2025-05-01
Sora339

myjlab/Git・GitHub Techプレゼン

0.Git・GitHubとは

GitとGitHubは異なるものです

Gitは、プログラムのソースコードなどの変更履歴を記録・追跡するための**分散型バージョン管理システム**です。Linuxカーネルの開発のために、リーナス・トーバルズによって開発されました。Gitを使うと、変更履歴の管理、ブランチを使った並行開発、変更の追跡などが可能になります。Gitはローカル環境で動作するソフトウェアです。

GitHubは、Gitのリポジトリをホスティングするためのウェブサービスです。GitHubを利用することで、インターネット上にGitリポジトリを公開し、複数人での共同開発が容易になります。プルリクエスト、Issue管理、プロジェクト管理、Actions（CI/CD）などの機能を提供し、ソフトウェア開発プロセス全体をサポートします。2018年にMicrosoftに買収されています。

GitHubはGitを利用したサービスであり、Gitがなければ存在しません。Gitはローカルのバージョン管理システムであり、GitHubはそれをインターネット上で共有・活用するためのプラットフォームです。

1.Git・GitHubの基本的な使い方

リポジトリを作成する(Git・GitHub)

- **リポジトリ(Repogitory)** とは、追跡下にあるファイルに対する変更履歴を保管する場所です。

リモートリポジトリの作成(GitHub)

1. GitHubにログインします（アカウントがない場合は作成してください）
2. 画面右上のプロフィールアイコンの横にある「+」ボタンをクリックし、「New repository」を選択します
3. リポジトリの名前を入力します
4. 必要に応じて説明文を追加します
5. リポジトリを公開（Public）または非公開（Private）に設定します
6. 「README file」、「.gitignore」、「ライセンス」などの初期ファイルを必要に応じて選択します
7. 「Create repository」ボタンをクリックします ローカルリポジトリの作成(Git)
8. VSCodeでGitで管理したいフォルダーを開きます
9. ターミナルを開いて以下のコマンドを実行します：

```
echo "# 宮治ゼミナールI 第４回 Techプレゼン Git・GitHub" >> README.md
git init
git add README.md
git commit –m "first commit"
git branch –M main
git remote add origin https://github.com/Sora339/test.git
git push –u origin main
```

ローカル上でブランチを切る(Git)

- **ブランチ(branch)** とは、もとの開発ラインに直接影響を与えずに変更を加えるために、もとの開発ラインから派生させる開発ラインです。

ブランチの作成

```
git branch <ブランチ名>
```

作成したブランチに移動

```
git switch <ブランチ名>
```

変更をステージングする(Git)

```
git add <ファイル名>
```

変更をローカルリポジトリに反映する(Git)

```
git commit -m "<コミットメッセージ>"
```

変更をリモートリポジトリ(GitHub上)に反映する(Git)

```
git push -u origin <ブランチ名>
```

変更をもとのブランチにマージする(GitHub)

- **プルリクエスト(Pull requests)** とは、リモートリポジトリにおいて、変更を他のブランチにマージするための一連のプロセスです。

プルリクエストの作成

1. GitHubのリポジトリページに移動します
2. 「Pull requests」タブをクリックします
3. 緑色の「New pull request」ボタンをクリックします
4. ベースブランチ（マージ先）と比較ブランチ（マージ元）を選択します
5. 「Create pull request」ボタンをクリックします
6. プルリクエストのタイトルと説明を入力します
7. 再度「Create pull request」ボタンをクリックして完了します

マージまでの手順

8. プルリクエストがレビューされるのを待ちます（必要に応じてレビュアーを指定できます）
9. レビュアーからのフィードバックを受け取ったら、必要に応じて変更を加えます
10. CIテストが成功していることを確認します（設定されている場合）
11. すべての条件が満たされたら、プルリクエストページで「Merge pull request」ボタンをクリックします
12. マージ方法（Merge commit、Squash and merge、Rebase and merge）を選択します
13. 「Confirm merge」をクリックします
14. マージが完了したら、必要に応じて「Delete branch」ボタンをクリックして作業ブランチを削除します

リモートリポジトリの変更をローカルリポジトリに反映させる(Git)

- **フェッチ(fetch)** とは、リモートリポジトリの変更を取得して、ローカルにある追跡ブランチに反映する操作です。

```
git fetch
```

- **マージ(merge)** とは、ローカルにある追跡ブランチの変更を、ローカルリポジトリに反映させる操作です。

```
git merge
```

- **プル(pull)** とは、リモートリポジトリの変更を取得して、ローカルリポジトリにマージする操作(fetch+merge)です。

```
git pull
```

mainブランチに戻る

```
git switch -
```

リモートリポジトリの変更をローカルリポジトリに反映させる

```
git pull
```

2.Git・GitHubのその他の使い方

リポジトリをクローンする(Git・GitHub)

- **クローン(clone)** とは、GitHub上のリモートリポジトリをローカル上にコピーする操作です。

1. クローンしたいリポジトリのページをGitHubで開きます
2. 緑色の「Code」ボタンをクリックします
3. 表示されるURLをコピーします（HTTPSまたはSSH）
4. ターミナルを開き、リポジトリをクローンしたいディレクトリに移動します
5. 以下のコマンドを実行します

```
git clone <コピーしたURL>
```

6. 認証が必要な場合は、GitHubのユーザー名とパスワード（またはPersonal Access Token）を入力します
7. クローンが完了したら、作成されたディレクトリに移動します

```
cd <リポジトリ名>
```

.gitignoreファイル

- **.gitignore** とは、Git監視下にあるディレクトリの中で、Gitの監視下に置きたくない特定のファイルを指定するためのファイルです。

1. プロジェクトのルートディレクトリに .gitignore という名前の新しいファイルを作成します。
2. 無視したいファイルやディレクトリのパターンを記述します。例えば：

```
# ビルド成果物
/build/
/dist/

# 依存関係
/node_modules/
```

```
# エディタの設定ファイル
.idea/
.vscode/

# システムファイル
.DS_Store
Thumbs.db

# ログファイル
*.log
```

3. また、すでに追跡されているファイルを無視したい場合は、追加の手順が必要です：

```
git rm --cached <ファイル名>
```

この後、.gitignoreに追加すると、そのファイルは今後追跡されなくなります。

コンフリクト解消(Git・GitHub)

- **コンフリクト(conflict)** とは、同じブランチに異なる変更を加えようとしたときにどちらを採用するべきか判断できず、衝突することです。
- あるコミットから、同じファイルの同じ行に異なる変更を加えてしまうことで起こります。

コンフリクトを解消する方法

1. マージやプル時にコンフリクトが発生すると、VSCodeのエディタでコンフリクトがあるファイルが特別な表示になります。
2. コンフリクトがあるファイルを開くと、VSCodeは3つのペインを表示します： 上部：現在のブランチ（通常は自分の変更） 下部：マージしようとしているブランチ（相手の変更） 中央：結果として採用される最終的な内容
3. 各コンフリクト部分に対して以下の選択肢があります： 「Accept Current Change」：現在のブランチの変更を採用 「Accept Incoming Change」：マージされるブランチの変更を採用 「Accept Both Changes」：両方の変更を採用（上下に配置） 「Compare Changes」：詳細比較 これらのオプションはコンフリクト箇所の上にあるボタンから選択できます。
4. 必要に応じて中央の結果ペインを直接編集して、最適な解決方法を実装することもできます。
5. すべてのコンフリクトを解決したら、ファイルを保存します。VSCodeのソース管理ビュー（Gitアイコン）で該当ファイルを右クリックし、「ステージ」を選択します。
6. すべてのコンフリクトが解決されたら、変更をコミットしてマージを完了します。

ここからは余談です。

3.GitHub×就活

- エンジニアとして就活をしていると、ESなどで自分のGitHubのプロフィールリンク(https://github.com/<ユーザ名>)を求められることが多々あります。
- 自分が実際に書いているコードを見てもらうことができるチャンスorピンチです。
- 自分のユーザ名のリポジトリを作り、README.mdを作ると、自分のプロフィールページに表示することができます。
- 採用担当者に見てもらいやすくしておくといいです。

4.GitHub×オープンソース文化

GitHubとオープンソース文化は密接に関連しており、相互に発展してきました。 GitHub自体がオープンソース開発のためのインフラを提供することで、オープンソース文化の拡大と発展に大きく貢献してきました。

- コードの共有と透明性：GitHubはパブリックリポジトリを無料で提供し、誰でもコードを閲覧・フォークできる環境を作りました。
- コラボレーション：プルリクエスト機能により、プロジェクトへの貢献のハードルが下がり、世界中の開発者が協力して一つのプロジェクトを改善できるようになりました。
- コミュニティ形成：GitHubはIssues、Discussions、Sponsors機能などを通じて、コードを中心としたコミュニティ形成をサポートしています。
- オープンソースライセンス：GitHubはリポジトリ作成時にライセンスの選択を促し、オープンソースライセンスの普及に貢献しています。

現在では多くの主要なOSS(オープンソースソフトウェア（Linux、React、TensorFlowなど))がGitHub上でホスティングされ、世界中の企業もオープンソースへの貢献を積極的に行うようになっています。 GitHubが2018年にMicrosoftに買収された際には、オープンソースコミュニティの一部から懸念の声もありましたが、買収後もGitHubのオープンソースへの貢献は継続・拡大しています。

Good First Issue

Issue機能では、そのプロジェクトに関するタスクを管理・閲覧することができますが、パブリックに公開されているOSSでは、世界中の誰でもそのタスクに取り組むことができるようになっている場合が多くあります。 意欲と自信のある人は、ぜひOSS貢献に取り組んでみましょう。 初心者向けのタスクを多くのOSSから集めた「Good First Issue」というプラットフォームもあります。まずはここを確認するのが良いでしょう。

Good First Issue https://goodfirstissue.dev/

5.より良いブランチ名・コミットメッセージの書き方

代表的なコミットメッセージの書き方の仕様として、「Conventional Commits」があります。

Conventional Commits https://www.conventionalcommits.org/ja/v1.0.0/

6.よく使うGitコマンド集

@uhooi Gitでよく使うコマンド一覧 - Qiita

https://qiita.com/uhooi/items/c26c7c1beb5b36e7418e

7.リファレンス

EaGitro git-process-training

https://github.com/EaGitro/git-process-training

GitHub Docs

https://docs.github.com/ja

Reference - Git

https://git-scm.com/docs

Git - Wikipedia

https://github.com/EaGitro/git-process-training