

## はじめに (という名の免責)

こんにちは。工藤研M2のかのんといいます。顔→  
今回この Git 練習会に興味を持ってくださってありがとうございます。



まずは皆さんに謝らせてください。私、実は Git 超弱者です、ごめんなさい。  
ただ、弱者は弱者なりに伝えられることもあると思い、今回この会をやることにしました。

私が過去に参加したインターン(Google, ヤフー)でゼーハーしながらもなんとか耐え抜いた超絶最低限の Git 知識をお伝えできたらと思います！

## [TEAra/git-practice](#) リポジトリについて

Git は多くのIT企業が採用するシステムで、インターンに参加するときの条件に「git が最低限使える」みたいなものがある会社もあります。

ですが、普段お一人様で共同開発の練習をする機会はそんなにあるわけでもありません。  
私も「やったことないから怖い」という恐怖心がなかなか消えなくて苦労しました(今もあります)。

なので、このリポジトリは「[git に関する実験をなんでもやっていい](#)」場所にしたいと思います。  
つまり、

- ファイルを足してもOK
- 消してもOK
- 使ったことのないGitHubの機能を使うのもOK
- レビューを誰に頼んでもOK
- どんな失敗をしてもOK
- このリポジトリ内の同志を傷つけるようなことでなければ何でもOK

ということです。

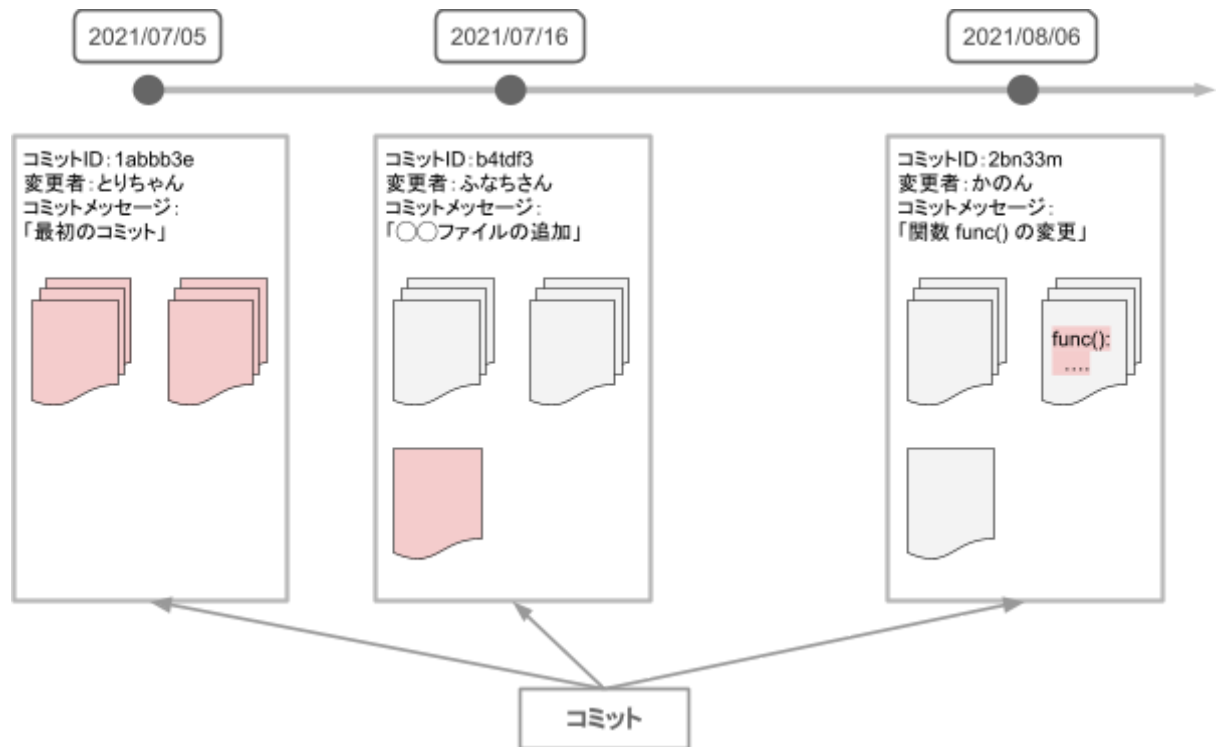
「git に関して試したいことがあるけど、インターンでやらかしたくはない」  
みたいな人には打ってつけの場だと思います。  
ぜひ有効活用してください。

## Git ってなに？

バージョン管理システムの一つ。  
ひとつのプロジェクトに対して、

「いつ、誰が、どのファイルに、どのような変更を加えたか」

を全て記録してくれる便利ツールのことです！



時間が経つにつれて形を徐々に変えていくプロジェクト、例えば

- WebアプリやAndroidアプリの開発
- 研究用に書いたコード

これらのプロジェクトに関わっていると、

「1年前ってあの関数のコードはどのように書かれてたっけ？」

「昨日マージした変更部分がバグってるから、その1個前のバージョンに戻りたい...」

みたいなことがどうしても起こります。

そんなときに、プロジェクトに関する全ての歴史を記憶してくれているバージョン管理システムが  
助けてくれる、というわけです！！

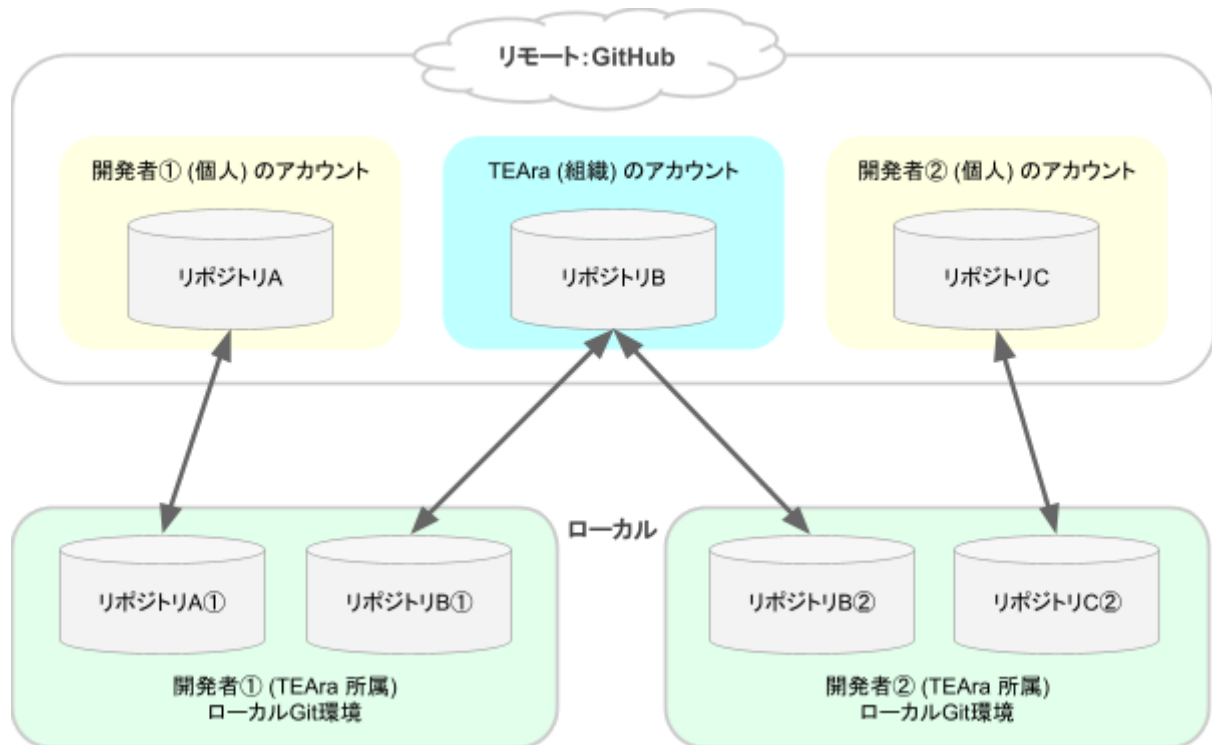
## GitHubってなに？

hub : 「中心」みたいな意味。

Gitで管理されているプロジェクト・プログラムをオンライン上で管理できるサービスみたいなもの(だと解釈している)。

個人のGitユーザーをクラウド(Hub)に集めているイメージ。

- 複数人でひとつのプロジェクト(リポジトリ)を共有して作業できる。
- GitHubにコードをアップしておけば、ローカル環境が壊れたり変化したりしても別の端末で開発することができる。



## Git の用語

### リポジトリ (repository)

プロジェクトのこと。

意味は違うけれど、プロジェクトに関係するファイル・フォルダを全部含んだ一番外側のディレクトリのことを指している、と思ってもいいかもしれない。

例:

- 前回ワークショップで作ったweb-frontend
- Androidアプリのプロジェクト
- 小さいpythonファイルひとつ (自分がそれをひとつのプロジェクトと見なせばそれも立派なリポジトリ)

## コミット (commit)

リポジトリに加えた変更の単位。

そのリポジトリの歴史の1ステップのことだと思ってもいいかも。

それぞれのコミットには固有のIDが割り当てられている。

## ブランチ (branch, 枝のこと)

本番環境に直接変更を加えて、もし失敗しちゃったら怖い。

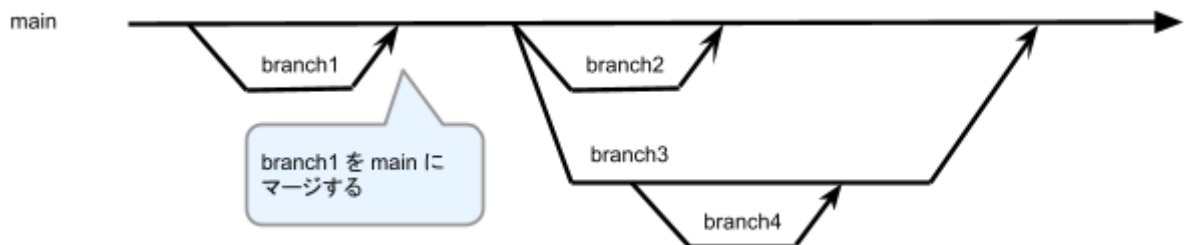
そういうときに便利なのがブランチです。

新しくブランチを作ってそこに変更を加えても元のブランチには何も影響を与えません。

試しに〇〇やってみたい、と思ったらブランチを作って、

- うまくいったら、本番環境に組み込む(マージする)
- うまくいかなかったら、ただそのブランチを削除すればいい

という感じです。



<よくあるパターン>

本番環境 : main ブランチ

開発環境 : develop ブランチ

main, develop を常に最新の状態に保ち、

何か変更を加えるときは develop から新しくブランチを切って、

それを develop ブランチにマージしていく、というやり方。

## ローカル (local)

自分のパソコンで開発している git リポジトリのことだと思っていれば大丈夫だと思う。

## リモート (remote)

GitHub上で管理されているリポジトリ達のことだと思っていれば多分大丈夫。

自分のアカウントのリポジトリの場合もあるし、会社や組織(ex: TEAra)のアカウント内のリポジトリの場合もある。

## プルリクエスト (pull request)

自分の開発した機能などに関わる変更をリモートのリポジトリに組み込んでもらうときに作るリクエストのこと。

共同開発のときにめっちゃめっちゃ耳にしたいと思います。

## マージする (merge)

自分の加えた変更を本番環境などに組み込ませること。  
組み込み先は main ブランチだけじゃなく、そのときの状況によって変わります。

## マージコンフリクト (merge conflict)

例えば、自分と別の開発者が同じファイルの関係のある同じ箇所をそれぞれ違った感じで変更しようとする、それはかち合っていると言える。この状態をマージコンフリクトといい、後から変更を加えようとした人が手動でコードを直す必要がある。

# 個人開発のやり方

## 一連の流れ

あらかじめGitHubで管理したいファイルが全て入っているディレクトリに移動しておく。

### 最初の手続き

1. ディレクトリの中で git init コマンドを打って git リポジトリとして初期化する。
2. git add, git commit で最初のコミットを作る。
3. GitHubの自分のアカウントで新しくリポジトリを作成する。
4. git remote add origin https://...で、リモートリポジトリを登録する。
5. git branch -M main で今いるブランチを main に変えておく。
6. git push origin main でローカルのファイルをリモートのリポジトリ (GitHub上) にアップロードする。

あとはこのローカルのリポジトリで開発して、その変更をリモートのリポジトリにアップロードしていく、という流れです！

## 練習①「kyo-proディレクトリを作って、GitHubにあげてみよう！」

```
$ mkdir kyo-pro  
$ cd kyo-pro
```

kyo-pro/ に新しいファイルを作ってみましょう！

なんでも大丈夫です！例えば、おなじみのHello Worldなら...

```
kyo-pro/helloworld.c

// 初めてのプログラム

# include <stdio.h>

int main() {
    printf("Hello World!");

    return 0;
}
```

ファイルを作り終わったら、とうとう git を使っていきます！

```
$ pwd
/Users/canon/TEAra2/kyo-pro

$ ls
helloworld.c

### この kyo-pro ディレクトリを git で管理しますよ！という宣言コマンド
$ git init
Initialized empty Git repository in /Users/canon/TEAra2/kyo-pro/.git/

### 今の git の状態を確認できるコマンド
$ git status
On branch master

No commits yet

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

    helloworld.c

nothing added to commit but untracked files present (use "git add"
to track)

### 今の変更を"一旦"保存するコマンド
$ git add .

$ git status
On branch master
```

```
No commits yet
```

```
Changes to be committed:
```

```
(use "git rm --cached <file>..." to unstage)
```

```
new file:   helloworld.c
```

```
### 今までの変更を1つの"コミット"として保存するコマンド
```

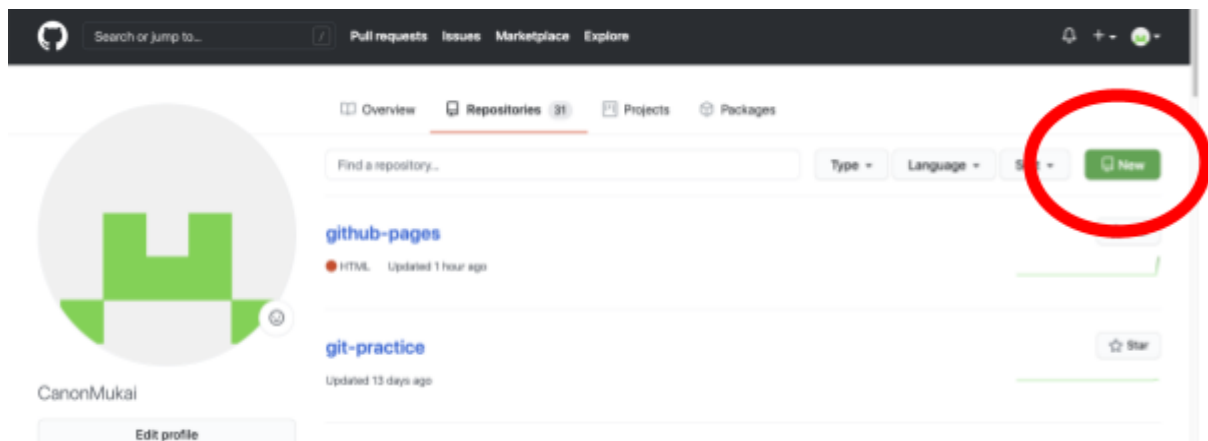
```
$ git commit -m "最初のコミット"
```

```
[master (root-commit) a60d232] 最初のコミット
```

```
1 file changed, 9 insertions(+)
```

```
create mode 100644 helloworld.c
```

ここから、オンラインのGitHub上で新しいリポジトリを作りましょう！



次のようなページにいきます。

## Create a new repository

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

Owner \*

CanonMukai ▼

Repository name \*

kyo-pro ✓

リポジトリの名前を入力します。  
kyo-proでも他の好きな名前でもOK！

Great repository names are short and memorable. Need inspiration? Read about [fuzzy-succotash?](#)

Description (optional)

☒ **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ **Add a README file**

This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**

Choose which files not to track from a list of templates. [Learn more.](#)

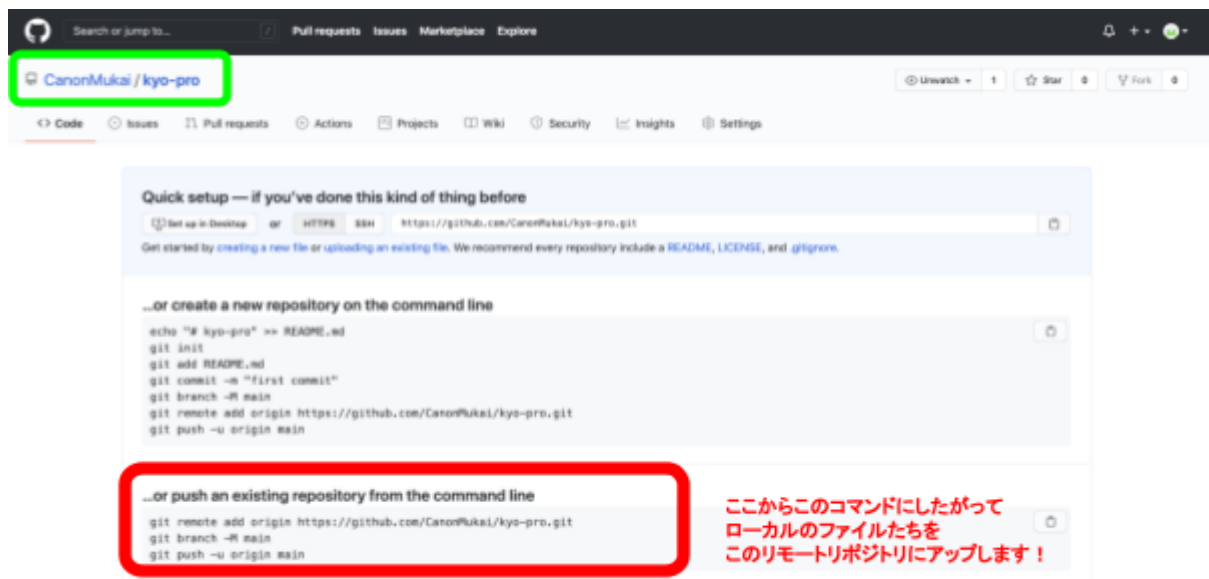
☐ **Choose a license**

A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

あとはデフォルトのまま  
← Create Repository 押しましょう！！

リモートに kyo-pro というリポジトリが作成されました！



Search or jump to... Pull requests Issues Marketplace Explore

CanonMukai / kyo-pro

Unwatch 1 Star Fork

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

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

Set up in Desktop or HTTPS SSH `https://github.com/CanonMukai/kyo-pro.git`

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

...or create a new repository on the command line

```
echo "# kyo-pro" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/CanonMukai/kyo-pro.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/CanonMukai/kyo-pro.git
git branch -M main
git push -u origin main
```

ここからこのコマンドにしたがって  
ローカルのファイルたちを  
このリモートリポジトリにアップします！

ターミナルに戻って、



```
### ローカルにリモートリポジトリを"origin"という名前で登録する
$ git remote add origin https://github.com/CanonMukai/kyo-pro.git

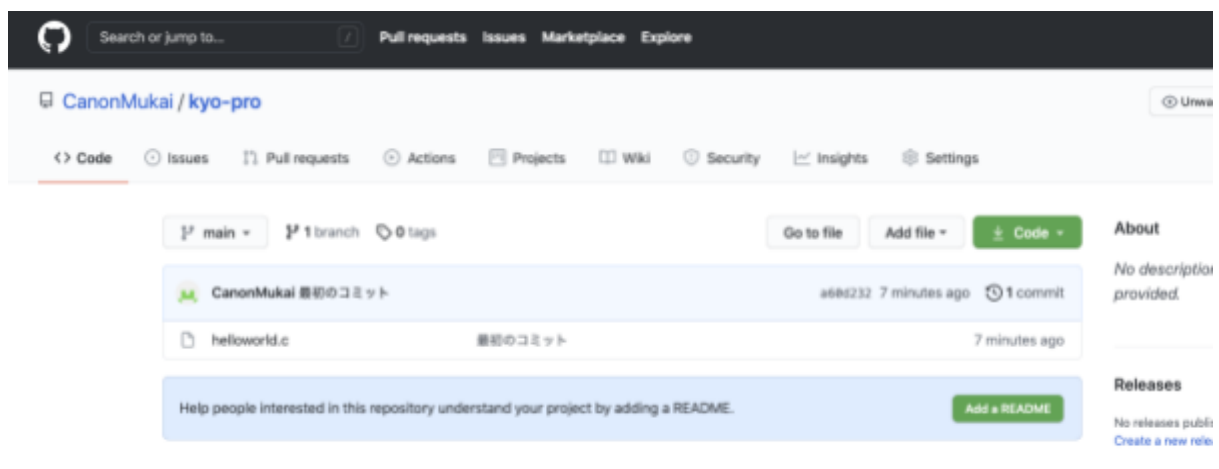
### リモートリポジトリの登録名を確認
$ git remote
origin # リモートリポジトリの呼び名です。ややこし

### 今いるブランチの名前を main に変えます。
$ git branch -M main

### ブランチ一覧を確認できるコマンド
$ git branch
* main # 今いるのが main ブランチだと確認できた

### さっき作った変更(コミット)をリモートリポジトリにアップロードする
$ git push -u origin main
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 359 bytes | 359.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/CanonMukai/kyo-pro.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

これでリモートリポジトリに kyo-pro/ の内容がアップロードされたはず！  
見てみましょう。  
さっきのGitHubのページをリロードすると、helloworld.cがアップされています！！



これで、kyo-pro/ がしっかり開発できるリポジトリになったので、

次は kyo-pro/helloworld.c を変更してみましょう！

次の練習が開発する中でもっとも多く繰り返す過程になると思います。

## 練習②「ブランチを作ってコミットしてみよう！」

helloworld.c の print する内容を変えるという変更(コミット)を作って、それをGitHubにアップしたいと思います。

オススメの開発方法は

1. main ブランチを元に別のブランチ(ex: branch\_a)を作る (俗に「ブランチを切る」という)
2. branch\_a に切り替えて、変更を保存(コミット)する。
3. main ブランチに切り替えて、branch\_a をマージする。
4. リモートリポジトリに push する。

この方法では、main ブランチを常に理想の状態に保つことができます。

では、まずブランチを作ってそこで作業しましょう！

```
### ブランチャー一覧 & 現在どのブランチにいるかを確認
$ git branch
* main

### change-printf という名前のブランチを作成
$ git branch change-printf

$ git branch
change-printf  # ブランチが作成されている
* main         # でも、まだ main ブランチにいます

### change-printf ブランチに切り替える
$ git checkout change-printf
Switched to branch 'change-printf'

$ git branch
* change-printf  # ブランチが切り替わった！
main
```

change-printf というブランチにいる状態でファイルを編集していきます。

```
kyo-pro/helloworld.c
```

```
// 初めてのプログラム
```

```
# include <stdio.h>

int main() {
    printf("Gitは最低限使うことができれば上出来だと思う。"); // 変更したよ!

    return 0;
}
```

変更を加えてファイルを保存したら、add, commitしていきます！

```
### add して
$ git add .

### commit する（説明になってねえ）
$ git commit -m "printfの内容の変更"
[change-printf f35b29d] printfの内容の変更
1 file changed, 1 insertion(+), 1 deletion(-)

### 今いるブランチの状態を確認
$ git status
On branch change-printf
nothing to commit, working tree clean

### 新しい変更がコミットとして保存されているのを確認しよう
$ git log --oneline
f35b29d (HEAD -> change-printf) printfの内容の変更 # あったあった
a60d232 (origin/main, main) 最初のコミット
```

今まで change-printf というブランチで作業して、このブランチにコミットしました。  
これを本流である main ブランチに組み込んでいきたいと思います。  
これを「change-printf を main にマージする」といいます！

```
### change-printf ブランチにいることを確認
$ git branch
* change-printf
  main

### main ブランチに切り替える
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
```

```
$ git branch
  change-printf
* main

### change-printf のコミットを main ブランチにマージしよう
$ git merge change-printf
Updating a60d232..f35b29d
Fast-forward
 helloworld.c | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

これで main ブランチのprintfの内容も変わったと思います！  
あとはリモートにプッシュしましょう！

```
### change-printf ブランチにいることを確認
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 443 bytes | 443.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/CanonMukai/kyo-pro.git
 a60d232..f35b29d  main -> main
```

これでリモートリポジトリも更新されました！！

この過程を繰り返していけば、開発したものをGitHubにアップすることができるし、今までの変更の歴史を保存することができます。

## 共同開発のやり方

### 一連の流れ

1. 開発に参加するリモートリポジトリをローカルにクローンする。
2. mainブランチを元に新しい作業用ブランチを切る。
3. 開発する！
4. 3の変更を add, commit し、リモートブランチに push する。
5. GitHub上のリモートブランチにて、プルリクエストを作る。
6. (他の人にレビューしてもらい、LGTMをもらえたらマージする。)
7. ローカルのmainブランチを git pull で最新の状態にする。
8. 2へ！

### 練習③「自己紹介ページを追加しよう！」

TEAraのGitHubで [git-practice](https://github.com/TEAra-official/git-practice) というリポジトリを作りました。  
「超シンプルなGit練習会ホームページの開発」という設定です。  
GitHub Pagesという機能を用いて、<https://teara-official.github.io/git-practice/> で公開しています。

まずはここに、皆さんの自己紹介ページを追加する練習をしてみましょう！

かのん(工藤研M2)の場合、URLは

<https://teara-official.github.io/git-practice/member/canon.html>

になります。

```
### git-practice を展開したいディレクトリにて、
$ git clone https://github.com/TEAra-official/git-practice.git
Cloning into 'git-practice'...
remote: Enumerating objects: 14, done.
remote: Counting objects: 100% (14/14), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 14 (delta 3), reused 9 (delta 3), pack-reused 0
Unpacking objects: 100% (14/14), done.

$ ls
git-practice

$ cd git-practice

### この tree コマンドはインストールしていないと使えませんが、
### フォルダ階層を見るだけなのでスキップしてもらって大丈夫です^^
$ tree
.
├── README.md
├── index.html
├── member
│   └── canon.html
└── member.html

###
$ git branch
* main

### ●●は自分の名前、例えばとりちゃんなら add-torichan みたいな感じで！
$ git branch add-●●
```

```
$ git branch
  add-●●
* main

$ git checkout add-●●
Switched to branch 'add-●●'

$ git branch
* add-●●
  main
```

ここまできたら、git-practice/member/の下に○○.htmlを足しましょう！  
git-practice/member/canon.htmlをコピーして編集するとラクです。  
「かのん」にまつわるところを自分好みに変えてみてください。

編集が終わって、これをマージしようと思えば、  
add, commit, push しましょう！

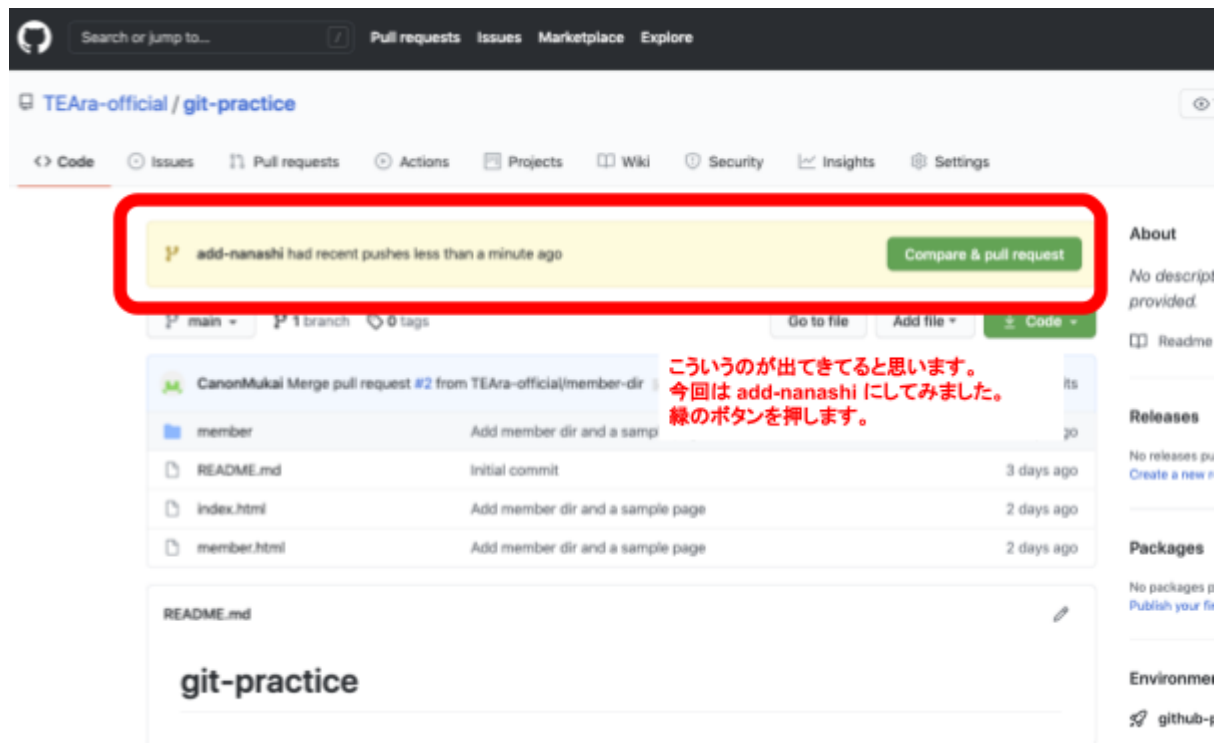
```
$ git add .

$ git commit -m "●●.htmlの追加"
[add-●● f114368] ●●.htmlの追加
1 file changed, 14 insertions(+)
create mode 100644 member/●●.html

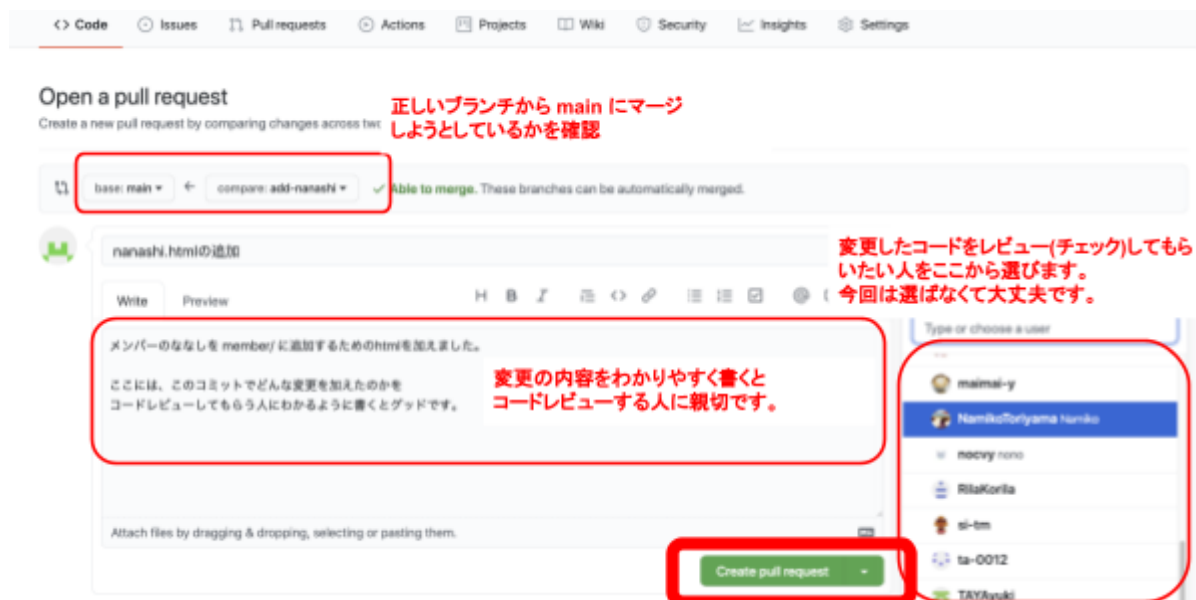
$ git log --oneline
f114368 (HEAD -> add-●●) ●●.htmlの追加
...

$ git push origin add-●●
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 627 bytes | 156.00 KiB/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local
object.
remote:
remote: Create a pull request for 'add-●●' on GitHub by visiting:
remote:
https://github.com/TEAra-official/git-practice/pull/new/add-●●
remote:
To https://github.com/TEAra-official/git-practice.git
* [new branch]      add-●● -> add-●●
```

これで、GitHub上の TEAra/git-practice に黄色い枠線で囲まれた箇所が出てきます。



Compare & Pull request ボタンを押すと



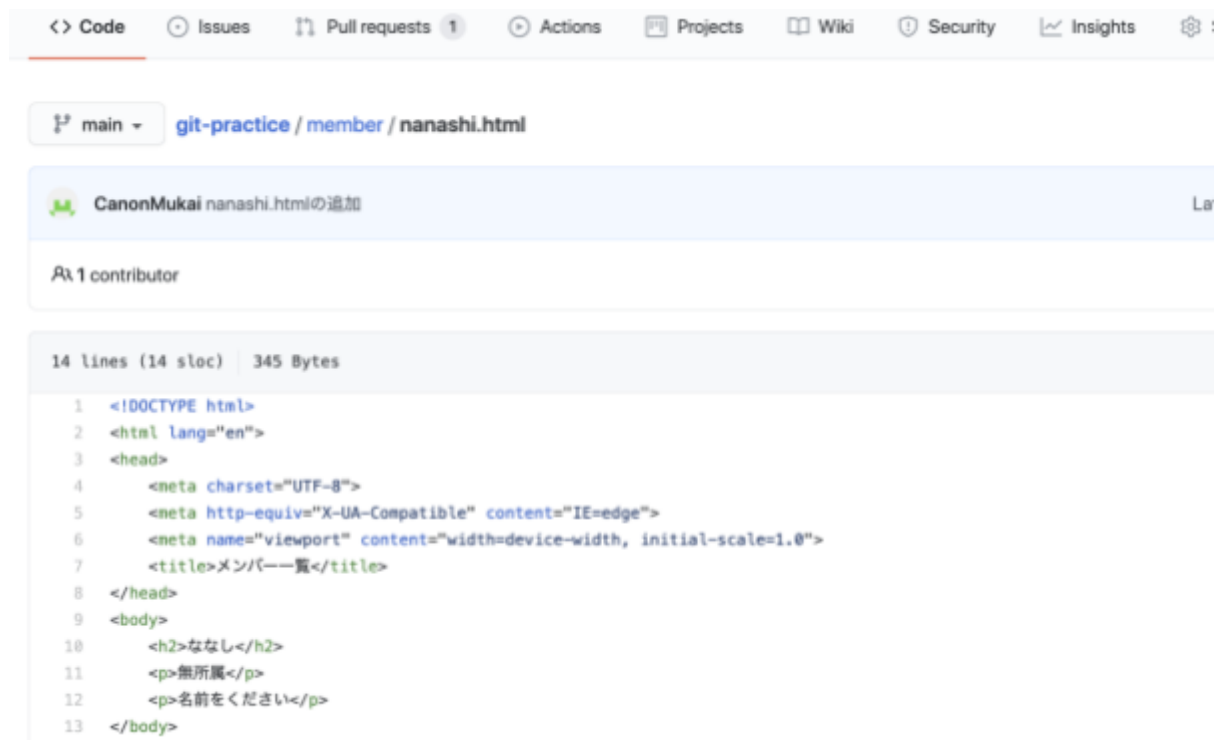
Create pull request ボタンを押すと、プルリクエストが作成されました！



Merge pull request ボタンを押すと、リモートリポジトリに自分の作った変更がマージされます！

<> Code タブに行って確認してみましょう。

下の図みたいに、ちゃんと nanashi.html が追加されています。



<https://teara-official.github.io/git-practice/member/nanashi.html>

このURLにいてみると、ななしさんのページが追加されていることがわかります。



皆さんは <https://teara-official.github.io/git-practice/member/〇〇.html> の〇〇を自分の名前に読み替えて行ってみてください！

最後に、ローカルの main ブランチを最新の状態にする方法です！

```
$ git branch
* add-●●
  main

$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

$ git branch
add-●●
* main

$ git pull origin main
remote: Enumerating objects: 14, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 6 (delta 3), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (6/6), done.
From https://github.com/CanonMukai/github-pages
 * branch                main          -> FETCH_HEAD
  db1c9ea..17339ae      main          -> origin/main
Updating db1c9ea..17339ae
Fast-forward
 member.html            | 1 +
 member/member3.html    | 0
 member/member4.html    | 0
 3 files changed, 1 insertion(+)
 create mode 100644 member/member3.html
 create mode 100644 member/member4.html
```

ブランチの削除

add-〇〇はもういらないな、と思ったら以下のコマンドで削除できます。

```
$ git branch -D add-●●
```

## 練習④「マージコンフリクトを解消しよう！」(時間があれば)

git-practice/member.html に次の一行を足してプルリクエストを作ってください。

```
$ git branch -D add-●●  
  
$ git commit
```

## 出てきたコマンドのまとめ

```
### 今いるローカルのディレクトリを git のディレクトリにする  
$ git init  
  
### GitHub上で公開されている任意のリポジトリをローカルに持ってこれる  
$ git clone https://...  
  
### ローカルのブランチの一覧を確認できる  
$ git branch  
  
### ○○という名前の新しいブランチを作る  
$ git branch ○○  
  
### 今いるブランチから○○というブランチに切り替える  
$ git checkout ○○  
  
### 加えた変更を一旦保存、的。ややこし  
$ git add .  
  
### add してた変更タイトルをつけて保存  
$ git commit -m "コミットのタイトル・説明"  
  
### originっていう名前のリモートリポジトリの○○っていうブランチに  
### ローカルで作ったコミットを送る  
$ git push origin ○○  
  
### 今いるブランチの状態を確認できる  
$ git status
```

```
### 直近のコミット一覧のタイトルや日付などの情報を確認できる
$ git log
### --onelineオプションをつけるとコミットのタイトル一覧だけ表示できて便利
$ git log --oneline

### origin というリモトリポジトリの main というブランチの最新状態を
### ローカルに持ってこれる
$ git pull origin main

### ローカルの〇〇というブランチを削除できる
$ git branch -D 〇〇

### 今いるブランチに〇〇というブランチの変更内容を組み込む(マージする)
$ git merge 〇〇
```

## 出てきてないけど便利なコマンドまとめ

```
### 1こ前のコミットに戻る
$ git reset --hard HEAD^

### 今いるブランチを元に新しいブランチ〇〇を作り、〇〇をcheckoutする
$ git checkout -b 〇〇
```