2023年5月作成

Github & Gitの使い方

はじめに

Gitはプログラマーや WEBデザイナーだけでなく、事務職の方など多くの人に使用されています。使ったことが無い、使えないとそれだけでできることが激減してしまうので、業種に関わらず、必須は技能だと思われます。

ここでは、gitを使えるようになるための一部の説明を提供します。文章の調整や言葉遣いについては行っていないため、見つけてもご容赦ください。

Visual Studio Codeを使用するユーザーを想定していますが、独自のツールを使用しても構いません。ここでは便宜上、VS Codeと呼びます。VS Code等のGUIのソフトウェア上でgit操作するツールは多々ありますが、より細かい操作を行いたい場合はコンソール上での操作を推奨します。エンジニア職は絶対！

ここで説明するgitの内容は不十分な場合があります。本格的に使用する場合は、自己責任で追加の調査を行ってください。
githubには公式のドキュメントもあるのでそこを参照すると良いです。

また、引用が複数あることや、この文章を再配布することは考慮していません。

2023年 5月
Matuyuhi

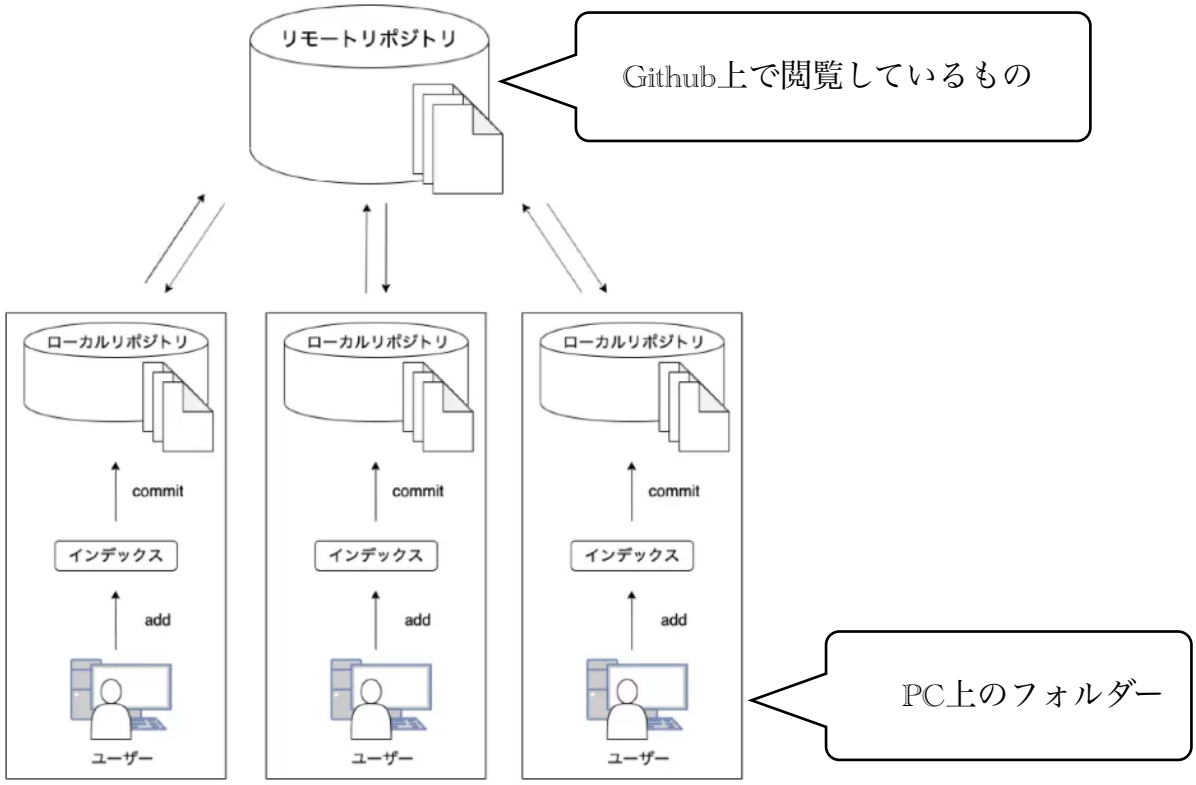
目次

I. Gitについて

・ Gitについて

調べると色々詳しいことが出てくるのでここでは書かないが、ファイルのバージョン管理が簡単にできるツールと覚えておけばなんとかなる。ここでのバージョンとは編集したファイルなどの履歴のことで、gitはその管理をとても簡単に行いことができる。

＊実際にプロジェクトとして使うときの構図



リポジトリ

リポジトリとは、ファイルやディレクトリを保存しておくためのスペースです。Gitにおけるリポジトリは主に２種類に分かれています。

・ リモートリポジトリ: 特定のサーバー上で設置に保存され、複数人で共有する　　ためのリポジトリです。github上で閲覧できるものはこれです。*後述

・ ローカルリポジトリ: ユーザーごとに配置される、手元で編集ができるリポジトリです。``git clone [url]``とかしてきたもの(自分のpcのフォルダー上で見れるやつ)の変更の登録先です。このローカルリポジトリに登録した変更をリモートに登録します。

インデックス

ローカルリポジトリ内で変更された内容は、全てがそのまま共有されるわけではありません。変更を共有したい場合は、一度インデックスに登録しなくてはいけません。編集したファイルをリポジトリへコミットする前にインデックスへ登録して仮置き（add）しておくようなイメージです。インデックスに仮置きしておくことで、ファイルのコミットし忘れや、余分なファイルを除外したりもできます。このことで、機能や作業ごとにバージョンを分け、効率的に進めることができます。

- GitとGithubの違い

簡単に言えばバージョン管理ツール自体はGitでそれweb上で見れたりするのがGithubです。

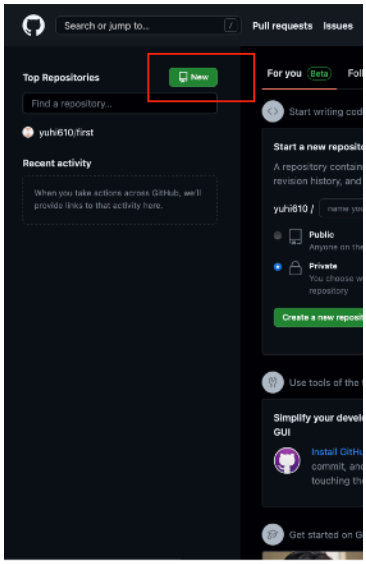
II. Gitの使い方

リポジトリの作成

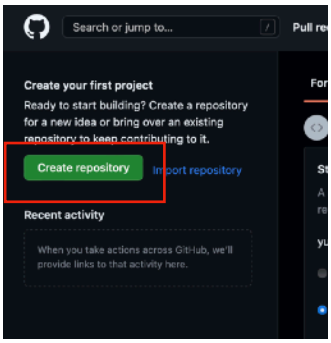
最初に必要になる、リポジトリの作り方。
Github上で作成する方法と、コンソールから作成する方法があるが、ここではgithubを使ってWeb上から作成する保存方法について説明する。

まず[https://github.com]にアクセスします。

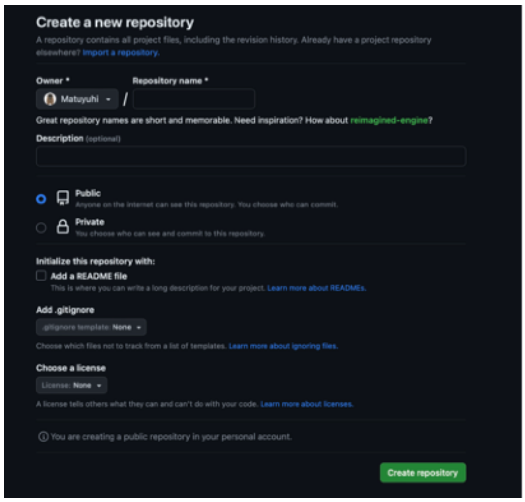
アクセスした画面の左の欄にある[new]ボタン(赤枠)を押すと、作成画面に移動します。



※ 一つもリポジトリがない場合は[create repository]というボタンになります。



ボタンを押すと次の画面に移ります。ここでリポジトリの設定をします。



Owner — このリポジトリの所有者
Repository name — リポジトリの名前 (必須)
Description — 説明文 (任意)
Public or Private — このリポジトリのアクセス権を設定します。publicは誰でもアクセス可能になり、privateはOwner、もしくはOwnerが許可した人のみ閲覧できます。(private推奨)

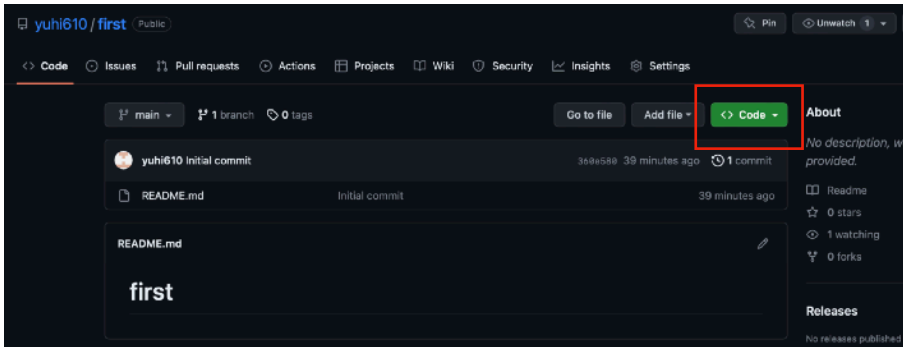
Add a README file — チェックにすると初期状態のリポジトリにREADMEというファイルを追加します。(※推奨)

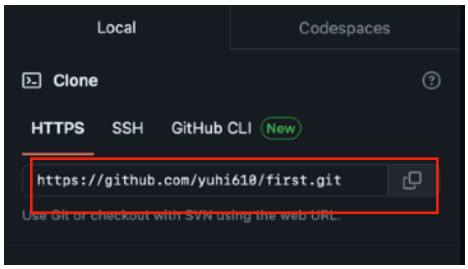
Add .gitignore — 選択したテンプレートのignoreファイルを追加します。
Choose a license — ライセンスの設定(基本無視で良い)

※ここで決めた情報は大抵の場合、後から変更出来るので間違えても問題ないです。

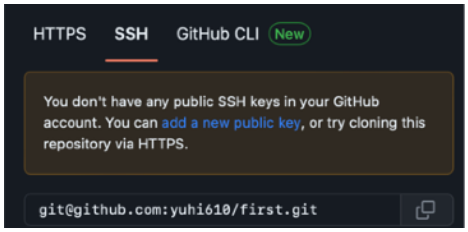
リポジトリのクローン

作業を開始するためにまず、自分のPCにリポジトリをクローンするということをします。この作業で作成されるものが前途したローカルリポジトリです。
リポジトリにアクセスすると次のような画面に移動します。





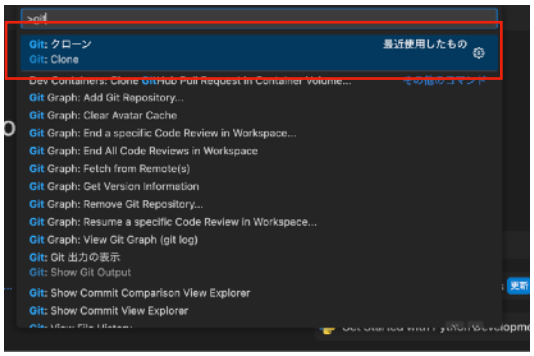
赤色の[Code]のボタンを押すと、左のポップアップが表示されます。
ここのHTTPS、もしくはSSHキーの設定が済んでいる人はSSHのタブからすぐ下に表示されるリンクをコピーしてください。



※ もし、SSHキーの設定ができていない人は左のような注意が出ます。

コピーしたURLを使ってクローンします。

VS Codeの場合：



開いて、[Ctrl + Shift + P]を押すと上部分に入力欄が出てくるので、git cloneと入力すると,[Git: クローン]が出てきます。

下の画像のような画面になるので、先ほどコピーしたURLを入力します。
保存するフォルダーを選んで終了になります。

⇐　CloneのURLを入力

コンソールの場合：

```
❯ cd work
❯ git clone [cloneのURL]
Cloning into
remote: Enumerating objects: 229, done.
remote: Counting objects: 100% (229/229), done.
remote: Compressing objects: 100% (149/149), done.
remote: Total 229 (delta 132), reused 156 (delta 62), pack-reused 0
Receiving objects: 100% (229/229), 67.37 KiB | 353.00 KiB/s, done.
Resolving deltas: 100% (132/132), done.
```

クローンしたいフォルダーでコンソールを開き、git cloneコマンドを入力します。
”cd [作業フォルダー]”
“git clone [url] “

この時、エラーが出ずにフォルダー内にリポジトリの名前と同じフォルダーが作成されていたらOKです。

共有までの流れ

続いて、クローンしたフォルダー内(ローカルリポジトリ)で作業し、変更したものを共有(リモートリポジトリに反映)するまでの流れを説明します。

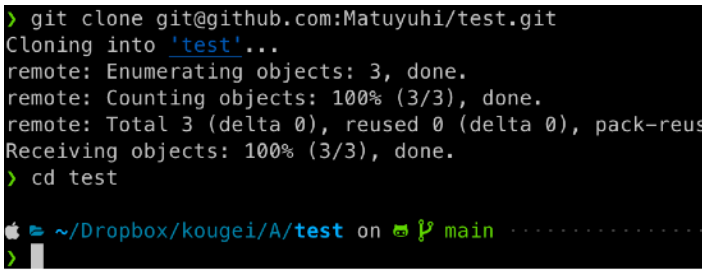
一つの変更を共有する手順を簡単にコマンドで言うとaddしてcommitしてpushするだけです。誰かが共有しているファイルをクローンして、ワークツリーで作業したファイルをインデックスに一度仮置きし、まとめてローカルリポジトリに登録（コミット）する。ローカルリポジトリにコミットしたファイルを共有するためにリモートリポジトリにプッシュするのが基本的な流れになります。

Gitではファイルの編集、追加、削除の全てが1つの変更箇所となります。そして、それら複数を一つの変更箇所のグループ(commit)となり、この１つごとにリポジトリのバージョンを戻したりできます。(commit内のファイル単体を戻すこともできます。)バージョンの確認について詳しくは、[コマンドについて]を参照してください。

Pushまで流れ

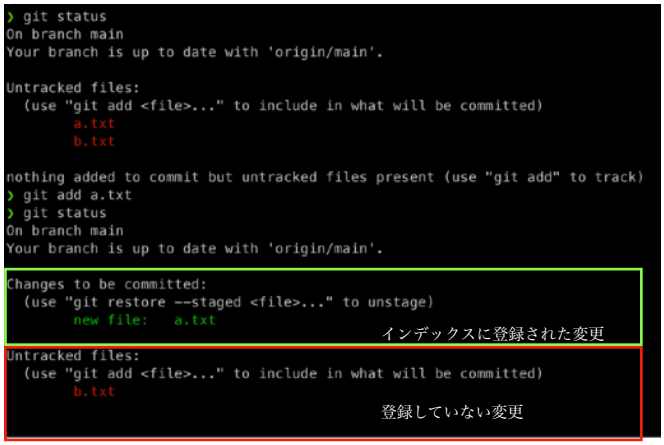
単純にローカルリポジトリ内を変更し、共有するまでの流れを１パターン説明します。

コンソールの場合：

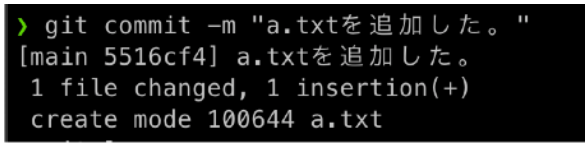


まずsshでcloneして、ダウンロードされたフォルダに移動します。

この時、移動したディレクトリに.gitというフォルダーがあれば大丈夫です。



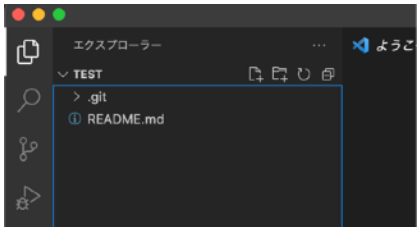
ファイルを追加等に変更を加えたとします。
赤文字で表示されているものが、変更のあるファイルです。
git add a.txtでインデックスに登録後、緑色で表示されているものが確認できます。(全ての変更をaddしたい場合はgit add —allと入力する。)
左ではa.txtがiインデックスに登録されていて、b.txtはインデックスに登録されていない変更(addしていない)となっている。
この時点でcommitをするとa.txtの変更のみが登録されます。同時にb.txtも登録したいb場合はgit add b.txtと入力します。
続いて、このインデックスをcommitします。
commitの方法はいくつかありますが、commit -m “メッセージ”と追加するのが一番楽です。これでエラーが出ていまければ、commitとして登録が完了しました。



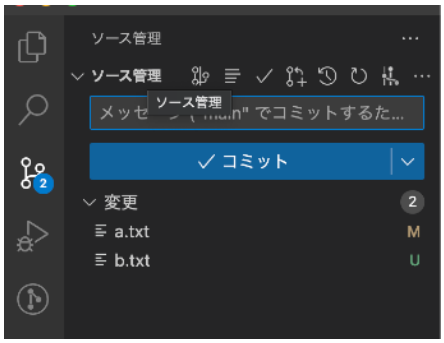
commitとして、登録が終えgit log等でcommitの確認ができるようになりましたが、この時点ではローカルのリポジトリに登録されただけで、リモートには反映されておらず、共有ができません。このcommitをリモートに反映させるにはpushコマンドを使います。左のようなログが出ていればOKです。これで別のpcやgithub上で変更が確認できます。

VSCodeの場合：
基本の流れはコンソールの場合と同じなので詳しい手順はそっちを見てください。

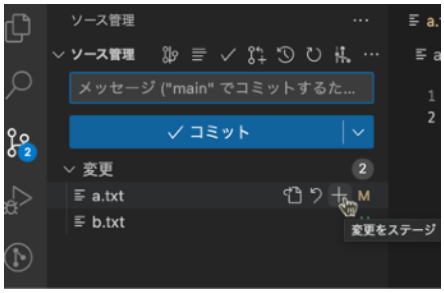
画面を開くと左のメニューに赤枠のような欄があるのでそこで基本は操作します。



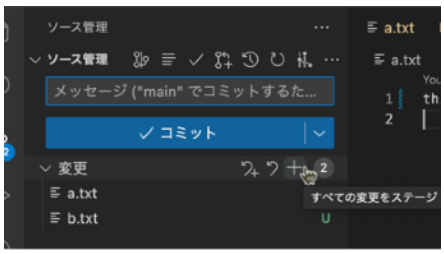
開いたフォルダーに[.git]のフォルダーが表示されていればOKです。ここにはgitの設定等が書かれているので、基本この中を操作する必要はないです。



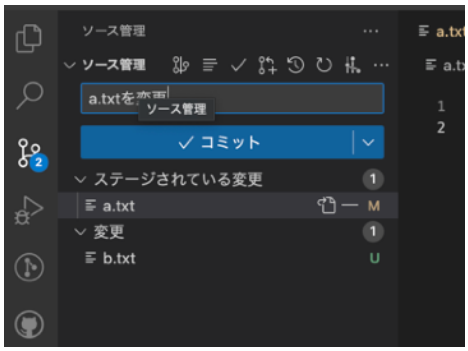
ファイル等に変更があると、このようにファイル名が表示されます。



インデックスに登録したいファイル名にカーソルを合わせると、+ボタンが出るので、クリックするとインデックスに登録されます。これがコンソールではaddと同じ操作です。

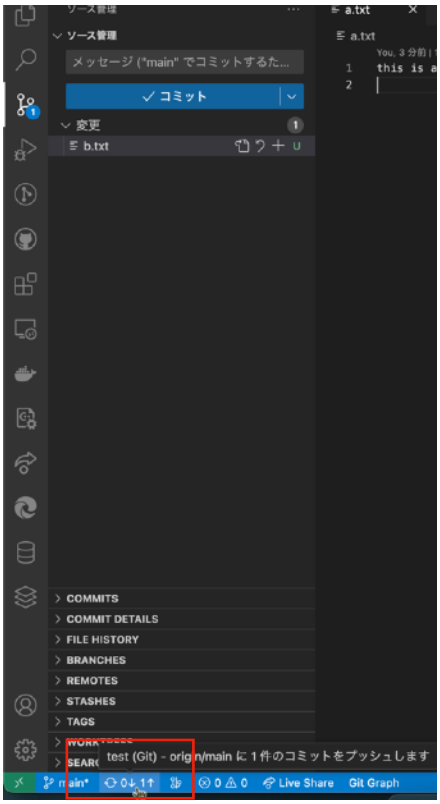


全ての変更を一括で登録したい場合は、変更タブの+ボタンを押せばOKです。



登録したい変更を登録し終えたら、上の入力欄にメッセージを入力して、コミットボタンを押します。これでコミットは完了です。←

Pushするときはウィンドウの下の方に矢印マークがあるので(0件のコミットをプッシュしますと出る)、クリックすれば共有ができます。→

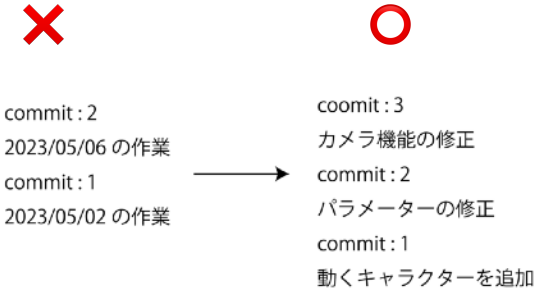


なぜ、commitでメッセージを入力するのか

コミットをする時は必ずと言っていいほど、メッセージを入力します。この時、単に変更を保存するだけでは、後からその変更を誰が行なったか、どのような目的で行ったのかを特定することが出来ません。このため、commit時にはメッセージを入力することが推奨されています。

Commitする前に何回も変更を加えてしまい、どの変更をcommitすればいいか分からなくなる

commitする前に変更を加えてしまった場合は、git statusコマンドで変更内容を確認し、どの変更をcommitするか決めてください。また、変更履歴を管理するために、commitは小さく区切って行うことが推奨されています。



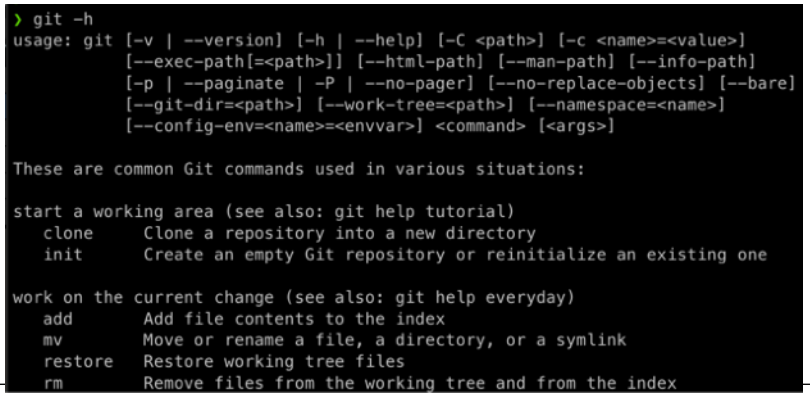
III. 機能について

コマンドについて

コンソールで入力するコマンドは基本 コマンド サブコマンド ...のような形 1 マス空白を開けながら入力します。そのコマンドのサブコマンドやオプションについては、そのコマンドの後ろに -hをつければ大体コマンドのヘルプが表示されます。(hはhelpの頭文字)

コマンドにはオプションが存在するものがあり、optionは[ーー] + [オプション名]で表すものがほとんどです。もしくは [ー] + [オプション名の頭文字]。

例) gitのコマンドが知りたい時 (git -h) or (git —help)



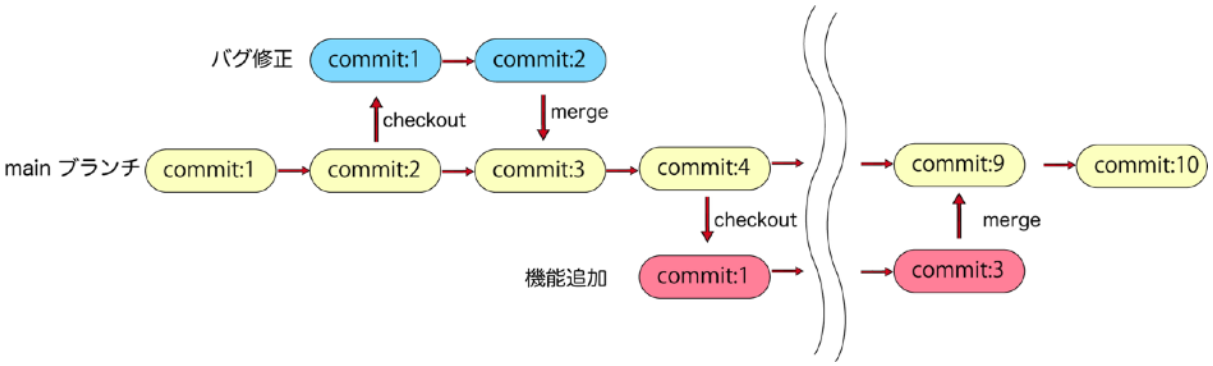
こんな感じで、たくさん出てくる

clone (クローン)

一言で言うとダウンロードに近いです。複数人で共有しているファイル（リモートリポジトリ）をまるごと自分のローカル環境（ローカルリポジトリ）に保存する機能なので、ほとんどの場合Gitで最初に行う作業になります。

push (プッシュ)

アップロードに近いです。ローカルリポジトリに登録したファイルの変更をリモートリポジトリに送信して共有する機能です。これをして初めて、他の人にファイルを共有できます。



branch（ブランチ）

ファイルの変更履歴(commitの履歴)を分岐させて記録していく機能。複数人での並行した作業を正確に管理するための重要な機能です。これがGitのバージョン管理を効率的にし、間違いを減らすためにもっとも活かされている機能ともいえるでしょう。

merge（マージ）

分かれたブランチの変更を自身のブランチにまとめることができます。上図の通り、mainブランチは別ブランチからの変更を取り込みまとめています。

pull（プル）

共有されているリモートリポジトリには存在するが、自身のローカルには無い変更を更新して、その変更分のみをダウンロードする機能です。一度クローンを行えば、その後の更新はpullを行えば、差分のみをダウンロードしてくれるので便利です。

fetch（フェッチ）

これはpullとは違い、リモートの情報を更新するだけで、ローカルのファイルの更新は行いません。ただし、情報は最新の状態になるので、別のブランチの情報や月の場所からの自身のブランチの更新を確認できるようになるので、万が一の衝突を事前に防ぎ、複数人で開発しているときは作業の進捗をすり合わせすることができます。pullは自動的に(fetchとmerge)を行い、更新しているので、衝突することもあるので、使えると便利です。

add（アド）

ファイルの変更をcommitするために、インデックスという仮の置き場に反映させるための機能です。

commit（コミット）

addを使ってインデックスに登録した変更を一つのバージョンとして登録します。

status（ステータス）

現在のインデックスに登録された情報やその他の変更を確認することができます。様々なオプションをつけることで、ファイル単位での確認やファイル内の変更点単位での状況を確認することができるので、addやcommit等のコマンドを使った後に確認としてこまめに使うと良いです。

checkout（チェックアウト）

作業しているブランチを変更することができます。ブランチを変更するコマンドは複数あり、それぞれで使い方があるので自身で調査してください。

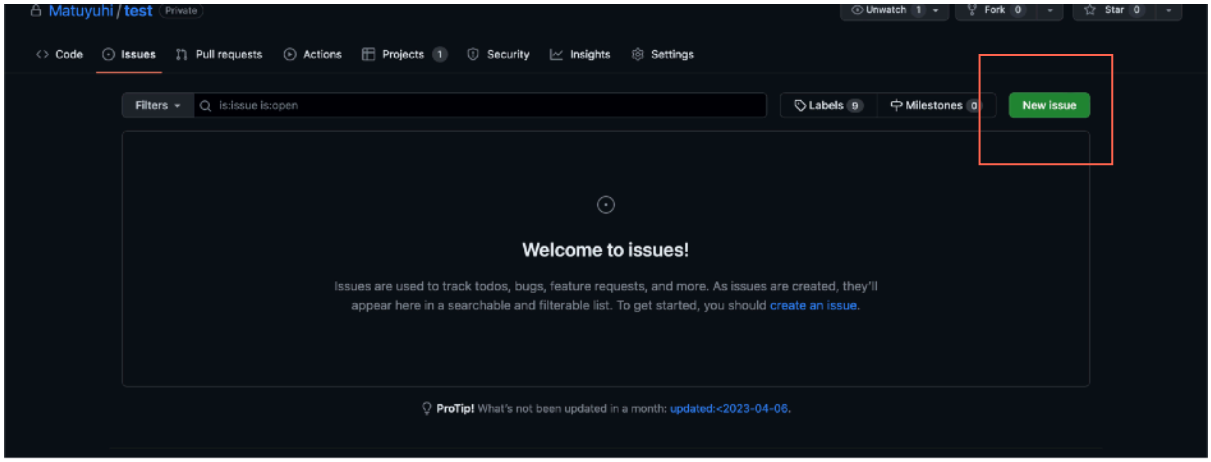
stash　（スタッシュ）

一時的にローカルの変更を無かったことに出来ます。また、stashした変更を元に戻すことも可能です。リモートとの衝突を一旦回避し、mergeやpullなどをして最新状態にしてから作業を再開するような使い方もできます。

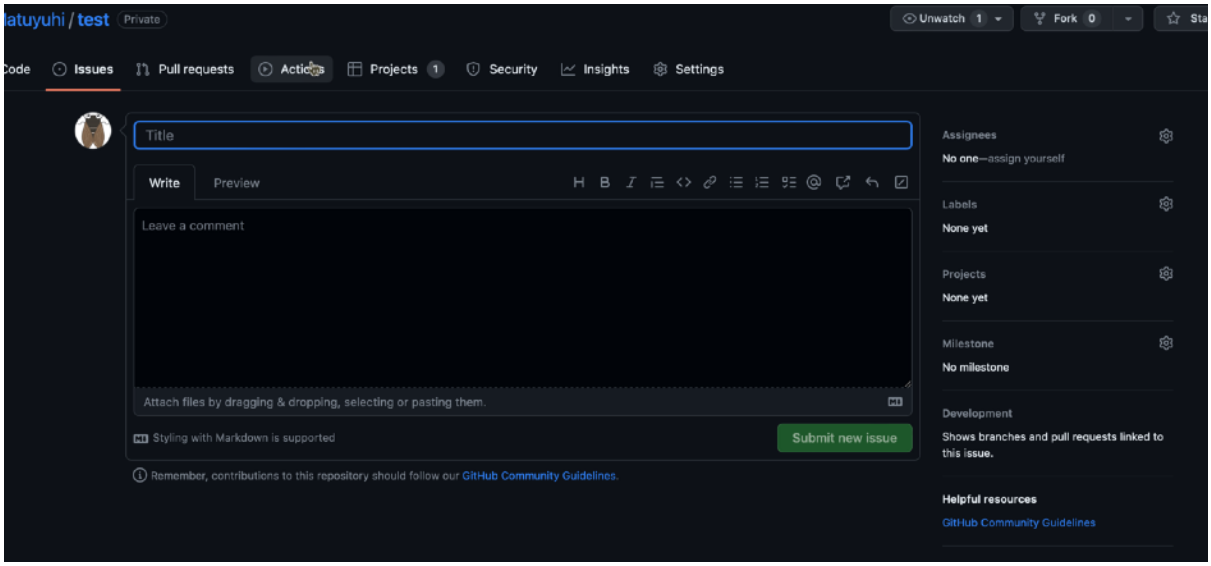
他にもコマンドはたくさんあるので、自分で調査してください。この手の人は一通り使えるのがデフォルトです。

IV. issueについて

プロジェクトの課題管理ツールです。
専用ツールと比べると機能は劣りますが、よく使われています。ソースコードやPull Requestに紐付けて管理ができるので情報を全て一箇所に集めることができ、ソースコードの問題などをより明確にすることが出来ます。
Issueの作成するにはそのリポジトリのissuesタブから作成できます。
日常使いや予定などを立てる場合は、issue専用にリポジトリを作成してもいいと思います。

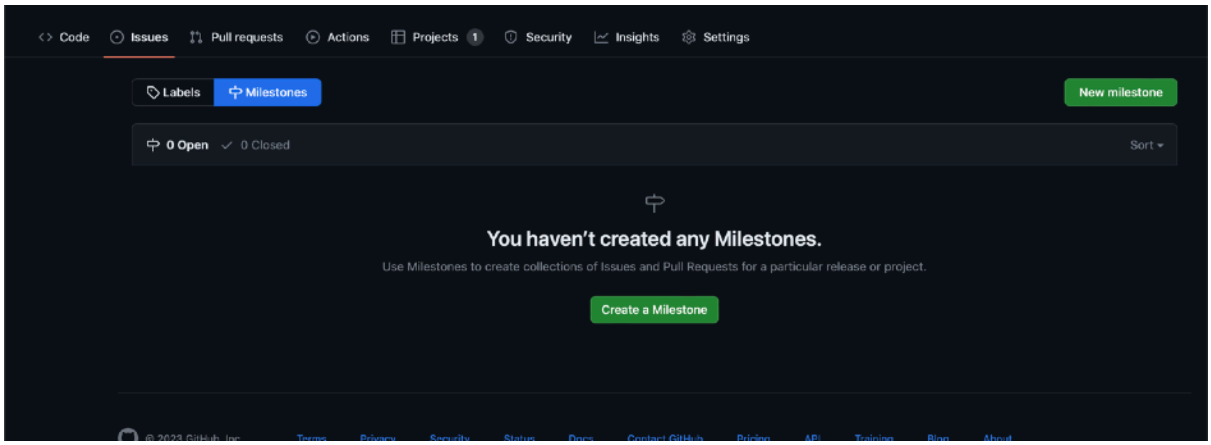


初期状態ではタイトルとその下に備考欄があります。
Assigneesには、バグや修正であればその対応者、予定であれば担当者など、そのissueの担当を設定します。(Assigneesに設定されるとそのissueの更新でユーザーのメールに通知が来ます。)
Labelsは増えたissueを種類別に分けることができる機能です。デフォルトで多くの種類が要されているので、基本それで足りると思います。
Projectsでは後に紹介しているprojectに紐付けて使いことが出来ます。
Milestoneではissueの機嫌を決めることが出来ます。issueごとに締め切りを作る訳ではなく、プロジェクト単位で段階ごとの締め切りを作ります。

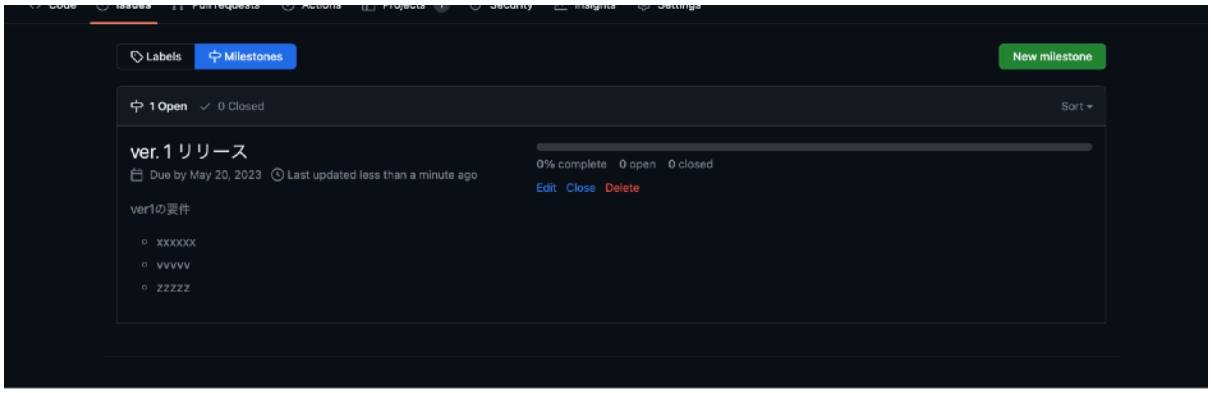


Milestone

プロジェクト管理には必須の機能です。Githubではissueごとに締切を作るのではなく、マイルストンを先に置いてそれにissueを紐付けます。

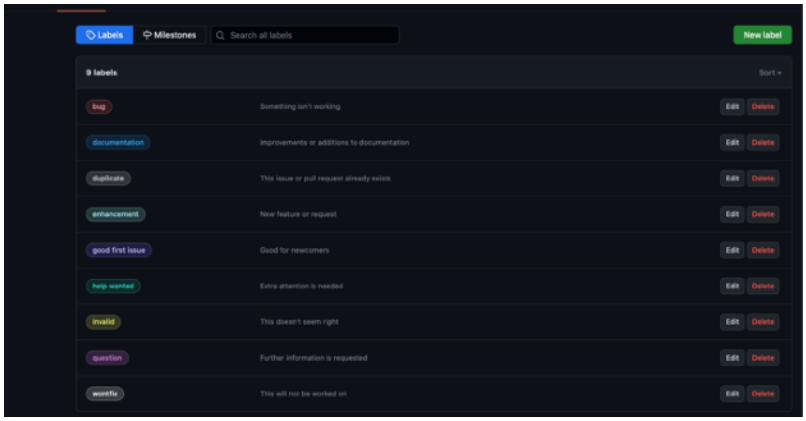


Create a Milestoneから作成します。titleと日付、説明を入力して作成します。
紐付けたissueの進度を見ることが出来ます。



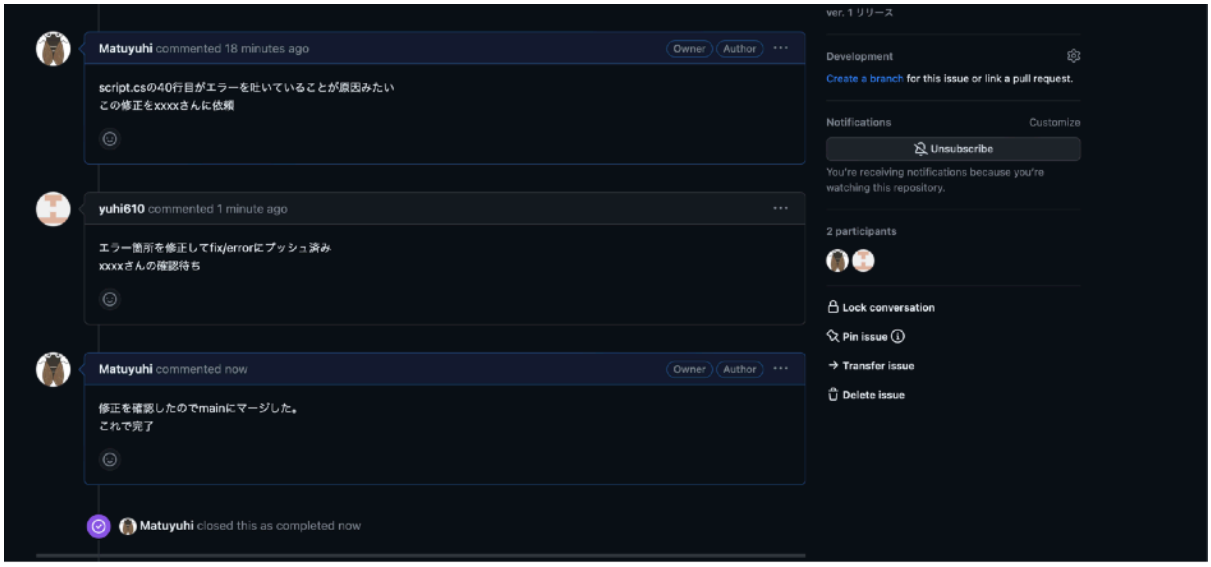
Label

Labelも同じく、種類分けに使います。



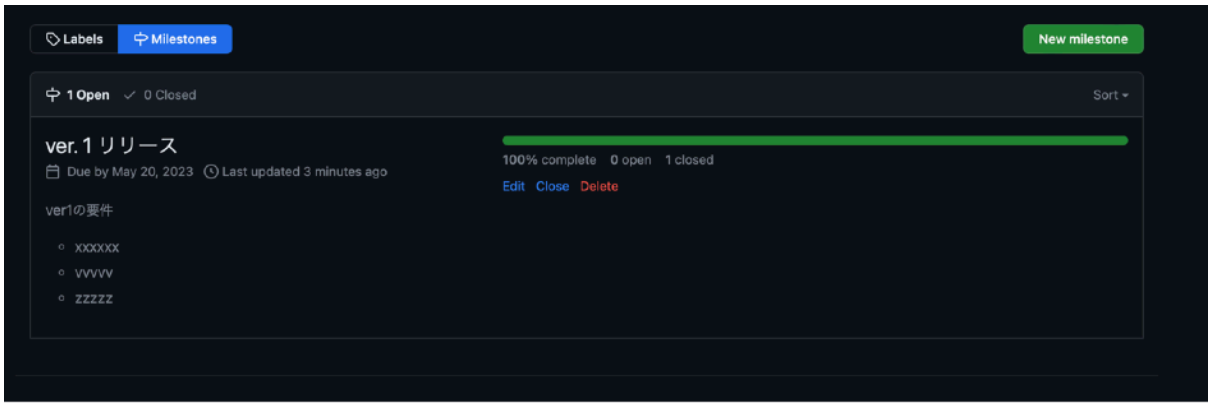
順序

一連の流れを作成するとこんな感じになります。



Issueに対してディスカッションをして、終わったらそのissueをクローズします。
この時、別の問題が発生したら、その問題で新しくissueを作成します。

マイルストーンに設定していると、クローズされたissueが反映されています。

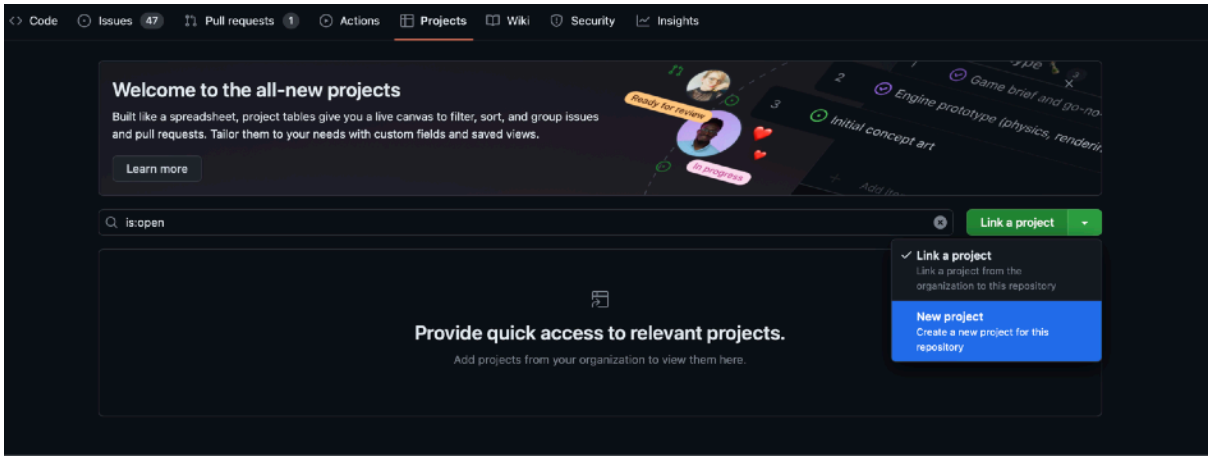


V. projectについて

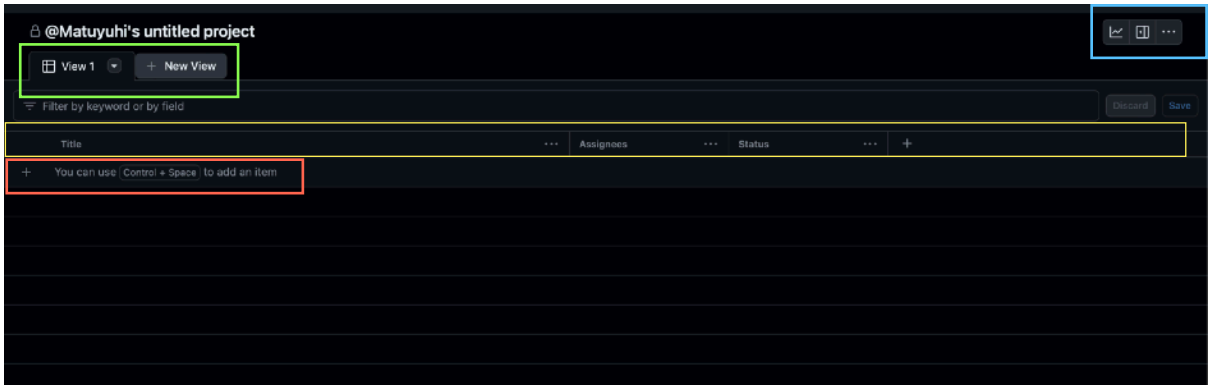
様々なToDo管理のアプリがありますが、Githubのprojectを使えば、ソースコードとタスクの管理を同時に行えるので便利です。タスク管理だけのアプリであると、定期的に見るのを忘れたり、慣れていないとごちゃごちゃになってしまったり、ソードが難しかったりします。その点projectを便利です。通常はチームで開発するときに使う機能ですが，日常のタスク管理にも十分転用できます。また、issueのラベルなどをそのままソートに使えたり出来るのでGithub内の他のツールとの親和性が良いです。一つデメリットを挙げるなら、通知がデフォルトだと無いことです。これはのちに少し説明するwebhhokについてで解決します。

Projectは基本リポジトリか、チーム(ユーザー)に紐付けて作成します。いくつでも作ることが出来ますが、見るコストが上がってしまうので、少ない方が良いです。

任意のページからProjectsタブに選択して、New projectで作成できます。



この時、templateを選択しなかった場合、以下のような空のviewが作成されます。



赤枠 - ここで名前を入力して要素(card)を追加できます。(以降は名前をクリックすると編集できます。)編集画面からissueに変換もできます。

緑枠 - project内に様々タイプのviewを作成できます。ここで看板やロードマップ等をprojectにあったカスタムをします。

黄枠 - fieldです。上図ではtitleとassigneesとstatusが表示されています。

青枠 - 左からinsight、description、optionsです。options内にあるsettingからフィールドの追加等の設定が行えます

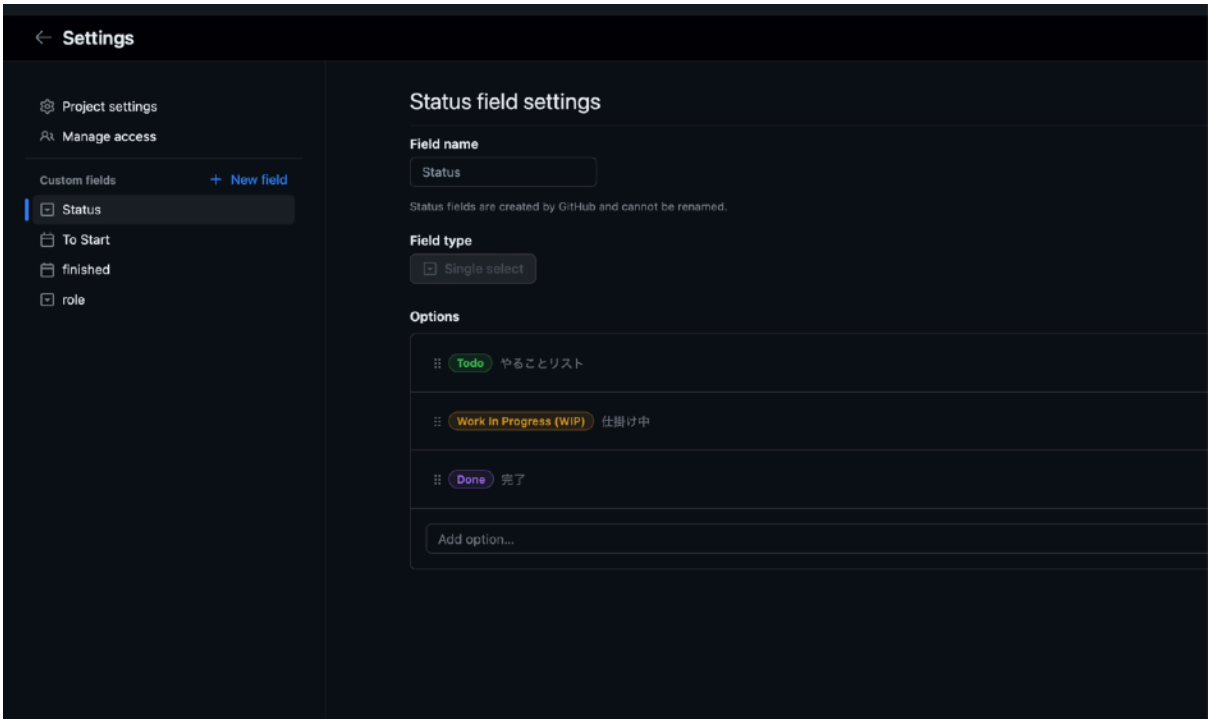
https://github.com/users/Matuyuhi/projects/1/views/5

ここで扱うサンプルを貼っておきます。

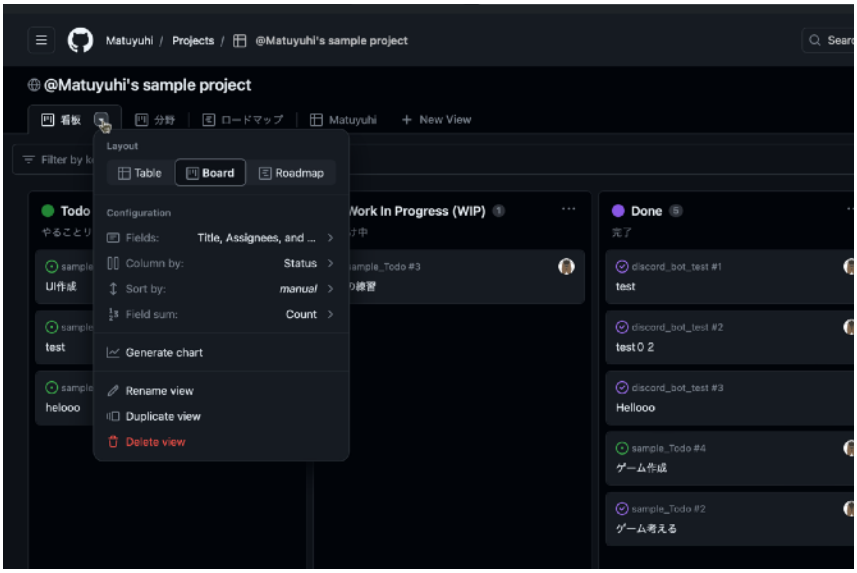
設定

上記した青枠にあるSettingからfieldをカスタムできます。

Field2は様々タイプがあり、指定したオプションから選択させるものや、入力を日付に限定することなどができます。(フィールドに日付に欄を作っておくと後に説明するロードマップのviewを作成することが出来ます。ここではTo Start(開始日)とfinished(終了日)が付与されています。)



Viewの編集にはタブの横にある矢印から編集ができます。



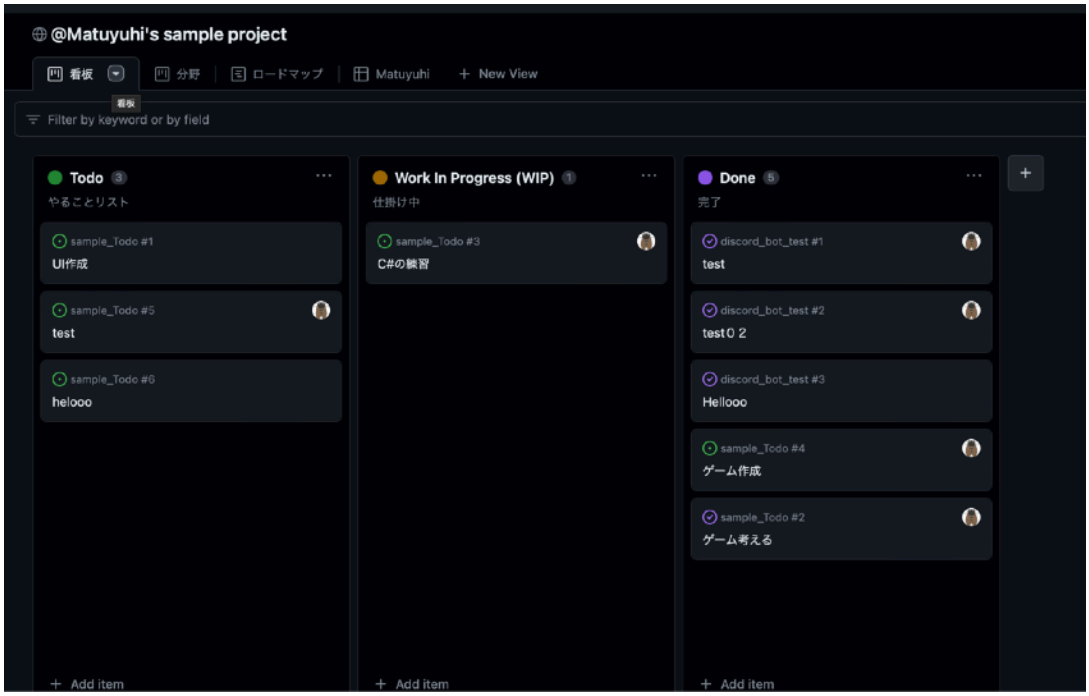
Layoutでviewタイプを３種類から選択できます。

Configurationで表示するフィールドやソートの方法を決めます。(Column by: でstatusフィールドによって列を分けて見やすくなっています。)ここでは
Todo: やるべきこと
WIP: 作業中
Done: 完成済み
ここの書き方はチームによって変わります。
Generate chartで使用履歴のようなものを生成出来ますが、小規模では使わないかも

Layoutの種類

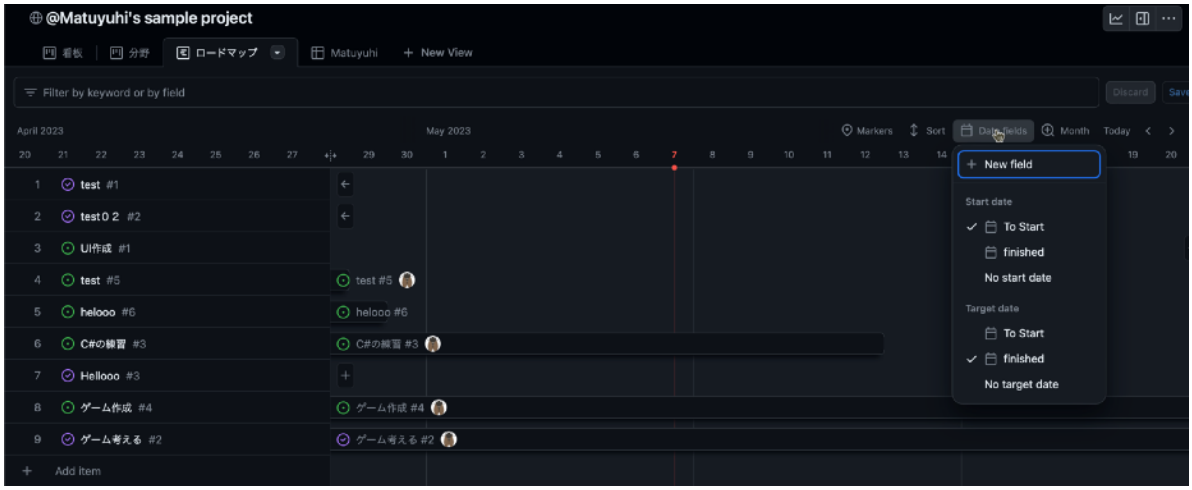
Board：

その名の通りです。confのColumn byからview内のデータを縦に分けることができるのが特徴です。cardをColumn間でドラッグして移動できるので、進捗管理にはこれが使いやすいです。



Roadmap：

フィールドにDateがあれば、日付順にソートしたロードマップを作れます。スケージュールがあるタスク管理に役立ちます。

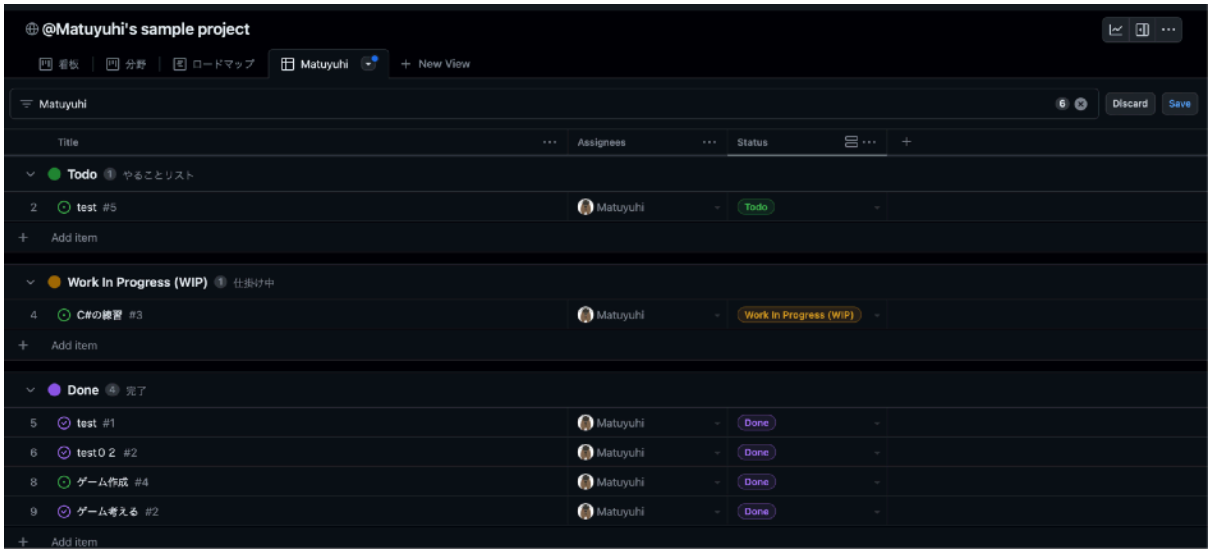


Table：

ベーシックなタイプのviewです。Boardとは違い、一列に並べながらグループ分けができます。予定リストに使いやすいです。

https://github.com/users/Matuyuhi/projects/4

予定表のサンプル



何か設定を編集したときはsaveボタンを押すまで更新されないので注意してください。

VI. その他

Webhookについて

https://docs.github.com/ja/get-started/exploring-integrations/about-webhooks
公式のドキュメント参照

Githubでは、特定のアクションが生じたときに外部のサーバーに通知を送る機能があります。これにより、ユーザーによってカスタムした通知サービスの作成が可能になります。

Discordのwebhook

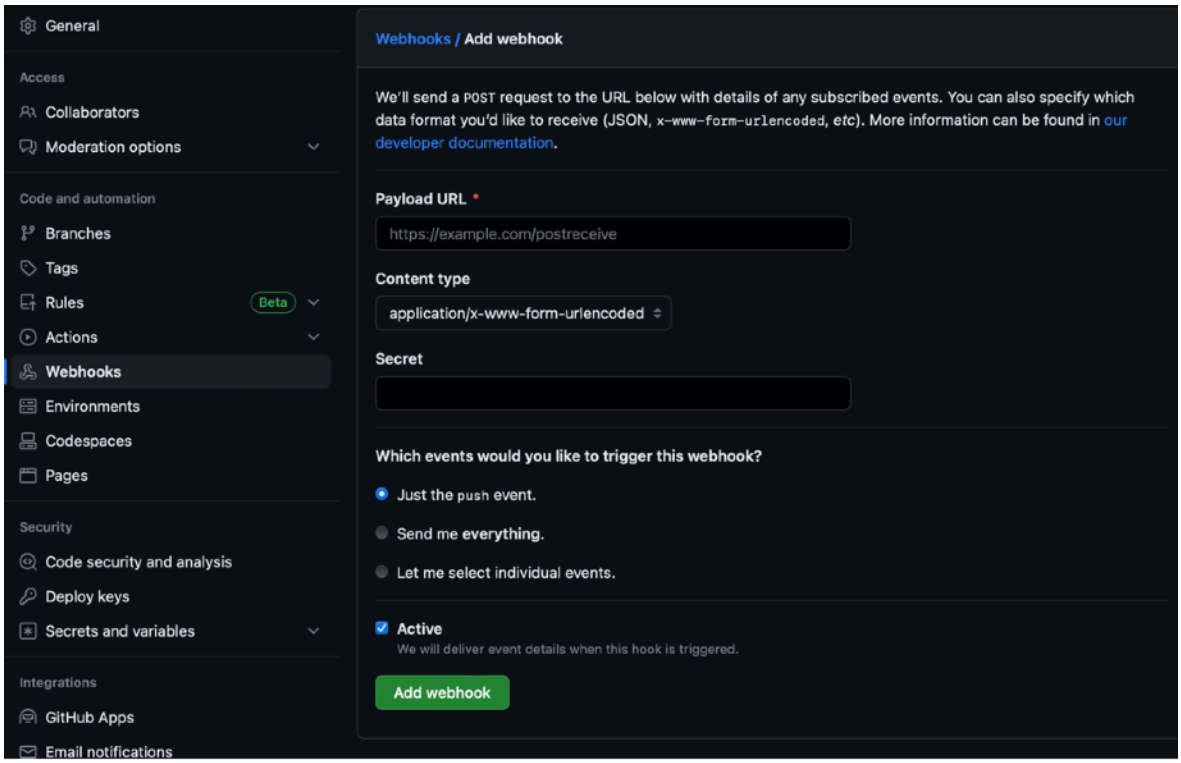
この通知を受け取る機能がdiscordにはあり、リンクさせることでGithub内の特定の通知を指定したDiscordのサーバーに送ることが出来ます。
（より通知をカスタマイブしたい場合は自身でDiscordのbotを作成することをお勧めします。）

通知を送りたいDiscordサーバーの設定画面から>連携サービスにウェブフックの欄があるのでそこから追加できます。

ウェブフックURLをコピーからそのウェブフックのURLを取得できるので、Github内でこれ登録します。

任意のチームか、リポジトリの設定＞Webhooksでウェブフックを登録できます。Payload URLに先ほどコピーしたURLを入力して、Content typeをapplication/jsonにします。Which events ~~の欄で送る通知のタイプを選択します。

上からpushのみ、全て、カスタムです。カスタム(Let me select ~~~)がお勧めです。ここでprojectの更新やissueの更新のみにしたりできます。



自作のBot (※上級)

Payload URLの部分に自身のWebサーバーにURLを入力すると、そのURLのPOSTメソッドが送られるので、通知を別のアプリに送ったり色々できます。

一般的に、自作のDiscordBotを作成する場合Python、もしくはnodejsを使って作成できます。出来ることに差異は無いので、慣れている方webサーバーを立てるといいと思います。

参照

- [【初心者向け】Gitとは何なのか。基本用語やその仕組みをまとめています。], (https://tcd-theme.com/2019/12/what-is-git.html), 01/05/23 閲覧
- [GitHub Projects で日常のタスク管理を行う],(https://zenn.dev/t4t5u0/articles/f3aeb3895fd1fb), 06/05/23 閲覧
- [チーム開発を変える「GitHub」とは？〜Issuesの使い方〜【連載第3回],(https://seleck.cc/647), 07/05/23 閲覧
- 協力 [openai.com]
- その他多くのサイト