォークした自分のリポジトリへプッシュする
プルリクエスト作成	元のユーザー、変更内容のフィードバックを求めるためのリクエストを作成
プルリクエストのマージ	元のリポジトリのユーザーが、プルリクエストをマージし、変更を反映させる
ブランチ削除	作業用のブランチを削除する

# モジュール7

- ・オープンソースプロジェクトでの共同開発
- ・クローン
- ・コミット
- ・プッシュ
- ・プル
- ・GitHub flow
- ・オープンソースプロジェクトの推奨プラクティス

# 参考: オープンソースプロジェクトを GitHub でホスト & 管理するための推奨プラクティス

- [LF-AI-DATA\\_Managing-OS-Projects-on-Github\\_ja.pdf](#)
- オープンソースプロジェクトの運営方針、リポジトリの設定、GitHub のセキュリティ機能などについて、**具体的なアドバイス**が紹介されている
- GitHub の基本機能やリポジトリの運用方法の基本を確認するために役立つ



# まとめ

- Gitを使った共同開発
- オープンソースの開発に参加する場合は事前にリポジトリをフォークする。それから、そのリポジトリのクローンを行う（リポジトリのコピーを開発環境へとコピー）。クローン後、変更点をローカルのリポジトリにコミットし、リモートのリポジトリ（GitHub上）へとプッシュ（git push）する。クローン後、リモートのリポジトリ（GitHub）の変更点をローカルのリポジトリに取り込むにはプル（git pull）する。
- 「GitHub flow」はこれらの、フォーク、ブランチ作成、変更・コミット、プルリクエストの作成などのシンプルな手順を定めたもの。

# 目次

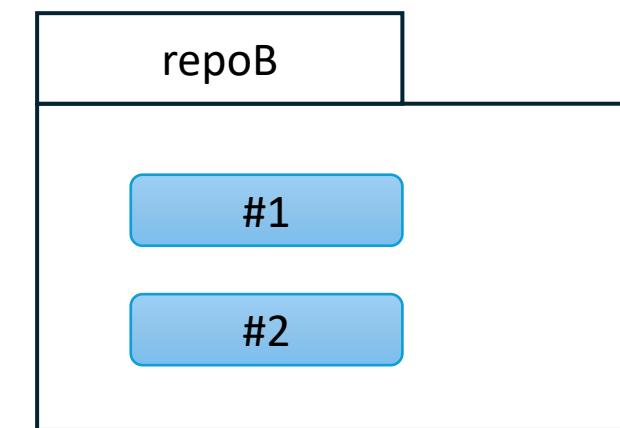
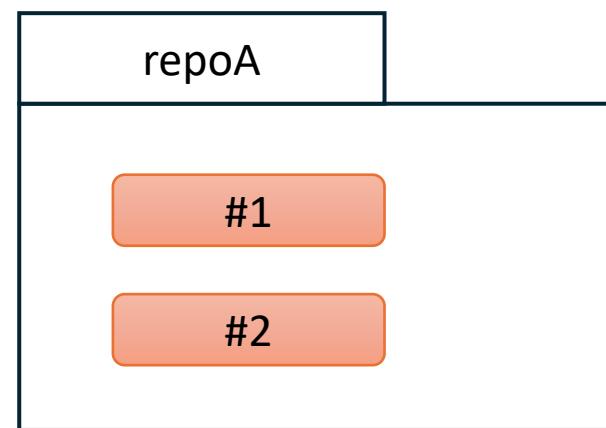
- 1 GitHub の概要
- 2 GitHub Codespaces でのコーディング
- 3 Markdown を使用して GitHub で効果的にコミュニケーションする
- 4 GitHub を利用し、リポジトリ履歴を検索し、整理する
- 5 GitHub で pull request を使用してリポジトリの変更を管理する
- 6 GitHub のベストプラクティスを使用してセキュリティで保護されたり ポジトリを維持する
- 7 GitHub のオープンソースプロジェクトに貢献する
- 8 GitHub Projectsで仕事を管理する

# GitHubを使用したプロジェクト管理

- プロジェクトを作る
- Item (IssueとDraft) を作る (Board ビューの利用)
- Tableビューの利用

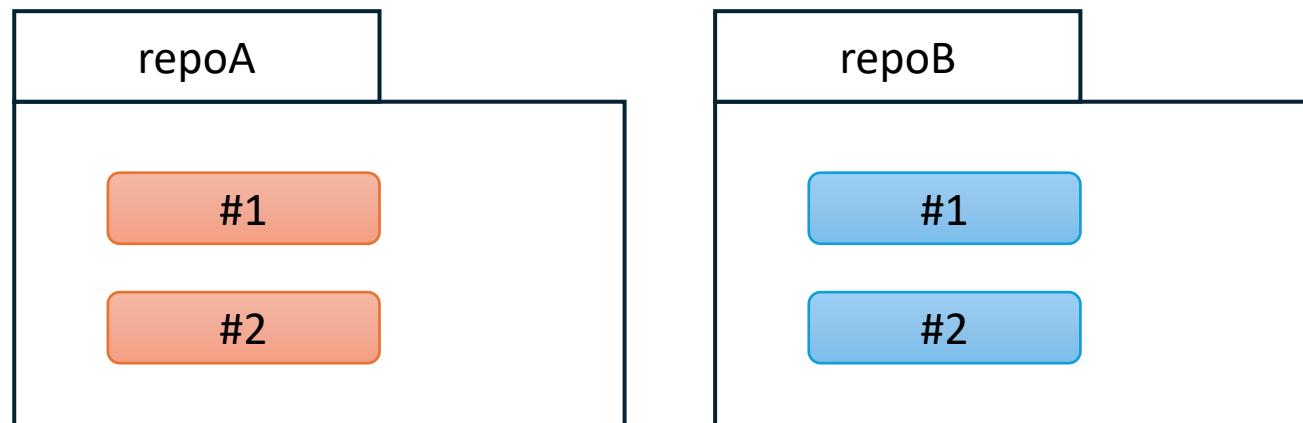
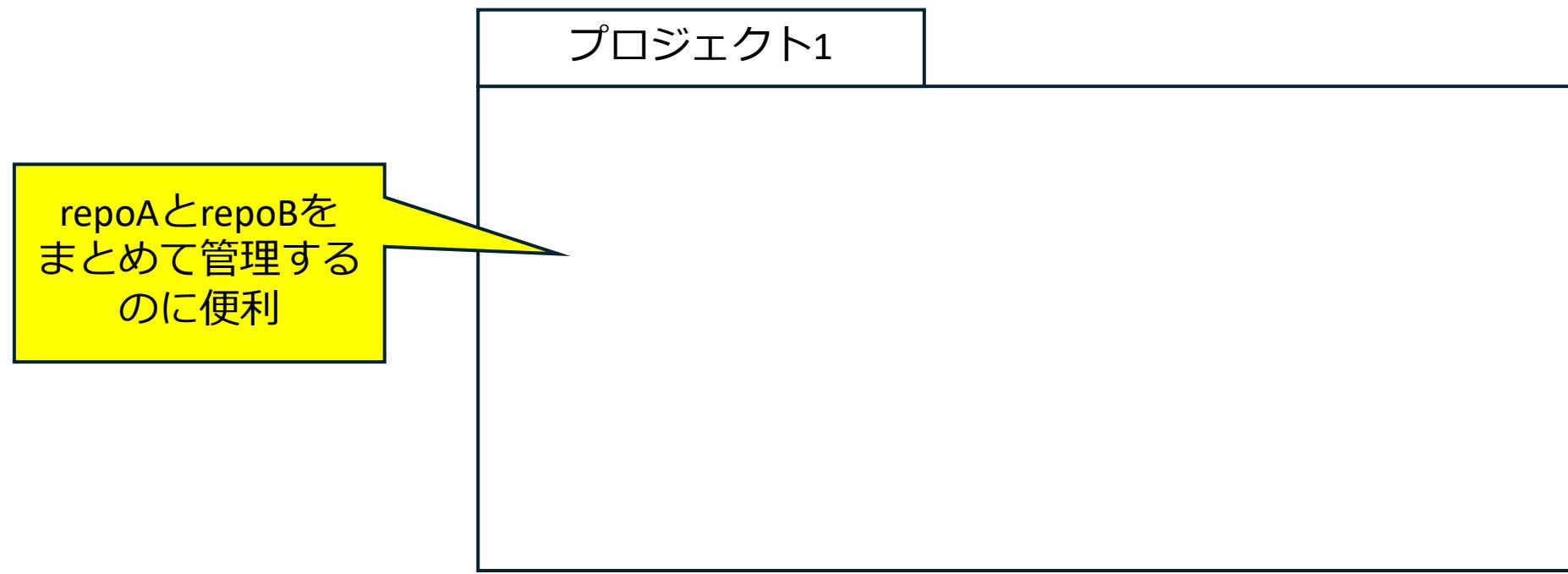
■2つのGitHubリポジトリ「repoA」と「repoB」があり、それぞれにIssueが作られている。

repoAとrepoBの関連性が高い場合、これらをまとめて管理したい・・・

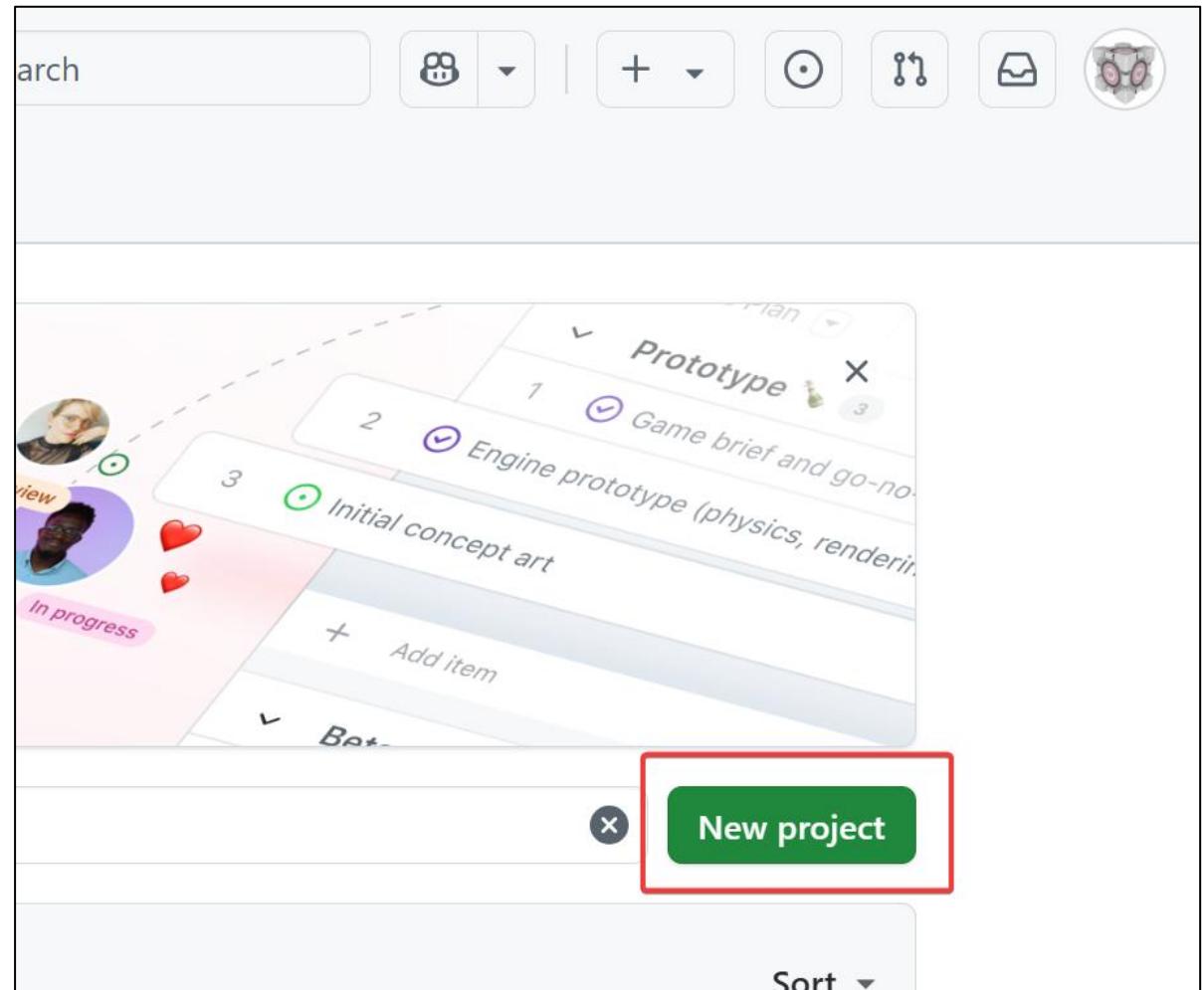
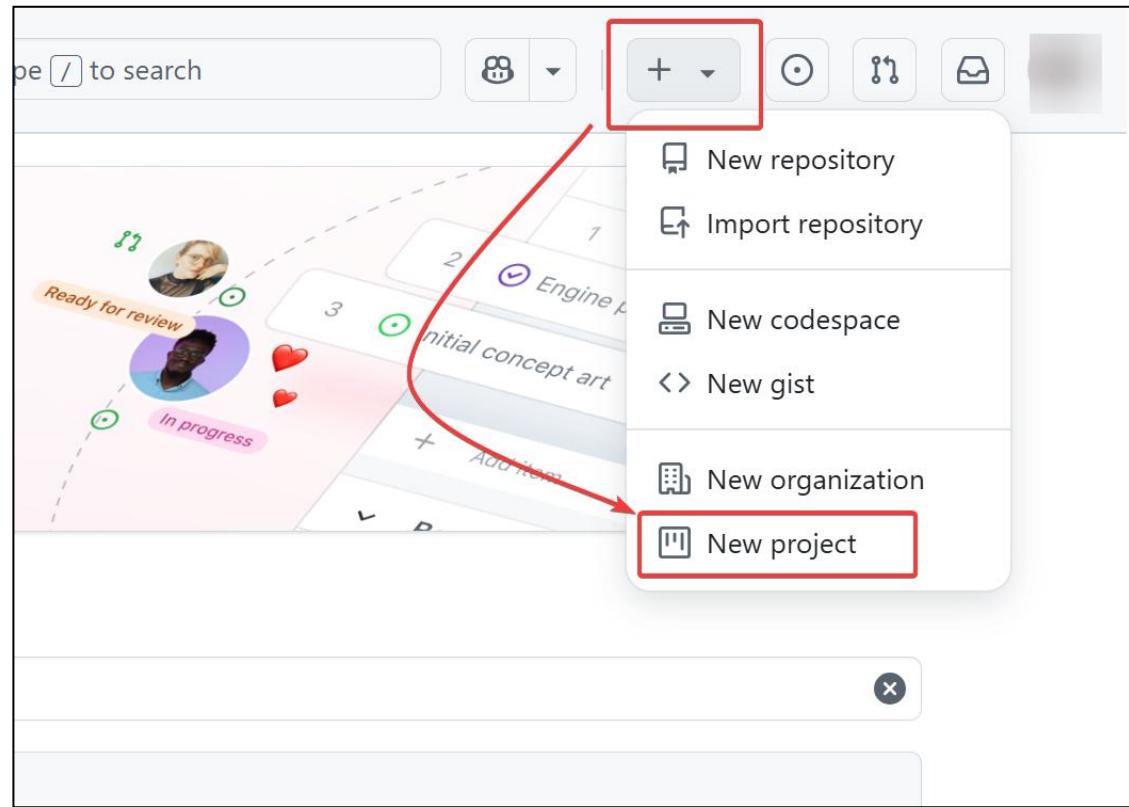


■ GitHub Projects の「プロジェクト」は、複数のリポジトリのIssueをまとめて管理するためのしくみ。

タスクを表す付箋を貼り付ける「ホワイトボード」のようなもの、と考えるとよい



■GitHubの画面上部の「+」をクリックし、New projectをクリック。緑色の「New Project」をクリック。



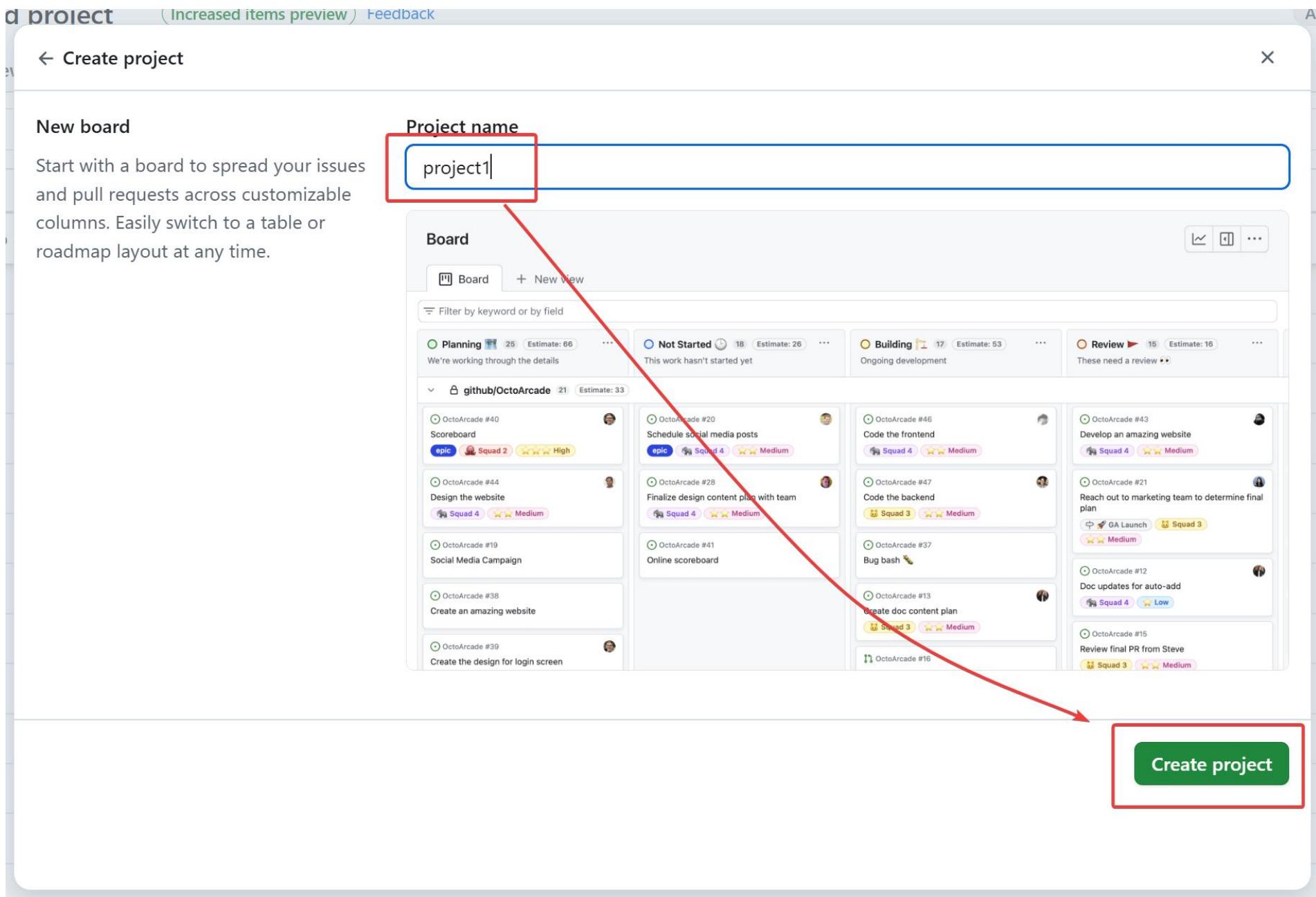
# ■いくつかの「テンプレート」が用意されているが、ここではまずシンプルな「Board」で練習してみよう

The screenshot shows the GitHub 'Create project' interface with the 'Project templates' tab selected. A search bar at the top right says 'Search templates'. Below it, the 'Featured' tab is highlighted. On the left, there are category links: 'Start from scratch', 'Table', 'Board' (which is highlighted with a red box), and 'Roadmap'. To the right of these are four detailed preview cards for different board types:

- Team planning • GitHub**  
Manage your team's work items, plan upcoming cycles, and understand team capacity
- Feature release • GitHub**  
Manage your team's prioritized work items when planning for a feature release
- Kanban • GitHub**  
Visualize the status of your project and limit work in progress
- Bug tracker • GitHub**  
Track and triage your bugs

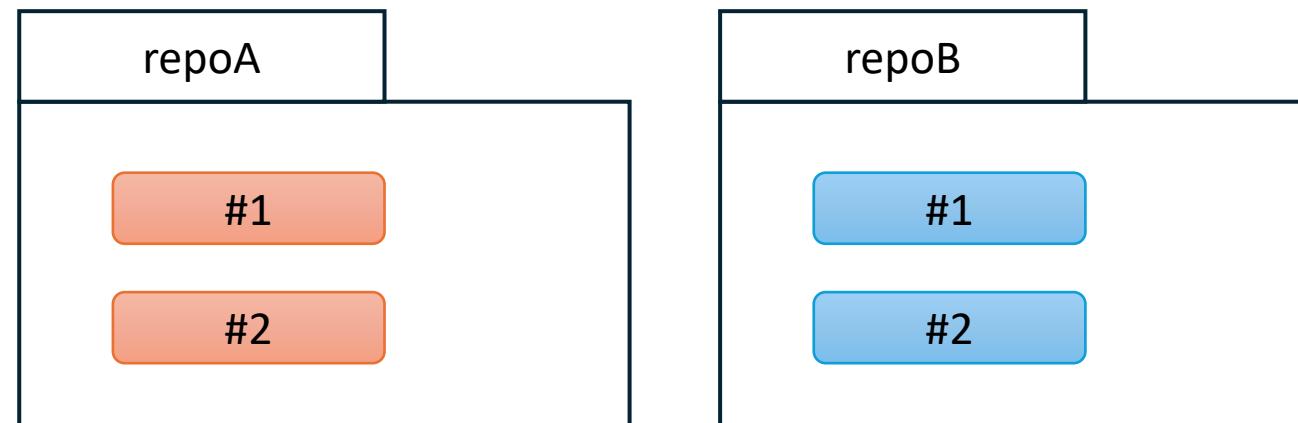
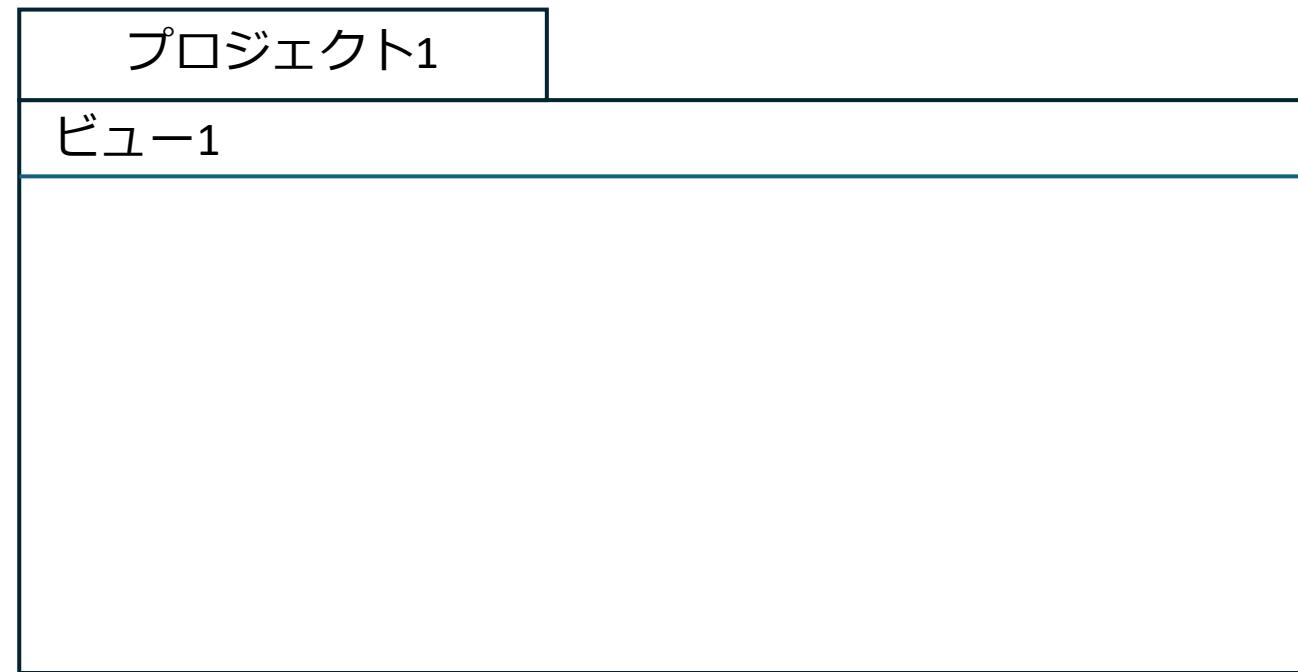
Each preview card shows a sample board with columns like Backlog, In Progress, Done, and In review, along with some sample items.

## ■プロジェクトの名前を適当に入力して、「Create project」をクリック

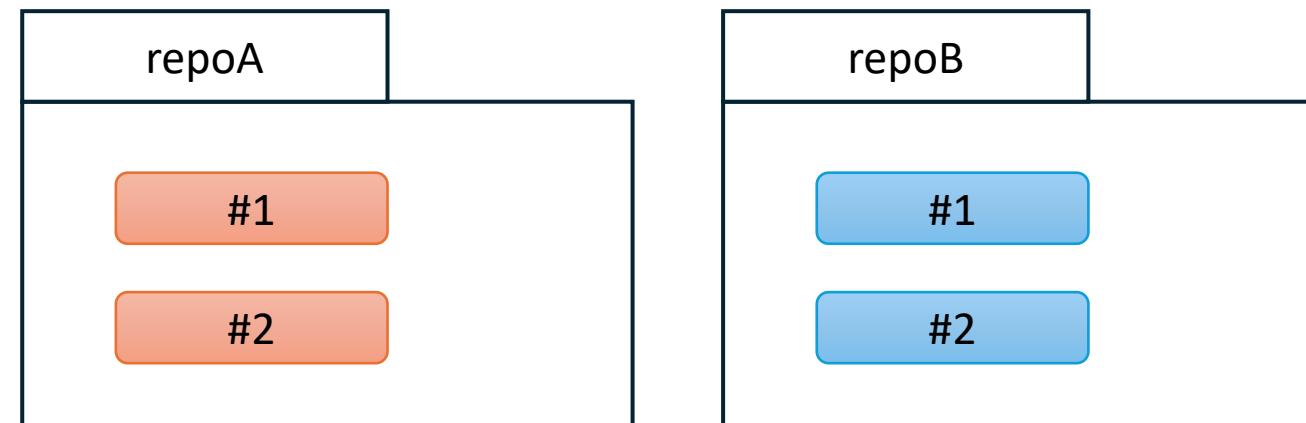
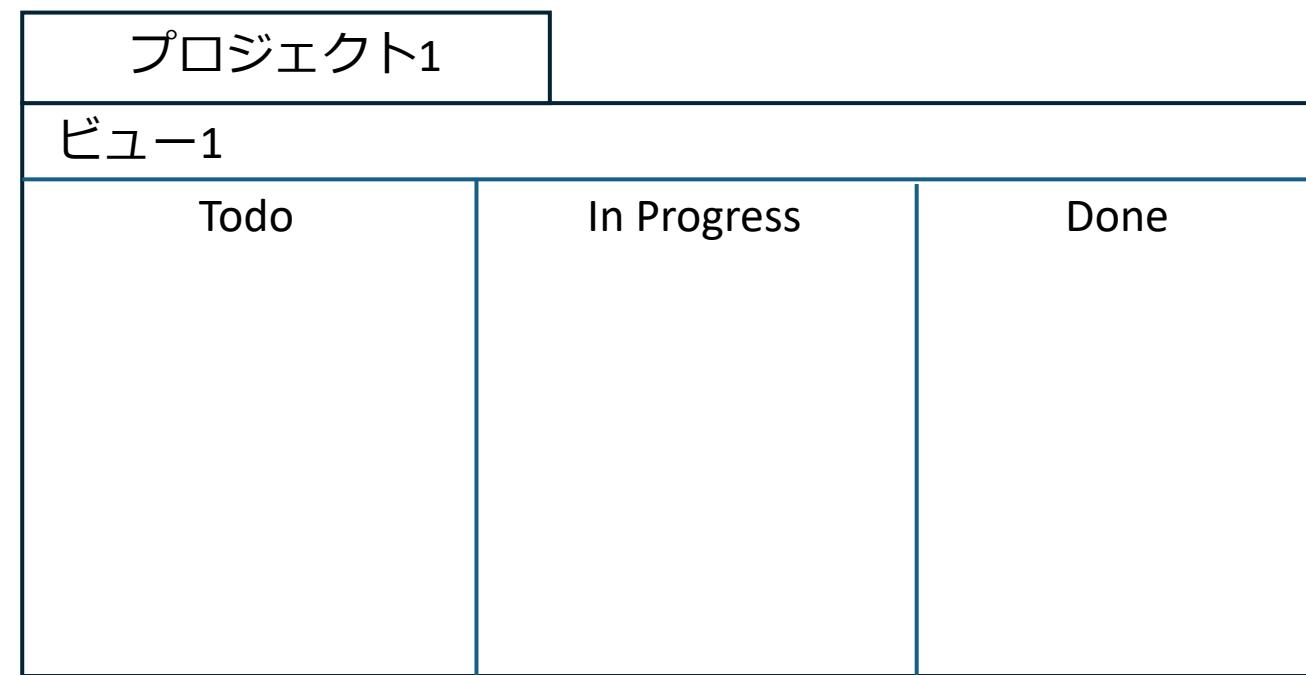


## ■プロジェクトが作られた。

プロジェクトの中には「ビュー」というものが作られる。これは「表示モード」的なものである。  
ビューはプロジェクトの中に複数作ることができる。初期状態では「ビュー1」が作られている。



- ここで、「ビュー1」では「**ボード**」（Board）という「**レイアウト**」が選択されている。  
「**ボード**」は Issue の「Todo」「In Progress」「Done」の3つの状態（Status）を視覚化するために使用される。

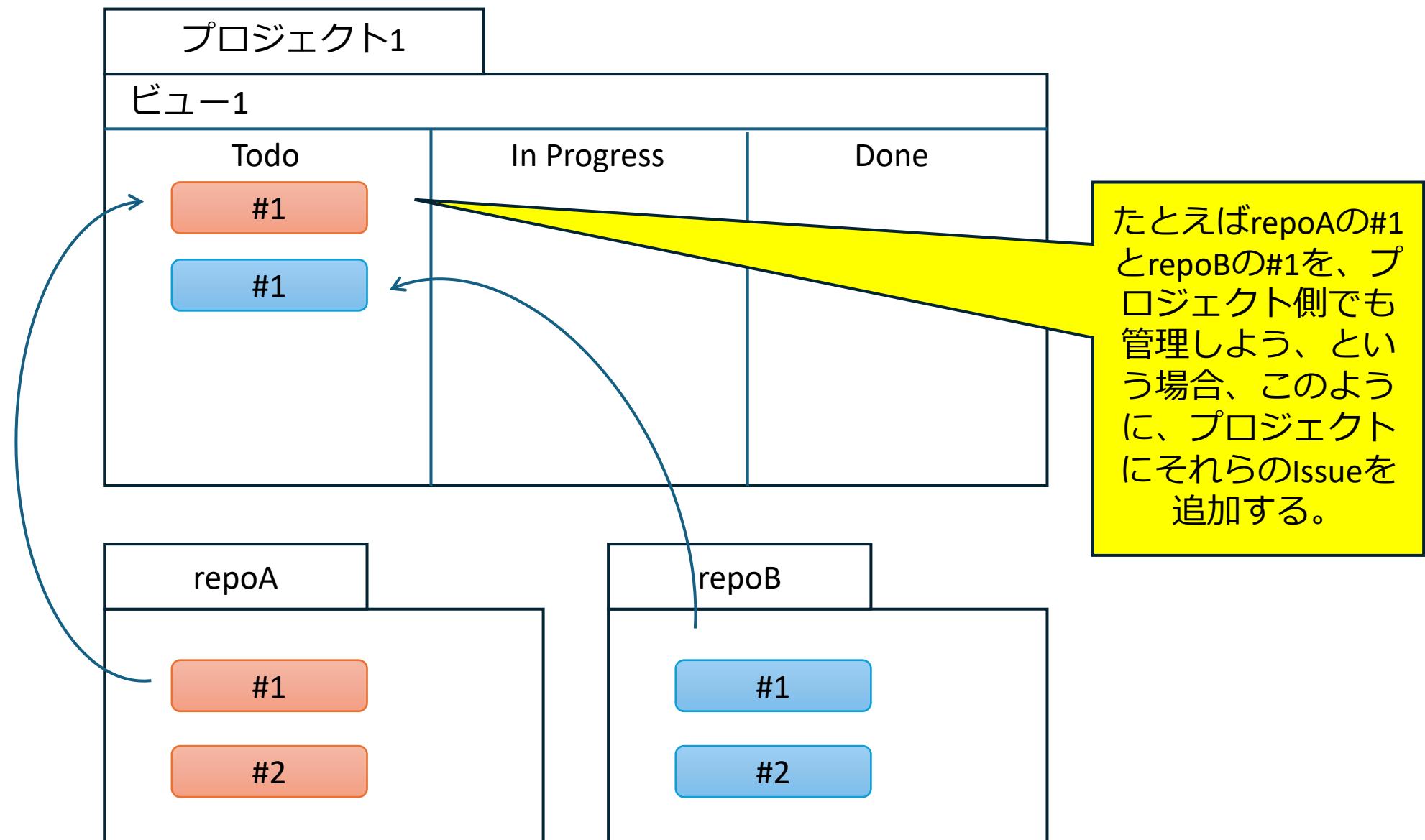


# GitHubを使用したプロジェクト管理

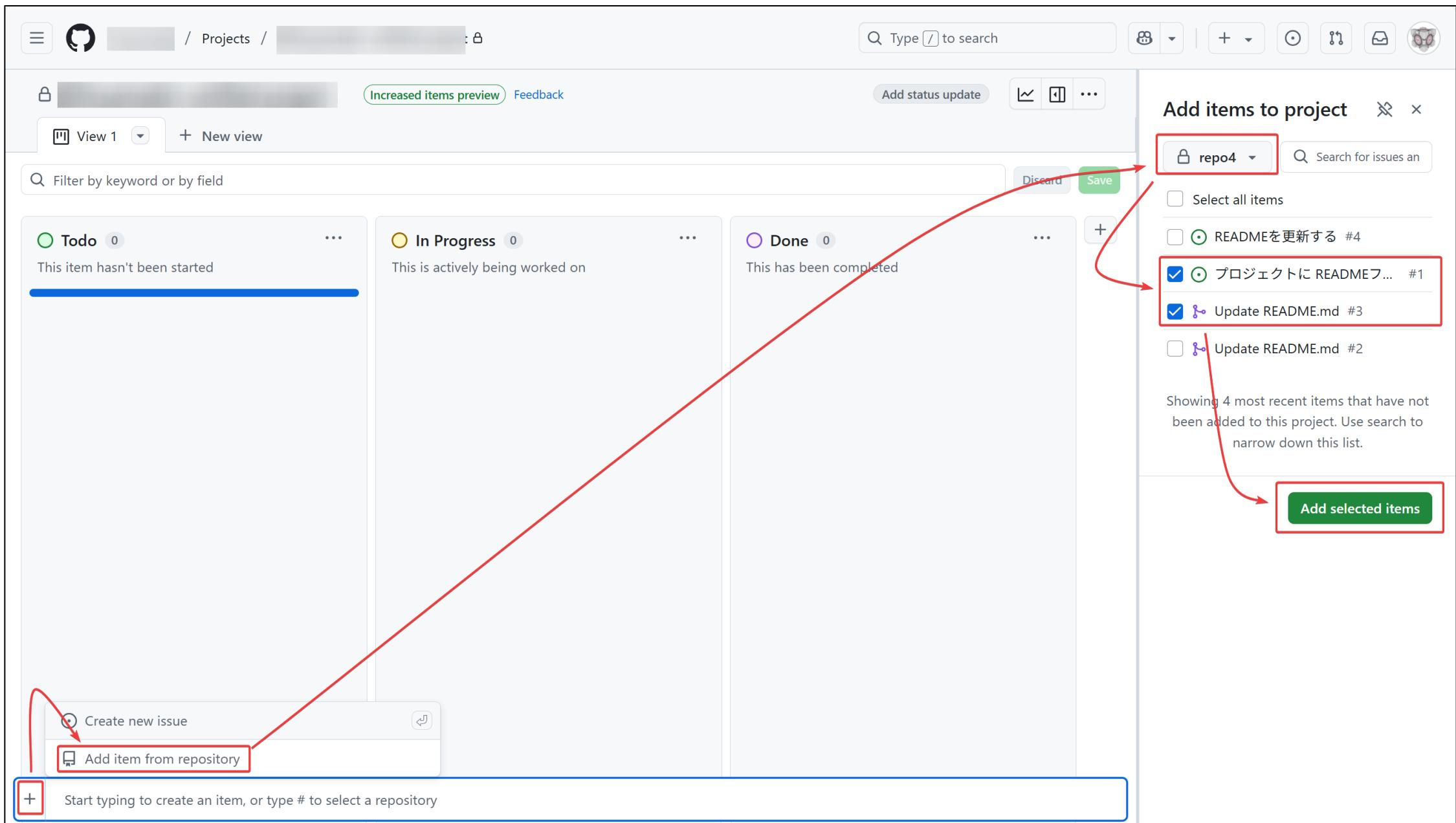
- プロジェクトを作る
- Item (IssueとDraft) を作る (Board ビューの利用)
- Tableビューの利用

■各リポジトリで管理されている Issue をプロジェクトに追加できる。

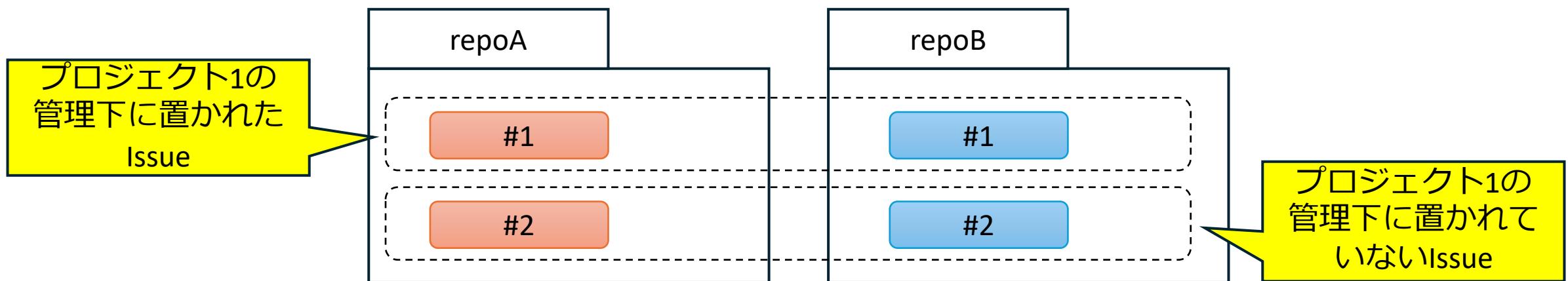
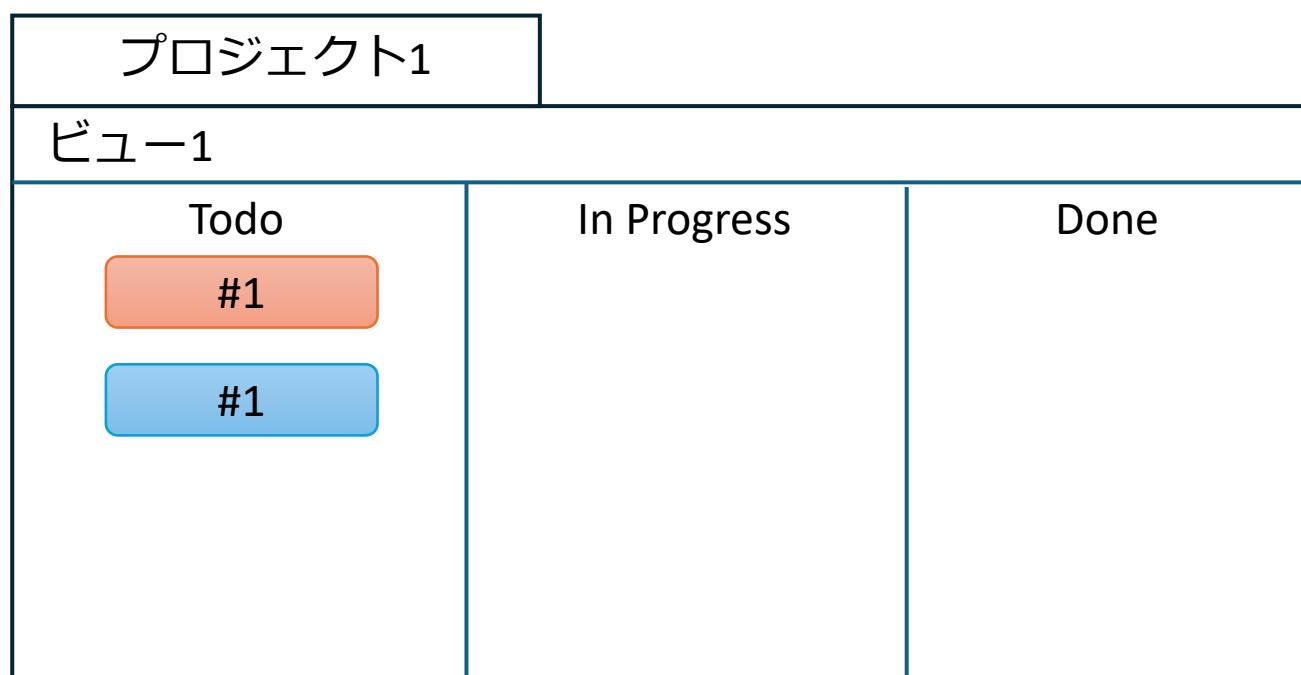
※このとき Issue のコピーが作られるわけではない。リポジトリに作成された Issue が、プロジェクトにも表示されている (Issueがプロジェクトでも管理されている) という状態となる。



■プロジェクトの下部で「+」をクリックし、「Add item from repository」をクリック。  
画面右側でリポジトリを選び、リポジトリ内のIssueなどを選択し、「Add selected items」をクリック。



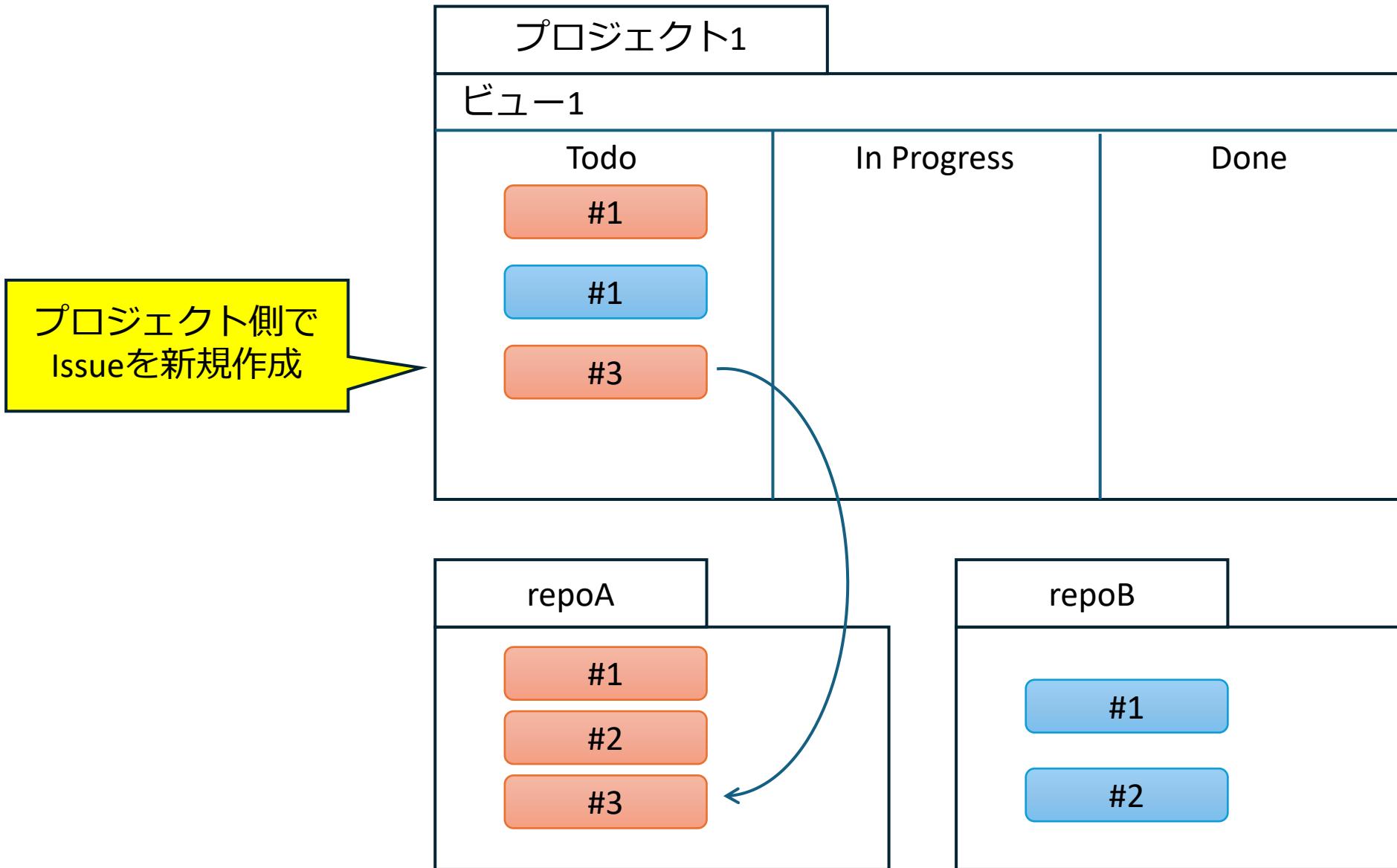
■ただし、リポジトリのすべての Issue がプロジェクトで管理されるわけではないことに注意。



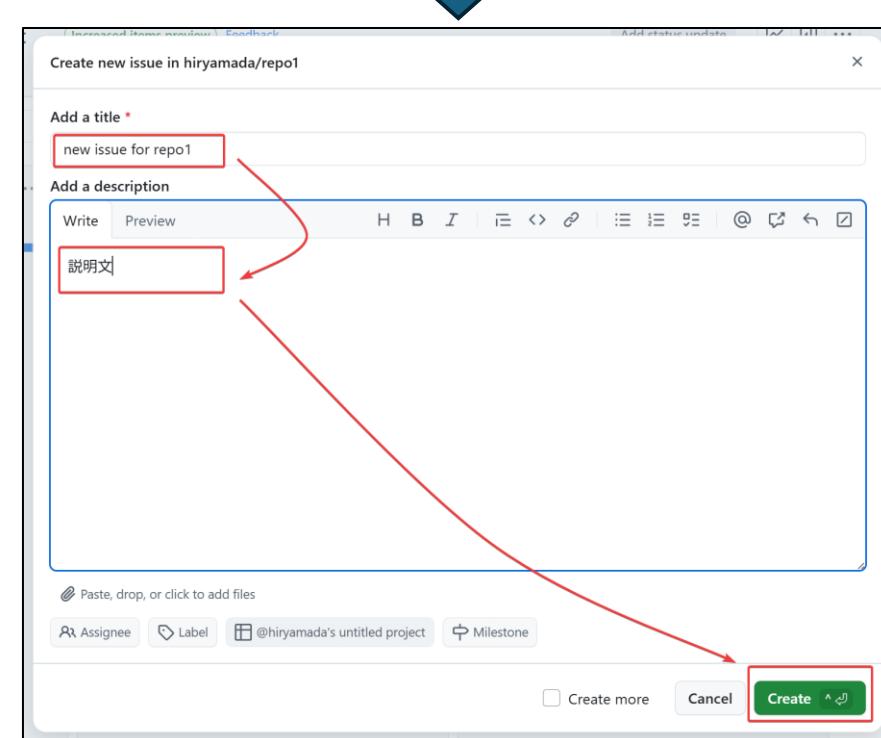
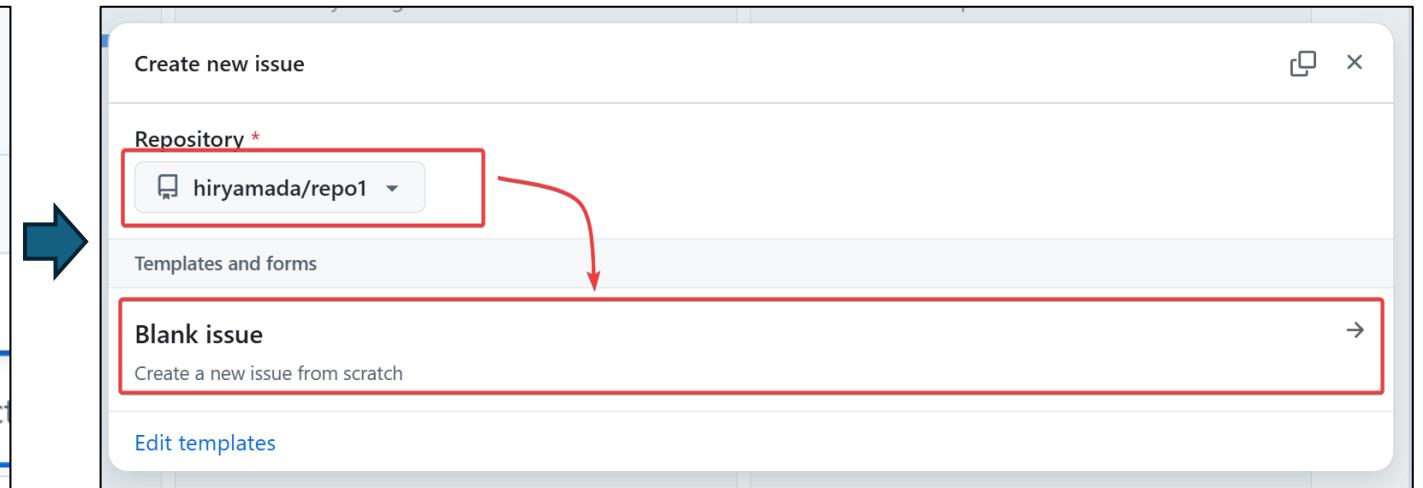
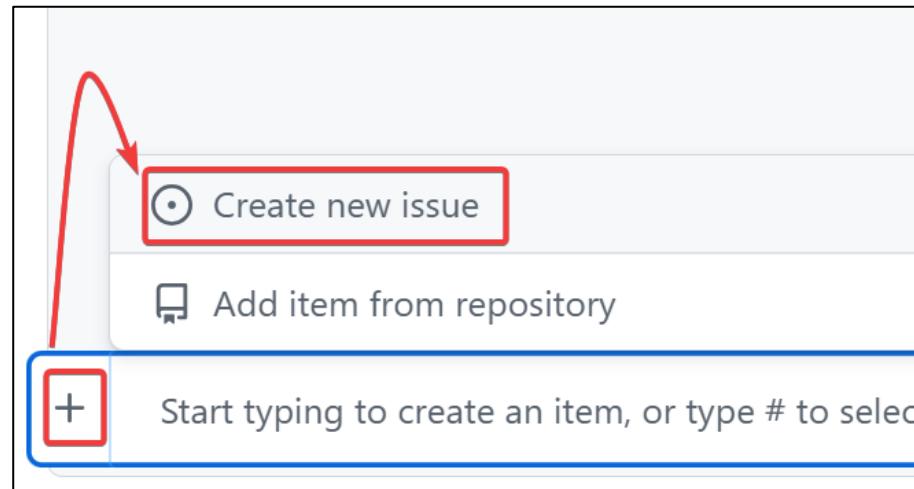
# 参考: Issueを自動的にProjectにも追加する

- Issueを一つ一つ手作業でプロジェクトに追加していくと、手間がかかり、漏れ・抜けが発生する可能性もある。
- リポジトリに追加されたIssueを自動的にプロジェクトにも追加する方法はいくつがある。
  - [アイテムを自動的に追加する - GitHub Docs](#)
  - [GitHub の Issue を自動で Project に追加する方法3選 - NIFTY engineering](#)
  - [開発組織の issue や Pull Request を自動で適切な GitHub Project に割り振っていく](#)

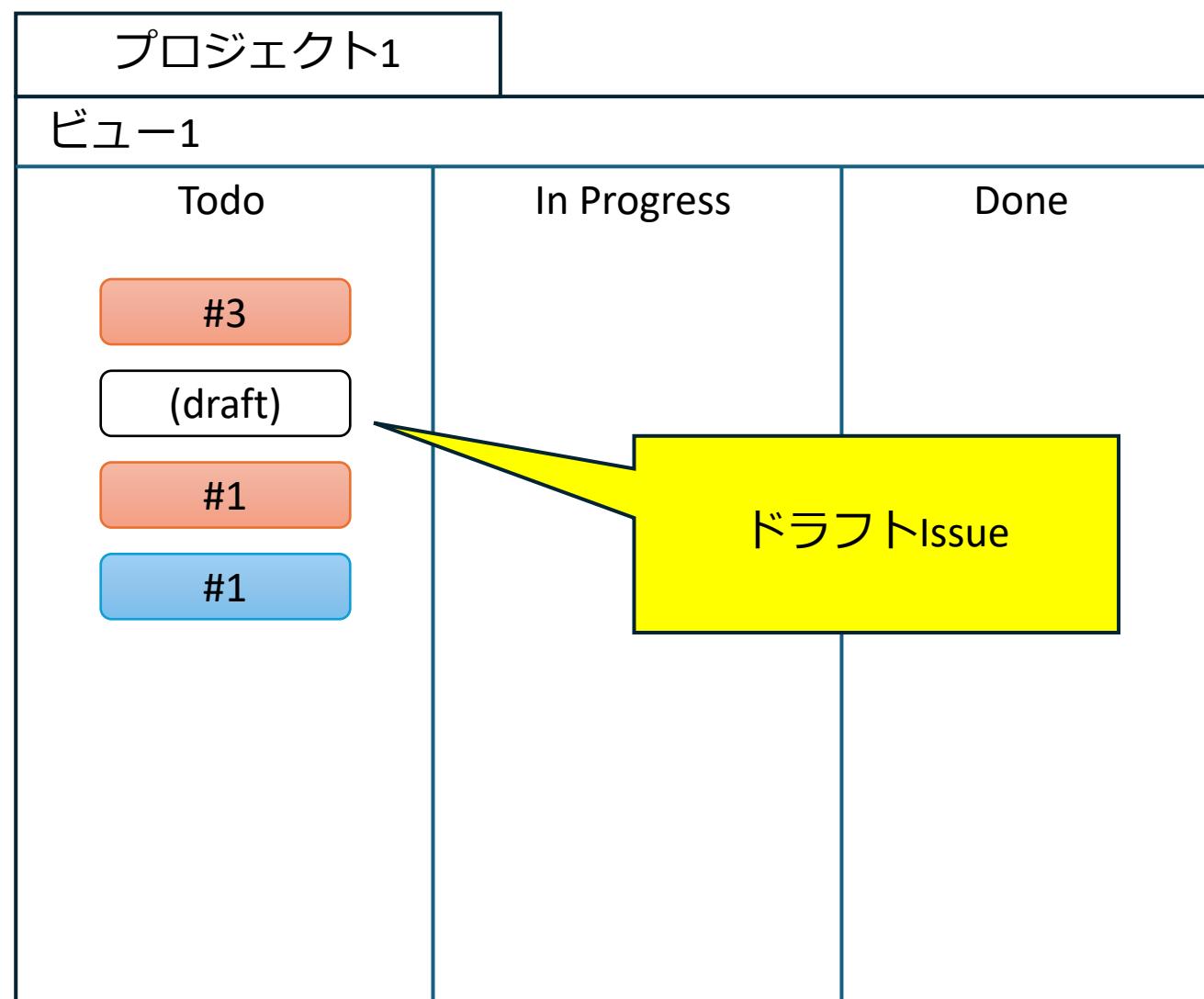
■プロジェクト側で、新しいIssueを作成することもできる。Issueのタイトルを入力し、リポジトリを選択する。



■プロジェクトの画面下部「+」をクリック、「Create new issue」をクリック、リポジトリを選択、「Blank issue」をクリック、Issueのタイトル・説明文を入力して「Create」をクリック。



- プロジェクト内で作られた、まだ特定のリポジトリに関連付けされていないIssueを「ドラフトIssue」という。ドラフトIssueはあとで通常のIssue（プロジェクトに関連付けられたIssue）へと変換できる。プロジェクト側ですばやくアイデアを記録し、あとで特定のリポジトリに関連付ける、といった使い方をする。



■ DraftやIssueは、作成されるとビューの一番下に追加される。  
必要に応じて、ビューの中で**上下の並び順**を手動で並び替えることができる。  
(ただしこの順番には、特別な意味はない。単に、ユーザーが好きな順に並べられる、というだけ)

プロジェクト1		
ビュー1		
Todo	In Progress	Done
<a href="#">#3</a>		
<a href="#">(draft)</a>		
<a href="#">#1</a>		
<a href="#">#1</a>		

WindowsやMacのデスクトップ上でアイコンを好きな位置に配置できるのと似ている

■手動での並び替えは、ビューの「Sort by」オプションが「No sorting (manual)」になっている場合に可能。

The screenshot shows a Jira board titled "project2". The "Board" tab is selected in the top navigation bar. A dropdown menu is open from the "Board" icon, and the "Sort by" option is highlighted with a red box. A secondary modal window titled "Sort by" is displayed, listing various fields for selection. The "No sorting (manual)" option is also highlighted with a red box at the bottom of the list.

project2

Board View 2 + New view

Layout

Table Board Roadmap

Configuration

Fields: Title, Assignees, and S... >

Column by: Status >

Group by: none >

Sort by: manual >

Field sum: Count >

Slice by: none >

Generate chart

Rename view

Move view

Duplicate view

Delete view

Export view data

You can use Control + Space to add an item

Sort by

Select up to 2 fields

Title

Assignees

Status

Labels

Linked pull requests

Milestone

Repository

Reviewers

Parent issue

Sub-issues progress

No sorting

■それ以外のオプションに設定した場合は、その指定に沿って自動的に並び替えが行われる。たとえば第1基準「担当者」、第2基準「タイトル」など。

The screenshot shows the Jira interface with a 'Board' view selected. A context menu is open, and a 'Sort by' dialog box is displayed. The 'Sort by' dialog box contains the following content:

**Sort by**  
Select up to 2 fields

Title 2

Assignees 1

Status

Labels

Linked pull requests

Milestone

Repository

Reviewers

Parent issue

Sub-issues progress

No sorting

The 'Title' and 'Assignees' options are highlighted with a red border.

■画面右上「...」をクリック、「Settings」をクリックすると、プロジェクトの設定が変更できる。ここで「カスタムフィールド」を定義できる。たとえば「重要度」というフィールドを作る。フィールドにセットできる値としてたとえば「低」「中」「高」を定義する。値の色も設定できる。

The image consists of three side-by-side screenshots from a project settings interface.

**Screenshot 1: Project Settings**  
Shows the main project settings menu with options like 'Add status update', 'Workflows', 'Archived items', 'Settings' (which is highlighted with a red box), and 'Make a copy'. A red arrow points from the 'Settings' button to the top right corner of the screen, where a three-dot menu icon is also highlighted with a red box.

**Screenshot 2: Custom Fields**  
Shows the 'Custom fields' section of the settings. It includes 'Project settings', 'Manage access', and a 'Custom fields' tab (which is highlighted with a red box). Below this are sections for 'Status' and 'Sub-issues progress'. A new field is being created, with 'Field name' set to '重要度' (highlighted with a red box) and 'Field type' set to 'Single select' (highlighted with a red box). Under 'Options', there are three entries: '低' (highlighted with a red box), '中' (highlighted with a red box), and '高' (highlighted with a red box). At the bottom are 'Add option...' and 'Save' buttons, with 'Save' highlighted with a red box.

**Screenshot 3: Options**  
Shows the 'Options' page for the '重要度' field. It lists the three defined options: '低' (Low), '中' (Medium), and '高' (High), each represented by a colored circle. Below them is a button labeled 'Add option...'.

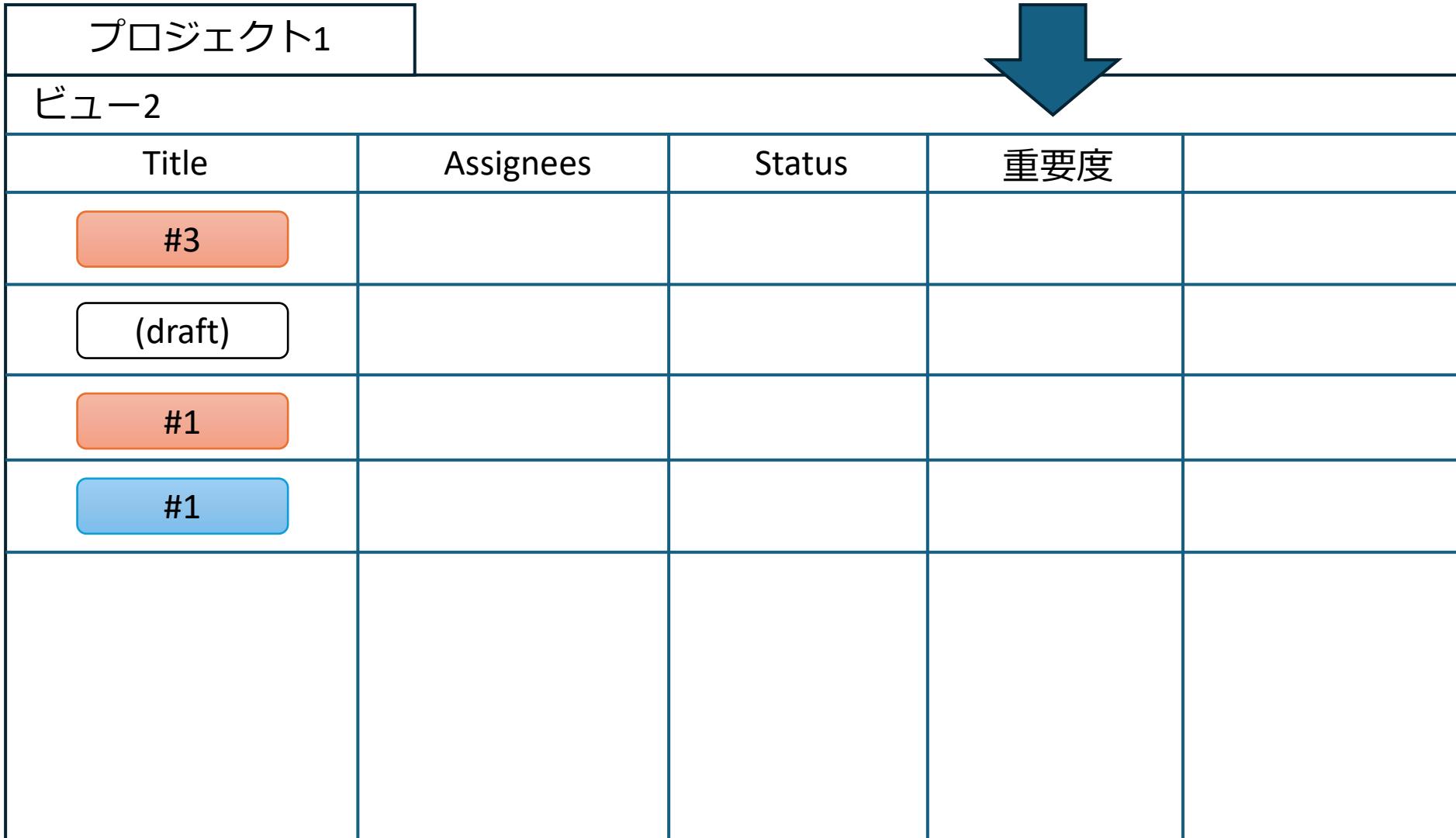
# GitHubを使用したプロジェクト管理

- ・プロジェクトを作る
- ・Item (IssueとDraft) を作る (Board ビューの利用)
- ・Tableビューの利用

- （「ボード」ビューでは、カスタムフィールドの値を設定できないため）  
新しいビュー「ビュー2」を作成し、レイアウトは「**テーブル**」とする。  
「**テーブル**」レイアウトのビューでは、Issueなどが表形式で表示される。

プロジェクト1			
ビュー2			
Title	Assignees	Status	+
#3			
(draft)			
#1			
#1			

■ テーブルの列名部分の「+」をクリックして、テーブルに列の表示を追加できる。  
ここでは先程追加した「重要度」フィールドの列を表示するようにする。



The diagram illustrates a table structure. At the top left, there is a header row with two cells: 'プロジェクト1' and 'ビュー2'. A large blue arrow points downwards from the 'ビュー2' cell towards the main table area. The main table has five columns: 'Title', 'Assignees', 'Status', '重要度' (Importance), and an empty column. There are four rows of data. The first row contains '#3' in an orange rounded rectangle. The second row contains '(draft)' in a white rounded rectangle with a black border. The third row contains '#1' in an orange rounded rectangle. The fourth row contains '#1' in a blue rounded rectangle. All other cells in the table are empty.

プロジェクト1	ビュー2			
Title	Assignees	Status	重要度	
#3				
(draft)				
#1				
#1				

■テーブルでは、各Issueにすばやく値を設定できる。

ビュー2				
Title	Assignees	Status	重要度	+
#3			高	
(draft)			高	
#1			中	
#1				

■各ビューの「Sort by」設定で、作成した「重要度」フィールドの値の順に並び替えて表示するように設定できる。たとえば「重要度の降順」（高、中、低、の順）というようにセットする。

The screenshot shows the Jira interface for configuring a view. A red box highlights the 'Sort by' section in the configuration menu.

**Sort by**  
Select up to 2 fields

- 重要度 (Selected)
- Title
- Assignees
- Status
- Labels
- Linked pull requests
- Milestone
- Repository
- Reviewers
- Parent issue
- Sub-issues progress
- No sorting

Discard Save

■すると、**重要度が高い順**にIssueなどを並べて表示することができる。

project2    Increased items preview    Feedback

Board    View 2    + New view

Filter by keyword or field

Title	...	Status	...	重要度	...	+
1 repo2-issue4 #4	...	Todo	...	高	...	+
2 test	...	Todo	...	高	...	+
3 repo2-issue2 #2	...	Todo	...	中	...	+
4 repo2-issue3 #3	...	Todo	...	低	...	+

+ You can use Control + Space to add an item

# まとめ

- GitHub Projects
- 複数のリポジトリに記録されたイシューをまとめて管理するしぐみ。複数のリポジトリにまたがった管理がしやすい。リポジトリのイシューを新規作成する、イシューの状態 (Todo, Doing, Done) を変更する、イシューを検索・一覧表示する、といった操作をすばやく実行できる。イシューが管理しやすいように、複数の「ビュー」（表示画面）を作ることができる。

# 全体のまとめ（復習）

このコースで学んだことを振り返りましょう

# まとめ

- GitHubとは？
- ソフトウェア開発のプラットフォーム。インターネット上での多数の開発者とのコラボレーションが可能。エンタープライズ（企業）向けの機能もある。Gitを使用してバージョン管理を行う。自動化機能（GitHub Actions）、セキュリティ機能（GitHub Advanced Security）、AIによる開発者支援機能（GitHub Copilot）などを利用できる。GitHubアカウントを作り、リポジトリ内で、Gitを使用してソースコードを管理する。

# まとめ

- GitHub Codespacesとは？
- Webブラウザーからアクセスして利用できる、クラウドベースの開発環境。Visual Studio Codeがベースとなっている。コードの開発・実行が可能。無料でも利用できる。

# まとめ

- Markdownとは？
- 構造化されたドキュメントを作成するためのプレーンテキスト形式。シンプルな文法で、人間にも読みやすく、書きやすい。GitHubでは追加の文法（GitHub flavored Markdown, GFM）やMermaid（技術的な図を表現するしくみ）も利用できる。

# まとめ

- GitHub Issuesとは？
- リポジトリにバグや機能追加要望などの「イシュー（問題）」を記録するためのしくみ。イシューを登録し、担当者を割り当てる。イシューのコメントではMarkdownを利用でき、「@ユーザー名」でユーザーへのメンション（通知）ができ、「#番号」でイシュー や プルリクエストへのリンクを参照できる。イシューが解決したらクローズする。

# まとめ

- GitHubの検索機能とは？
- GitHub全体、特定のリポジトリ内、などをすばやく検索できる。キーワードを指定して、それを含むリポジトリ（コード）、issues、プルリクエストなどを検索できる。

# まとめ

- ・プルリクエスト
- ・多数のユーザーでの共同開発を行うための、GitHubの最も重要な機能。リポジトリをフォーク（コピー）し、新しいブランチを作成して、そこでコードの修正作業などを行う。変更部分をプルリクエストとして元のリポジトリに送信。元のリポジトリのオーナーはプルリクエストの内容を確認し、それをマージ（取り込み）できる。

# まとめ

- GitHubのセキュリティ機能
- (1) 「Dependabot」は、依存関係（リポジトリで使用しているライブラリなど）をスキャンし、脆弱性があるライブラリを使用している場合はDependabotから通知が飛ぶ。
- (2) 「.gitignore」ファイルを使用して、Gitバージョン管理から除外する（つまりGitHubリポジトリに入れないと）ファイルを指定する。
- (3) 「シークレットスキャン」では、リポジトリにキーやパスワードが送信された場合、アラートを発報してリポジトリの管理者に知らせる。
- (4) 「SECURITY.md」は、GitHubリポジトリのセキュリティ関連情報を記載するファイルで、脆弱性発見時の報告先や報告方法などの情報を含める。
- (5) 「GitHub Code Security」では、リポジトリのコードのセキュリティ脆弱性とコーディングエラーを自動的に検出し、アラートを発報してしてリポジトリの管理者に知らせる。GitHub Copilot Autofixを使用するとその脆弱性を修正する提案が生成される。

# まとめ

- Gitを使った共同開発
- オープンソースの開発に参加する場合は事前にリポジトリをフォークする。それから、そのリポジトリのクローンを行う（リポジトリのコピーを開発環境へとコピー）。クローン後、変更点をローカルのリポジトリにコミットし、リモートのリポジトリ（GitHub上）へとプッシュ（git push）する。クローン後、リモートのリポジトリ（GitHub）の変更点をローカルのリポジトリに取り込むにはプル（git pull）する。
- 「GitHub flow」はこれらの、フォーク、ブランチ作成、変更・コミット、プルリクエストの作成などのシンプルな手順を定めたもの。

# まとめ

- GitHub Projects
- 複数のリポジトリに記録されたイシューをまとめて管理するしくみ。複数のリポジトリにまたがって、リポジトリのイシューを新規作成する、イシューの状態 (Todo, Doing, Done) を変更する、イシューを検索・一覧表示する、といった操作をすばやく実行できる。イシューが管理しやすいように、複数の「ビュー」（表示画面）を作ることができる。

# GH-900

## GitHub の基礎

お疲れ様でした！



A course thumbnail featuring a small icon of a person with a green gear, followed by the word "COURSE" and the title "GitHub Copilot". Below the title is the subtitle "Course GH-300T00-A: GitHub Copilot".

