エンジニアへの第一歩！

学習記録をGitHubで管理しよう

DESIGN
ENGINEERING

今回のゴール

- ✓ 環境構築を行う
- ✓ Gitの操作（status/add/commit/push/pullなど）に慣れる
- ✓ ブランチを使い、JavaScriptのコードをチャプターごとに管理する
- ✓ 実務に近いフローを練習する

目次

- 事前の環境構築
- 新しいリポジトリを作成
- プロジェクトをローカルに作成
- コミットとプッシュ
- 章ごとにブランチを切る
- プルリクエストとマージ

事前の環境構築① VS Codeの準備

✓ Visual Studio Codeのインストール

- ○ VS Code公式からインストール用プログラムをダウンロード
  （https://code.visualstudio.com/download）
- ○ アプリを開いた後、左メニューの「拡張機能」から「Japanese Language Pack for Visual Studio Code」をインストールして日本語化すると便利

【参考資料】

【環境構築】GitHubアカウントを作成してVSCodeと連携できるようにする

Visual Studio Code のインストール方法

※Visual Studio Codeのアイコン：https://code.visualstudio.com/brand

事前の環境構築② Gitの準備

Gitのインストール確認

✓ VS Codeのターミナルで下記のコマンドを入力。

「git version 2.49.0」などが出ればOK

```
git --version
```

✓ 表示されなければ Git をインストール

○ Windows : https://gitforwindows.org/

○ macOS : https://git-scm.com/downloads

【参考資料】

Gitをインストールしてみよう！Windows/Macどちらも丁寧に解説

※Gitのアイコン：https://git-scm.com/downloads/logos

事前の環境構築② Gitの準備

Gitの初期設定

✓ VS Codeのターミナルで名前とメールアドレスを登録（GitHub と同じメール推奨）

```
git config --global user.name "あなたの名前"
```

```
git config --global user.email "あなたのメールアドレス"
```

✓ 初期ブランチ名を main に固定（PCごとに一度やればOK）

```
git config --global init.defaultBranch main
```

事前の環境構築③ GitHubのアクセス

- ✓ https://github.com/ へアクセス
- ✓ 「Sign up for GitHub」に登録するメールアドレスを入力して、アカウント作成
  - ○ パスワード、ユーザーネームを入力。Country/RegionがJapanになっていることを確認して「Create account」
  - ○ メールに届く認証コードで承認、ログインできることを確認する

【参考資料】

【入門】GitHubの使い方｜設定や基本操作など

※GitHubのアイコン：https://github.com/logos

事前の環境構築④ SSH鍵の確認

✓ すでに鍵があるかの確認

○ ターミナルで下記を入力

```
ls ~/.ssh
```

id_ed25519 / id_ed25519.pub があれば鍵あり

```
Air ~ % ls ~/.ssh
id_ed25519
id_ed25519.pub
```

○ 「.ssh」が存在しない場合は、下記コマンドでホームディレクトリに .ssh フォルダを作成する

```
mkdir -p ~/.ssh
```

エラーが出なければOK

事前の環境構築④ SSH鍵の作成と登録（1）

✓ VS Codeのターミナルで下記コマンドを実行

```
# 1) SSHの公開鍵と秘密鍵を作成（メールはGitHubに登録したものを推奨）
ssh-keygen -t ed25519 -C "your-email@example.com"

# 2) 公開鍵をクリップボードにコピー（Mac）
pbcopy < ~/.ssh/id_ed25519.pub
# (Windowsの場合は下記のコマンドでコピー)
clip < ~/.ssh/id_ed25519.pub
```

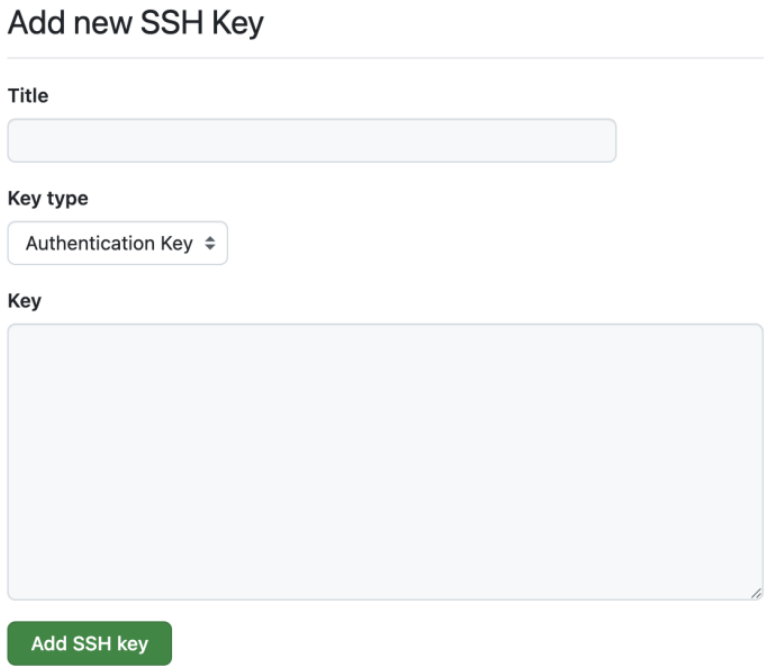
✓ GitHubの下記設定ページへ移動、SSH and GPG keys > New SSH key へ

https://github.com/settings/ssh/new

事前の環境構築④ SSH鍵の作成と登録（2）

✓ クライアントの公開鍵を登録

- ◯ **Title**：例「MyComputer」
- ◯ **Key**：先ほどクリップボードにコピーしたものを貼り付け
  （先頭が `ssh-ed25519` で始まる文字列。メールアドレスは不要）
- ◯ **Key type**：Authentication Key のままでOK



事前の環境構築④ SSH鍵の作成と登録（3）

✓ SSH設定ファイルの作成。ターミナルで下記コマンドを入力（Mac）

```
touch ~/.ssh/config
cat >> ~/.ssh/config << 'EOF'
Host github.com
  HostName github.com
  User git
  IdentityFile ~/.ssh/id_ed25519
  AddKeysToAgent yes
  UseKeychain yes
EOF

eval "$(ssh-agent -s)"
ssh-add --apple-use-keychain ~/.ssh/id_ed25519
```

事前の環境構築④ SSH鍵の作成と登録（3）

✓ SSH設定ファイルの作成。ターミナルで下記コマンドを入力（Windows）

```
touch ~/.ssh/config
cat >> ~/.ssh/config << 'EOF'
Host github.com
  HostName github.com
  User git
  IdentityFile ~/.ssh/id_ed25519
  AddKeysToAgent yes
EOF

eval "$(ssh-agent -s)"
ssh-add ~/.ssh/id_ed25519
```

事前の環境構築④ SSH鍵の作成と登録（4）

✓ 下記ターミナルのコマンドでSSH接続の確認を行う

```
ssh -T git@github.com
```

✓ 初回は以下のようなメッセージが表示される場合があるが、 yes を入力

「This key is not known by any other names.

Are you sure you want to continue connecting (yes/no/[fingerprint])?」

✓ 成功すると下記のメッセージが出る

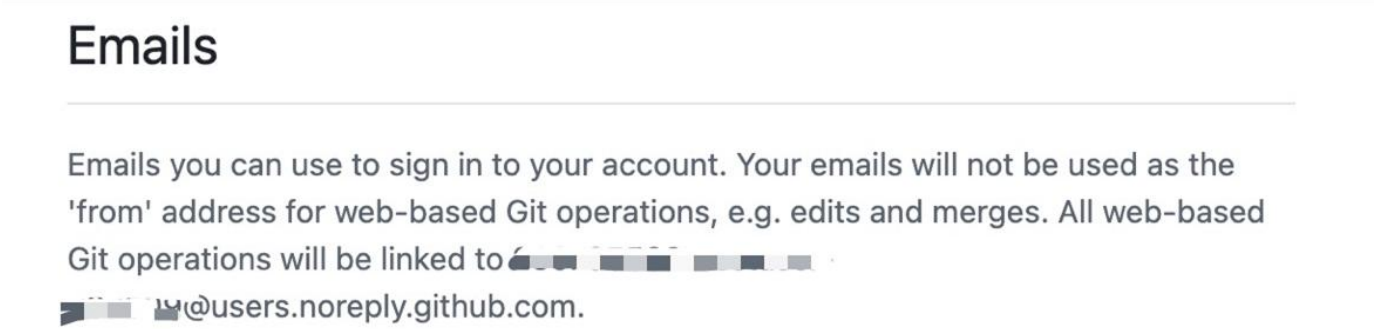
Hi <ユーザー名>! You’ve successfully authenticated, but GitHub does not provide shell access.

事前の環境構築⑤ 自身の公開アドレスを隠す（任意）

✓ GitHubの設定ページへ（https://github.com/settings/emails）

- ◯ **Keep my email addresses private** をONにする
- ◯ **Block command line pushes that expose my email** をONにする

✓ **下記画面の「～@users.noreply.github.com」をコピーする**



事前の環境構築⑤ 自身の公開アドレスを隠す（任意）

✓　ローカルのGitのメールアドレスを、noreplyメールアドレスに設定する

```
git config --global user.email "~@users.noreply.github.com"
```

【参考資料】

Git を macOS にインストールする

GitHub の noreply メールアドレスの設定と、スパム業者に関するtips

GitHubのSSH key作成と設定方法

GitHub コミットメールアドレスを設定する

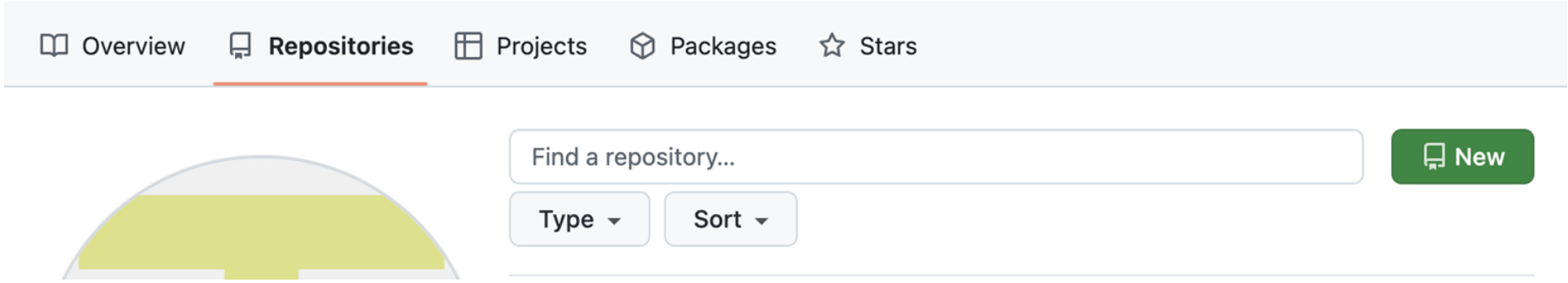
ローカルとリモート

- ✓ ローカル (Local)
  - ○ 自分のPCの中にあるリポジトリ（作業用の場所）
  - ○ VS Codeで編集したり、git add / git commit する場所

- ✓ **リモート (Remote)**
  - ○ **GitHubなどサーバー上のリポジトリ（クラウドの保管庫）**
  - ○ **チームと共有したり、バックアップとして保存する場所**
  - ○ git push でアップロード、git pull （今回は fetch / merge を使用） で取り込む

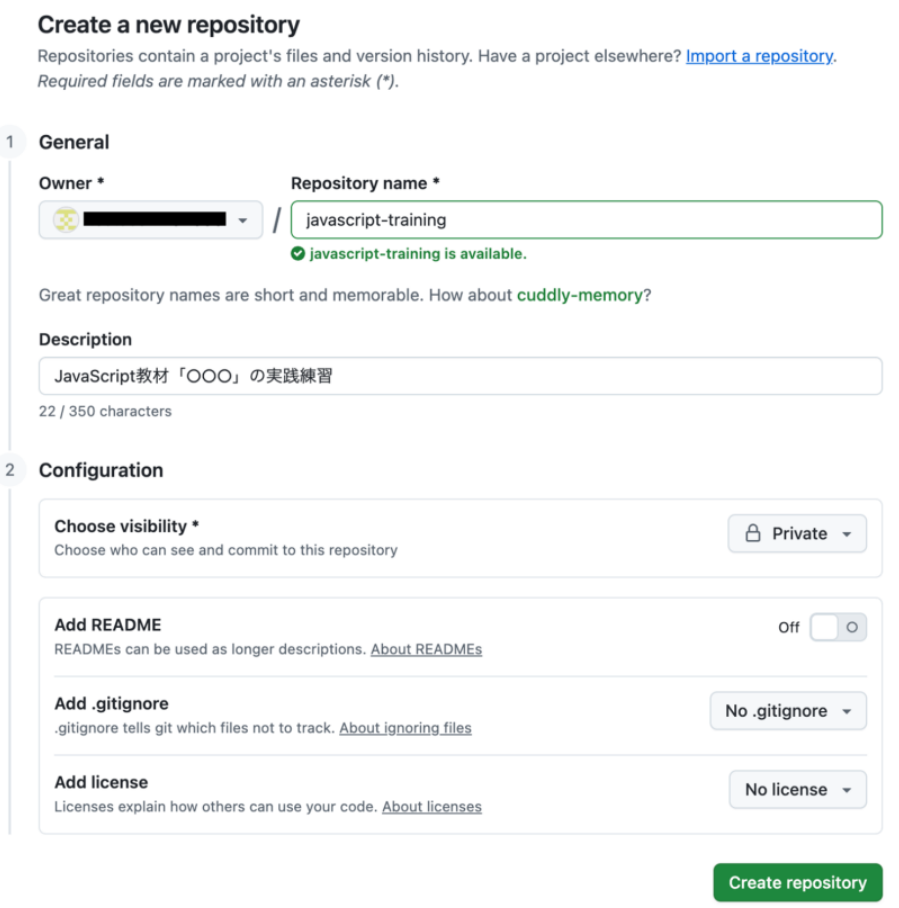
教材用に新しいリポジトリをGitHubで作成する（1）

✓ GitHubのマイページから Repositories に遷移し、「New」で新しいリポジトリを作成する



教材用に新しいリポジトリをGitHubで作成する（2）

- ✓ リポジトリ名とDescriptionは任意のものでOK
- ✓ Choose visibility を「Public」にすると、全世界に公開になる。今回は練習用なので「Private」で作成
- ✓ Create repository で進むと作成完了



プロジェクトをローカルに作成する（1）

✓ ターミナル（Windowsはコマンドプロンプト）でコマンドを入力して、学習用フォルダを作成

```
mkdir html-lesson
```

- ○ 今回の例では「html-lesson」としましたが、名前は任意です
- ○ FinderからHomeにフォルダをつくっても同じです

✓ VS Code でそのフォルダを開く

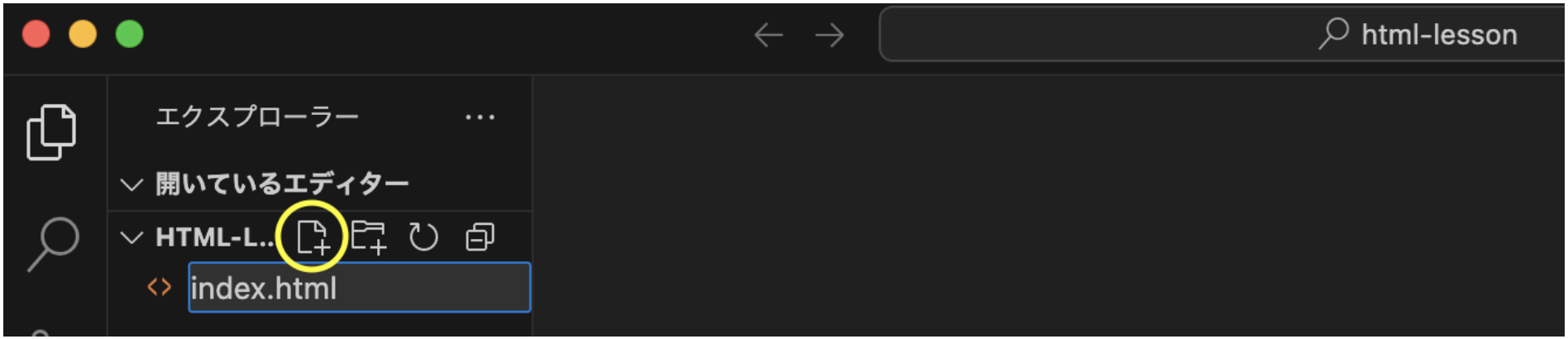
- ○ 「ファイル」→「開く」→上でつくったファイルを選択して開く

プロジェクトをローカルに作成する（2）

✓ 最初のファイルを作る（index.html など）

○ 左メニューのエクスプローラーから「新しいファイル」を選択し、
「index.html」など名前を入力してファイル作成

○ ファイル名やファイル形式は教材にしたがってください



プロジェクトをローカルに作成する（3）

✓ git init を使ってみよう

○ VS Codeのターミナルで下記を入力

```
git init
```

git init ＝ Gitリポジトリを新規作成するコマンド

○ エクスプローラー上で index.html の色が変わり、左メニューの「ソース管理」に①がつく

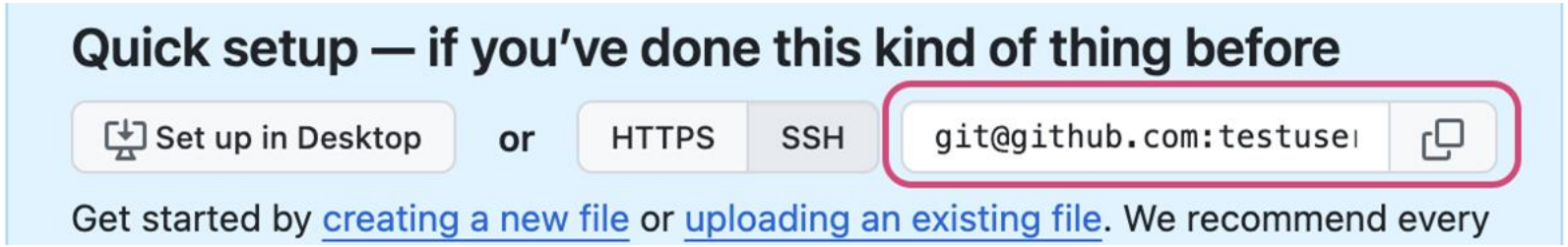
リモートリポジトリの設定（1）

✓ git remote add origin … でリモートを登録

○ VS Codeのターミナルで下記を入力

```
git remote add origin git@github.com:<USER>/<REPO>.git
```

○ git remote add origin に続く黄色い部分は、GitHubのリポジトリ画面から下記部分をコピーして入力しましょう



リモートリポジトリの設定（2）

○ すでに「origin」が設定されている場合、下記のようなエラーが出ることがあります

```
error: remote origin already exists.
```

○ この場合は、現在のリモート設定を削除してから再設定してください

```
git remote remove origin
git remote add origin git@github.com:<USER>/<REPO>.git
```

リモートリポジトリの設定（3）

- 「git remote add origin 〜」によって、このローカルリポジトリを指定したGitHub上のリポジトリとつなげることができました
- 下記のコマンドで、設定されているリモートリポジトリの確認が可能

```
git remote -v
```

```
[redacted] html-lesson % git remote add origin git@github.com:testus
[redacted] html-lesson % git remote -v
origin  git@github.com:testuser-[redacted]/javascript-training.git (fetch)
origin  git@github.com:testuser-[redacted]/javascript-training.git (push)
```

※リモートリポジトリの設定は「ステージングとコミット」の直後でも OK

ステージングとコミット（1）

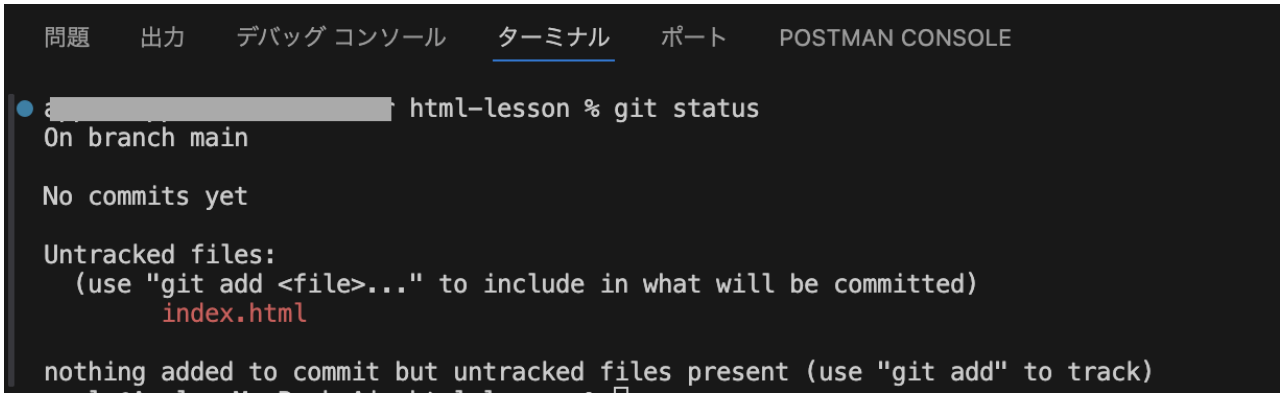
✓ git status を使ってみよう

○ VS Codeのターミナルで下記を入力

```
git status
```

git status ＝ 状態を確認する

○ index.html が “Untracked files” に出ていればOK



ステージングとコミット（2）

✓ git add を使ってみよう

○ VS Codeのターミナルで下記を入力

```
git add index.html
```

git add ●● ＝ ●●ファイルをステージング（コミット準備）に追加

git add . ＝ すべての変更をまとめてステージングに追加

⚠ git add . の注意点

- すべての変更をひとまとめにインデックスに登録してコミット可能
- 便利な一方、意図しないファイルも反映することがあるので、実務に入ったときは避けて「git add 対象ファイル名」で個別に追加したほうが安全

ステージングとコミット（3）

✓ git commit を使ってみよう

○ VS Codeのターミナルで下記を入力

```
git commit -m "メッセージ内容"
```

git commit -m “” ＝コミットを記録し、変更が履歴に保存

-mはコミットにメッセージを含めるためのオプション

○ “ ”の中は“初回のコミット・index.htmlファイルの作成” など、何を変更したかの内容を書くとベター

○ コミットはゲームにおけるセーブポイントと捉えると良い

○ コミットを使い、好きなタイミングで作業状態を保存できるようになる

プッシュ

✓ git push を使ってみよう

○ VS Codeのターミナルで下記を入力

```
git push -u origin main
```

git push ＝ ローカルの変更をリモートに送る

○ SSH鍵のパスフレーズを設定した場合は下記のように入力を求められることがあります。

```
Enter passphrase for key '/Users/apple/.ssh/id_ed25519':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 259 bytes | 259.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:[redacted]/javascript-training.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

章ごとにブランチを切って運用する（1）

ここからは「ブランチ」を使って章ごとの作業をGitHubにあげていきます。
プロジェクトの既存のコードベースから新しい「枝（ブランチ）」を作成し、そこから独立して作業を進めることを「ブランチを切る」といいます。

✓ 「fetch」と「merge」を用いて、mainブランチを最新の状態にする

○ VS Codeのターミナルで下記を入力

```
git checkout main
```

git checkout main ＝ mainブランチに移動

```
git fetch origin
```

```
git merge origin/main
```

git fetch origin ＝ リモートの変更を取得
git merge origin/main ＝ リモートのmainを現在のブランチ（main）に統合

章ごとにブランチを切って運用する（2）

✓ 「checkout -b」を用いて、作業する章のブランチを切る

○ VS Codeのターミナルで下記を入力

```
git checkout -b feature/chapter1
```

git checkout -b ブランチ名 ＝ (ブランチ名の)ブランチをつくり、そこに移動

もしくは `git switch -c feature/chapter1` でも同じことができる

※今回のブランチ名は任意ですが、チーム開発のお作法などは下記記事をご参照ください

Gitフローとブランチ命名規則

章ごとにブランチを切って運用する（3）

✓ 「git branch」を用いて、今いるブランチを確認する

○ VS Codeのターミナルで下記を入力

```
git branch
```

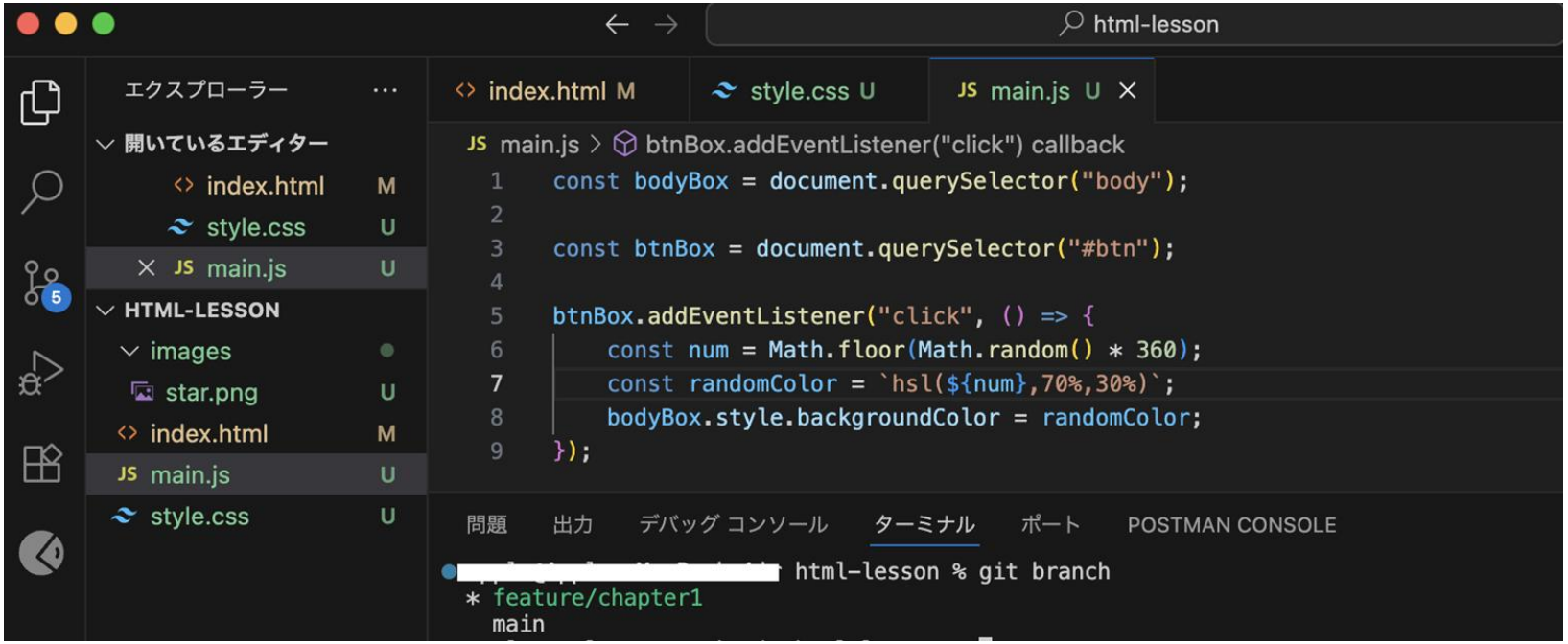
git branch ＝ 現在いるブランチとブランチ一覧が見える

○ いまはmainでなく新たに作成したブランチにいることを確認してから作業する

章ごとにブランチを切って運用する（4）

✓ 教材にしたがって、コードを書いたりファイルを編集、追加したりする

○ スクリーンショットは一例



章ごとにブランチを切って運用する（5）

✓ 変更の内容と確認

```
git status
```

どのファイルが変わったのか？を確認

```
[redacted] html-lesson % git status
On branch feature/chapter1
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .DS_Store
        images/
        main.js
        style.css

no changes added to commit (use "git add" and/or "git commit -a")
```

index.htmlが変更（modified）、
新たに作成された
imagesディレクトリ、main.js、
style.css、.DS_Store（Macが自動で作成する隠しファイル）がUntrackedとして記載されている

章ごとにブランチを切って運用する（6）

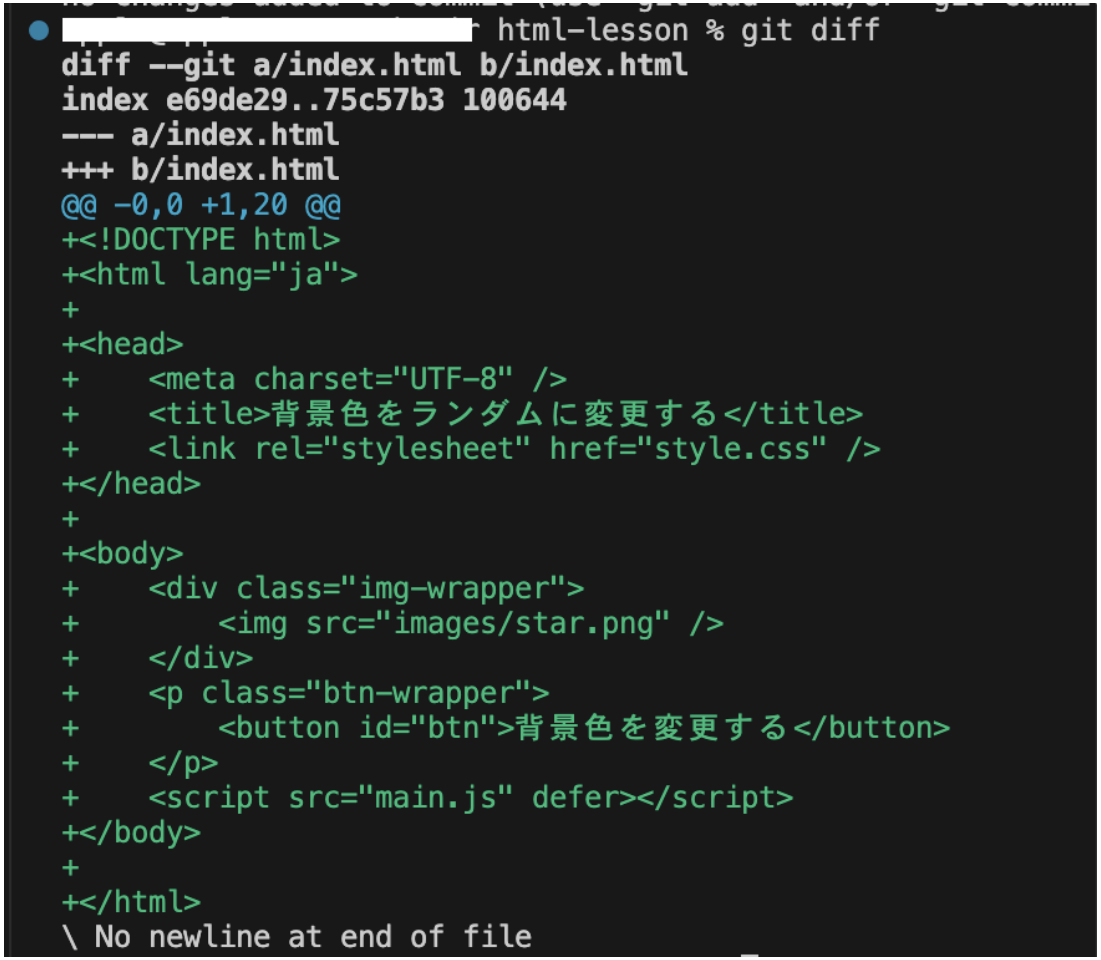
✓ 変更の内容と確認

```
git diff
```

git diff ＝ 差分の中身を確認

git diffの出力には、変更が行われた部分が「追加」や「削除」として示される。

- + が付いている行は「追加」された行。
- - が付いている行は「削除」された行。



章ごとにブランチを切って運用する（7）

✓ git add でステージング＆ステージング後の確認

```
git add index.html style.css main.js images
```

ステージングに入れたいファイルを追記する。ファイル名をすべて打たなくても、「ind」くらいまで打ってTabを押すとindex.htmlと補完して入力してくれる

.DS_Store（Macが自動的に生成する隠しファイル）は不要なのでgit add に追加しなくてOK。

.gitignoreというファイルをプロジェクト直下につくって、「.DS_Store」を記述すると誤ってgit add することなく安全に運用できる（下記参考）

【参考資料】

うさぎでもわかる 今更だが.DS_Storeとはなにか、GitHubとかに公開するとどんな危険性があるのか

章ごとにブランチを切って運用する（8）

*補足： .gitignoreの具体例

`.gitignore` には以下のようなファイル/フォルダもよく追加されます：

- `node_modules/` … npmなどで自動生成される依存ライブラリ。容量が大きく、共有不要。
- `.env` … 環境変数ファイル。APIキーやパスワードなど機密情報が入るため公開NG。
- `*.log` … ログファイル。実行環境ごとに変わるため不要。

git ignoreの設定例

```
.DS_Store
node_modules/
.env
*.log
```

章ごとにブランチを切って運用する（9）

✓ git status で再確認

```
git status
```

Changes to be committed:
→「コミットするべき内容がありますよ」の意味

```
[redacted] html-lesson % git add index.html style.css main.js images
[redacted] html-lesson % git status
On branch feature/chapter1
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   images/star.png
        modified:   index.html
        new file:   main.js
        new file:   style.css

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .DS_Store
```

章ごとにブランチを切って運用する（10）

✓ コミットとプッシュ

```
git commit -m "メッセージ内容"
```

```
git push origin ブランチ名
```

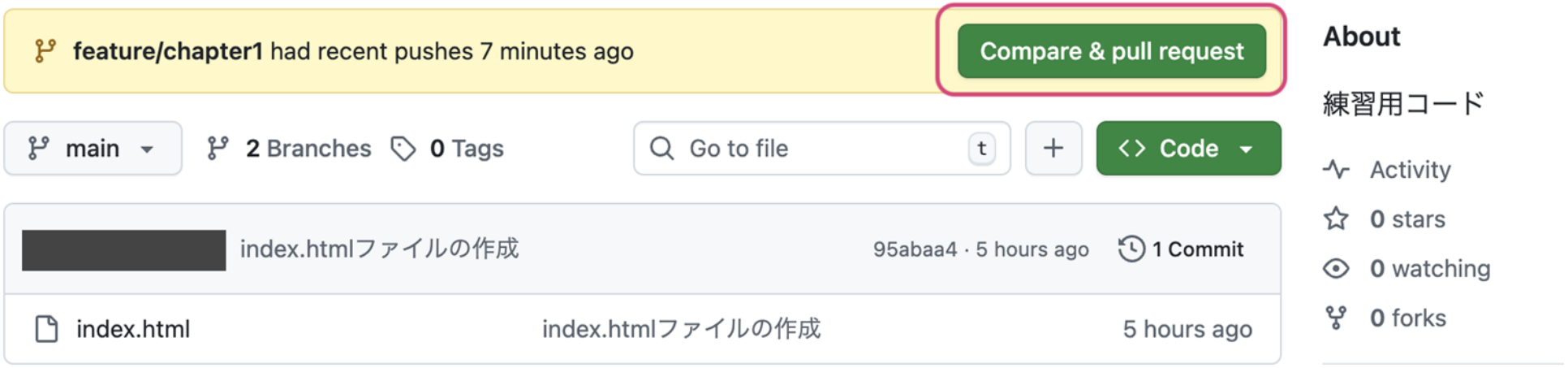
“メッセージ内容”には作業内容を記載するとベター

今回ブランチ名は「feature/chapter1」
リモートも同名でブランチが作成される

```
[redacted] html-lesson % git commit -m "第1章の実装"
[feature/chapter1 c3d631a] 第1章の実装
 4 files changed, 61 insertions(+)
 create mode 100644 images/star.png
 create mode 100644 main.js
 create mode 100644 style.css
[redacted] html-lesson %
[redacted] html-lesson % git push origin feature/chapter1
```

GitHubでPR作成 → レビューの練習 → マージ（1）

✓ GitHubのリポジトリページへ移動すると、先ほどプッシュしたブランチが表示されている



✓ 「Compare & Pull request」を押して、プルリクエストを実践してみる

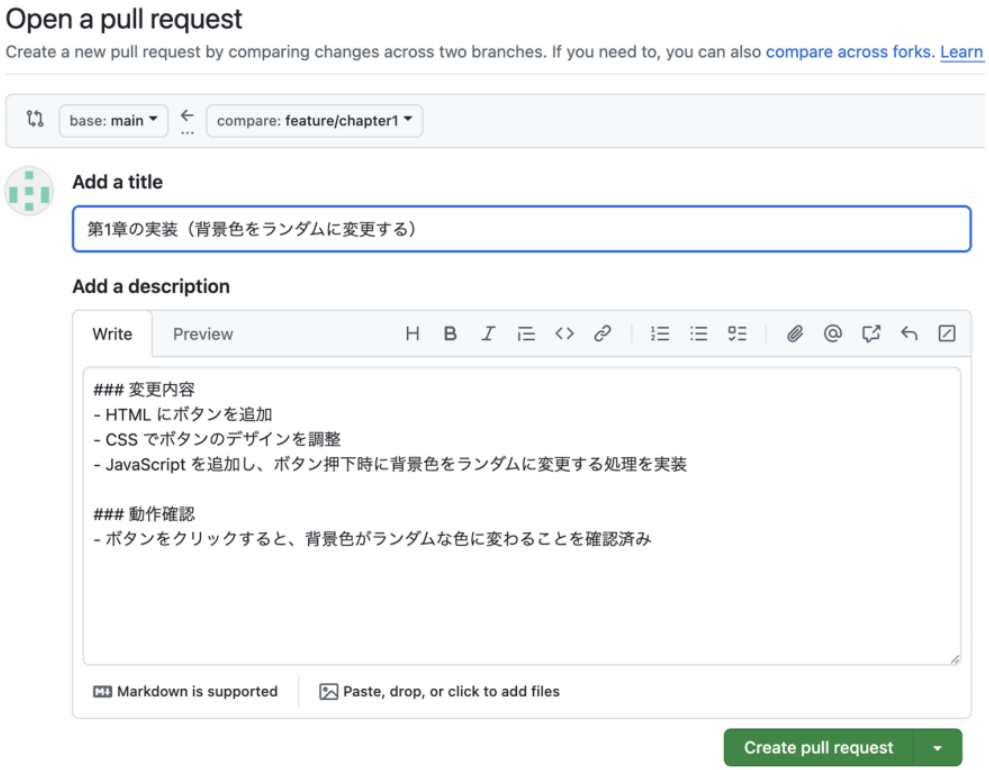
GitHubでPR作成 → レビューの練習 → マージ（2）

- ✓ プルリクエスト（プルリク / PR）とは
  - ○ 「このブランチで作業したので、main に取り込みたい」と申請する仕組み
  - ○ 本来はチームで「レビュー（内容確認）→ OKならマージ」という流れを作る
- ✓ マージとは
  - ○ プルリクが承認されたあと、ブランチの内容を main に統合する操作
  - ○ GitHub では「Merge pull request」ボタンを押すだけでOK
- ✓ 今回の自主学習でやる目的
  - ○ 「何を変えたか」が GitHub 上で視覚的に確認できる
  - ○ 作業ブランチで実験し、問題なければ main に入れる流れが身につく

GitHubでPR作成 → レビューの練習 → マージ（3）

✓ Open a pull requestでプルリクエストの内容を記載する

- ○ titleはcommitで記述したものが記載（変更や追記可能）
- ○ descriptionには行った変更をシンプルに書く
- ○ Create pull request でプルリクが送られる

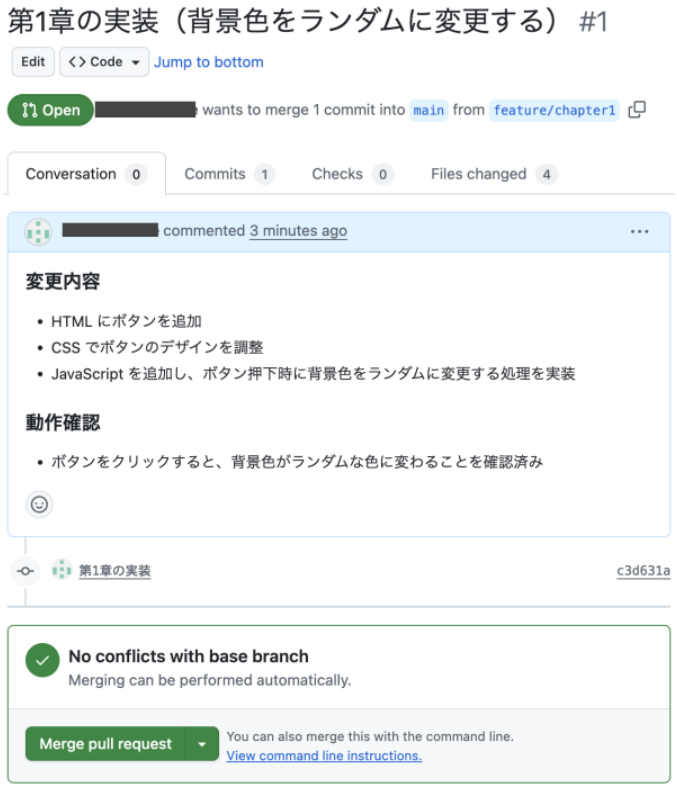


GitHubでPR作成 → レビューの練習 → マージ（4）

✓ レビュー（内容確認）→ マージの流れを体験する

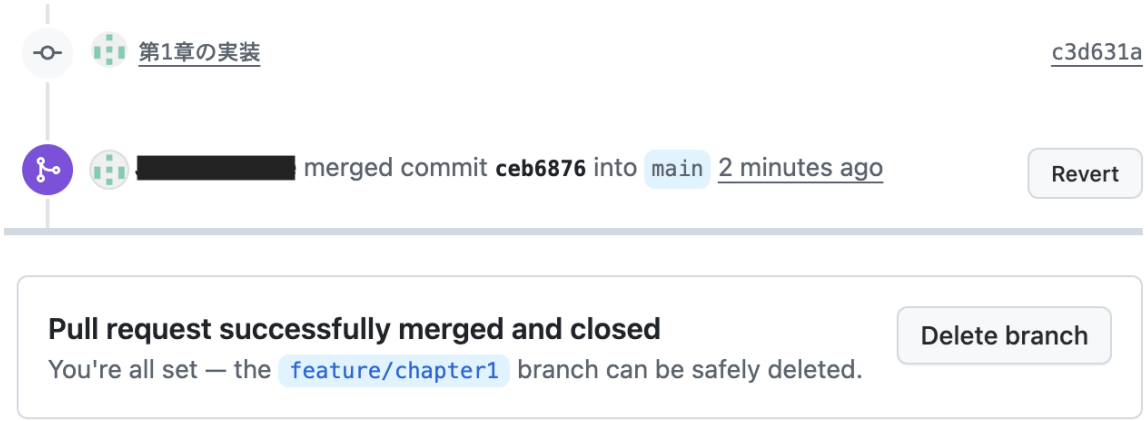
- ○ プルリクエストの Files Changedからコードを読み、必要に応じてコメントを入力
- ○ レビュー方法は下記記事が参考になります
  【GitHub】プルリクエストのレビュー方法について

- ○ 問題がない場合は「Merge pull request」ボタン →「Confirm merge」ボタンでmainブランチにマージ（統合）



GitHubでPR作成 → レビューの練習 → マージ（5）

○ 完了すると下記のようにmainブランチへマージされたこと、プルリクエストがクローズしたことが示される



○ 本来は別のレビュアーが行う作業ですが、今回は練習としてセルフで行い、流れをつかむようにする

ローカルでmainブランチの取り込み、次章のブランチ作成 （1）

✓ リモートのmain最新版をローカルに取り込む

```
git checkout main
```

```
git fetch origin
```

```
git merge origin/main
```

mainブランチに移動し、

リモートリポジトリの最新mainを取得し、

ローカルのmainにマージする

○ ローカルのmainブランチが
GitHubの最新mainブランチと
一致し、先ほどの作業で行った
変更が反映される

```
apple@Apples-MacBook-Air html-lesson % git checkout main
Switched to branch 'main'
Your branch is behind 'origin/main' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)
apple@Apples-MacBook-Air html-lesson % git fetch origin
Enter passphrase for key '/Users/apple/.ssh/id_ed25519':
apple@Apples-MacBook-Air html-lesson % git merge origin/main
Updating 95abaa4..ceb6876
Fast-forward
 images/star.png | Bin 0 -> 26025 bytes
 index.html      |  20 ++++++++++++++++++++
 main.js         |   9 +++++++++
 style.css       |  32 ++++++++++++++++++++++++++++++++
 4 files changed, 61 insertions(+)
 create mode 100644 images/star.png
 create mode 100644 main.js
 create mode 100644 style.css
```

ローカルでmainブランチの取り込み、次章のブランチ作成 （1）

✓ 次章のブランチを切って移動し、作業を開始する

```
git checkout -b feature/chapter2
```

もしくは

```
git switch -c feature/chapter2
```

○ git branch で今の作業ブランチを再確認すると安心

```
● [redacted] html-lesson % git checkout -b feature/chapter2
Switched to a new branch 'feature/chapter2'
● [redacted] html-lesson % git branch
  feature/chapter1
* feature/chapter2
  main
```

まとめ　～今回の個人開発の流れ～

▼ 環境構築が完了

▼ GitHubでリポジトリを新規作成（private）

▼ `mkdir フォルダ名` などでプロジェクトをローカルに作成

▼ `git init` で新しいGitリポジトリを作成する

▼ `git remote add origin git@（以下省略）` で、ローカルのリポジトリをGitHub上のリポジトリに紐づける（`git remote -v` で確認可能）

▼ `git status` で状態を確認し、`git add` でステージングに追加

▼ `git commit -m “メッセージ内容”` でコミット

▼ `git push -u origin main` でプッシュし、ローカルの変更をリモートに送る

まとめ　～章ごとのブランチ運用～

▼ `git checkout main` mainブランチに移動（他にブランチがなければ不要）

▼ `git fetch origin` `git merge origin/main` で最新のmainを取り込む

▼ `git checkout -b ブランチ名` で新たなブランチを作成し、そこに移動

▼ ファイルを編集し、`git status` `git diff` で変更内容と差分を確認

▼ `git add` でステージングに入れ、`git status` で再確認

▼ `git commit -m “メッセージ内容”` でコミット

▼ `git push origin ブランチ名` でプッシュし、同名のブランチがリモートに作成

▼ GitHub上でプルリク＆（今回は自主学習のため）自分でコードレビュー＆マージ

▼ 再び一番上の `git checkout main` に戻り、最新のmainを取り込み、次の章のブランチを作成してそこで作業を行う

参考資料一覧

【環境構築】GitHubアカウントを作成してVSCodeと連携できるようにする

Visual Studio Code のインストール方法

Gitをインストールしてみよう！Windows/Macどちらも丁寧に解説

【入門】GitHubの使い方｜設定や基本操作など

Git を macOS にインストールする

GitHub の noreply メールアドレスの設定と、スパム業者に関するtips

GitHubのSSH key作成と設定方法

GitHub コミットメールアドレスを設定する

Gitフローとブランチ命名規則

うさぎでもわかる 今更だが.DS_Storeとはなにか、GitHubとかに公開するとどんな危険性があるのか

【GitHub】プルリクエストのレビュー方法について

コミットを適切に分けよう：git add . の危険性と代替策

Git入門：初心者向けの基本操作と概念

未経験エンジニアにGitとGitHubは必修項目

ご覧いただきありがとうございました

DESIGN
ENGINEERING