

# 計算機科学実験及演習3 ハードウェア 「Gitの使い方」

京都大学情報学科計算機科学コース

## 実験3 ハードウェアでは、バージョン管理と設計データ・課題提出にGitHubを 사용합니다。

### ① Gitの概要

バージョン管理システムとGitの基本概念を紹介

### ② Gitチュートリアル

Gitの基本的使い方を簡単な例を用いて説明

### ③ リモートリポジトリ・GitHubの概要

本実験で使用するリモートリポジトリGitHubの紹介

### ④ GitHubチュートリアル

GitHubの使い方を説明

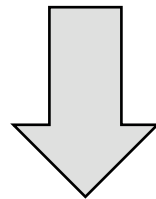
### ⑤ GitHub Classroom

GitHub Classroomの説明

# ①Gitの概要

## PCでの作業で起こりうること

- エラーを含んだ状態でプログラムを保存してしまった。
- レポートのファイルを間違えて削除してしまった
- 1週間ぶりに開くファイル、以前どんな編集をしたのか忘れてしまった。
- 二人で1つのファイルを編集してしまい、一方の人がした編集が反映されなかった。



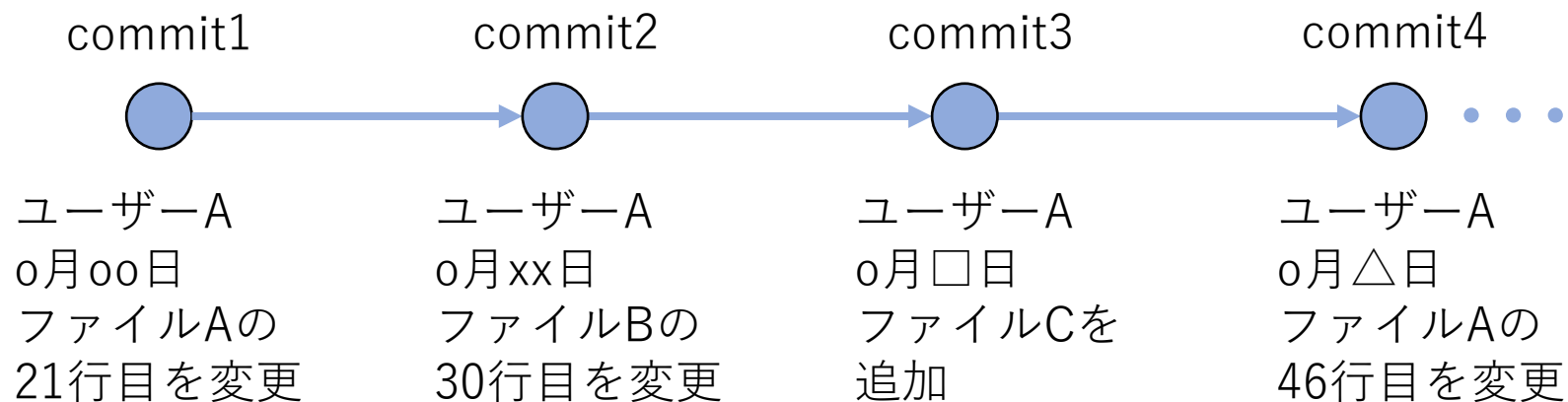
適切なバージョン管理が必要

## Gitによるバージョン管理

- Gitは分散型のバージョン管理システム。
- Linuxのソースコードを管理するためにLinus Torvalds自身が開発。
- Gitの機能
  - ファイルの変更履歴を保存する。
  - 履歴には、いつ、誰が、どんな変更を行ったかを記録できる。
  - いつでもファイルを保存した履歴の状態に戻せる。
  - 他人が編集したファイルを上書きしようとするとき警告を出す。

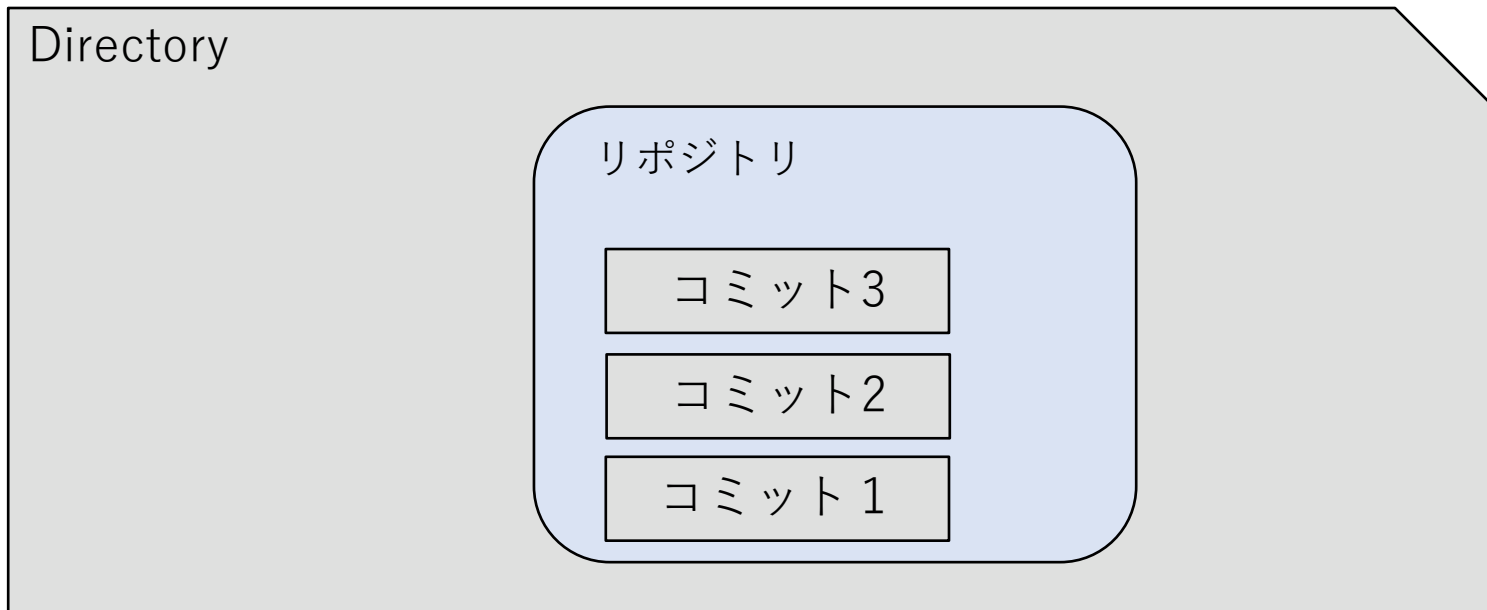
## commit

- Gitでファイルの変更履歴をつけていくことを**commit**という。
- commitには、**いつ、誰が、どんな変更**を行ったかを記録しておける。
- いつでも、任意のcommitに戻ることができる。



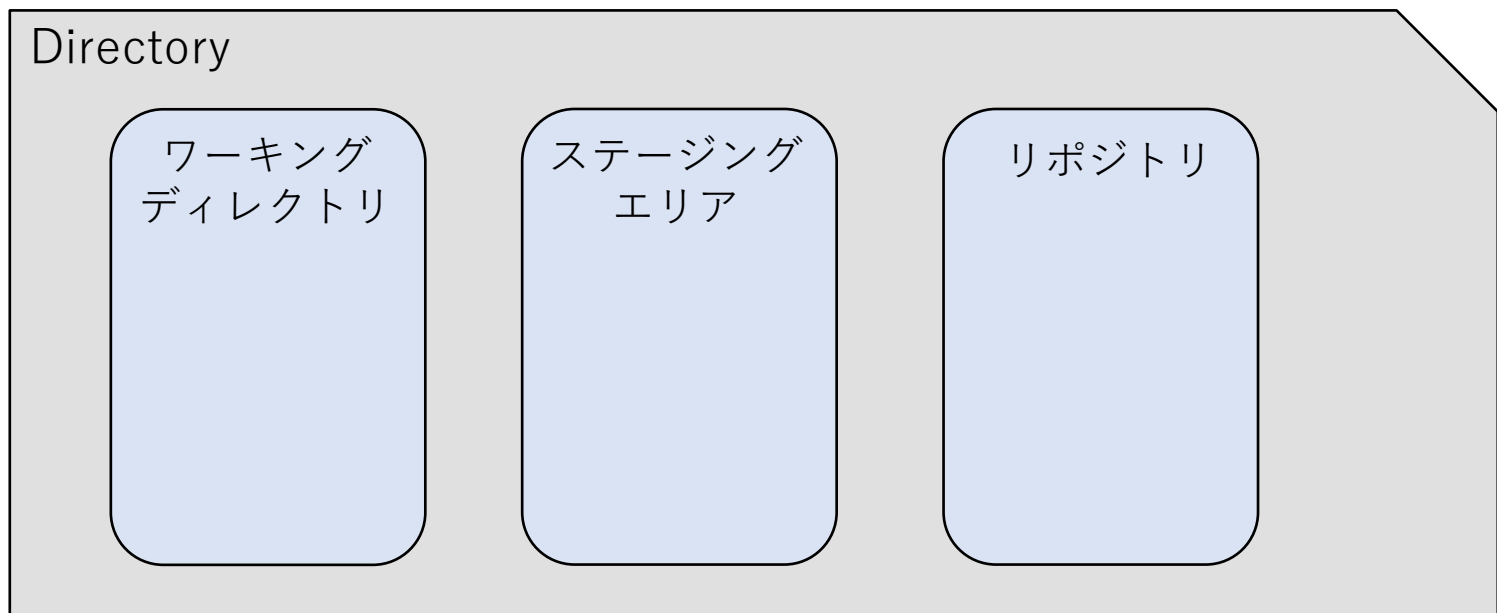
## リポジトリ

- commitは **リポジトリ** と呼ばれる場所に保存されていく。
- commitするたびに、その時のファイルの状態が保存されていく。



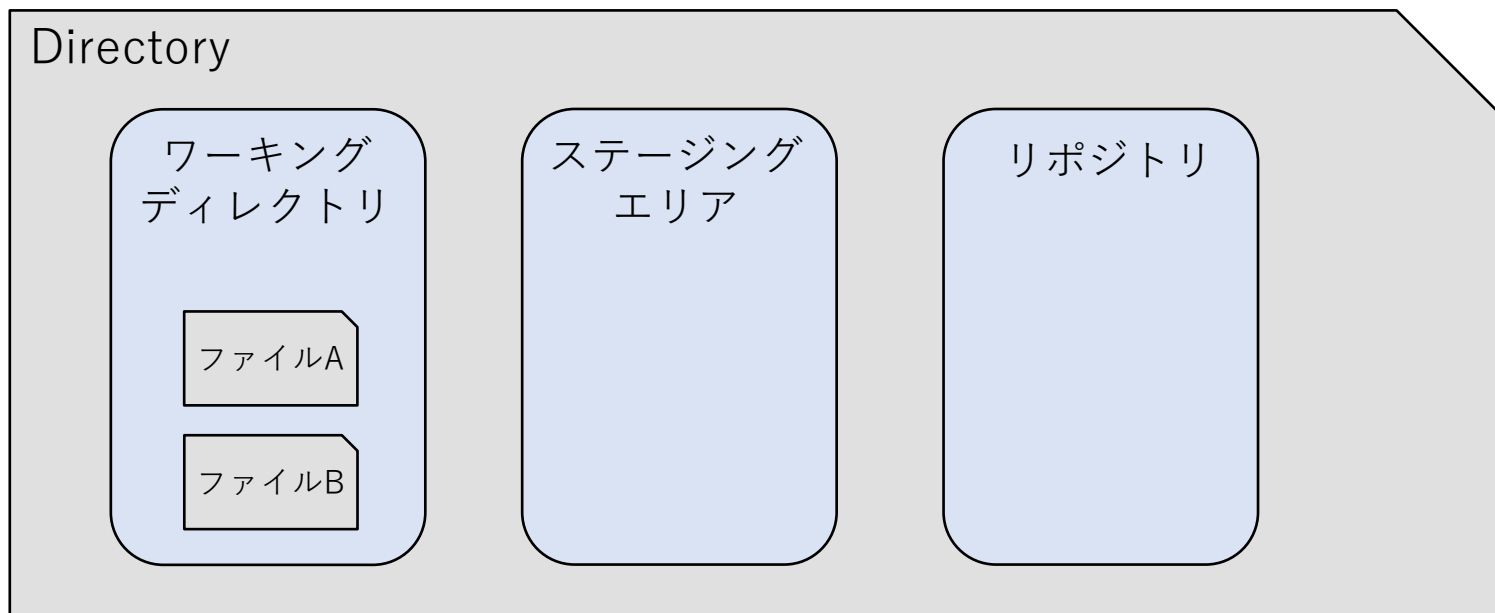
## commitの手順

- ディレクトリをGitの管理下に置く（初期化する）とワーキングディレクトリ、ステージングエリア、リポジトリの3つの場所が作られる。



## commitの手順

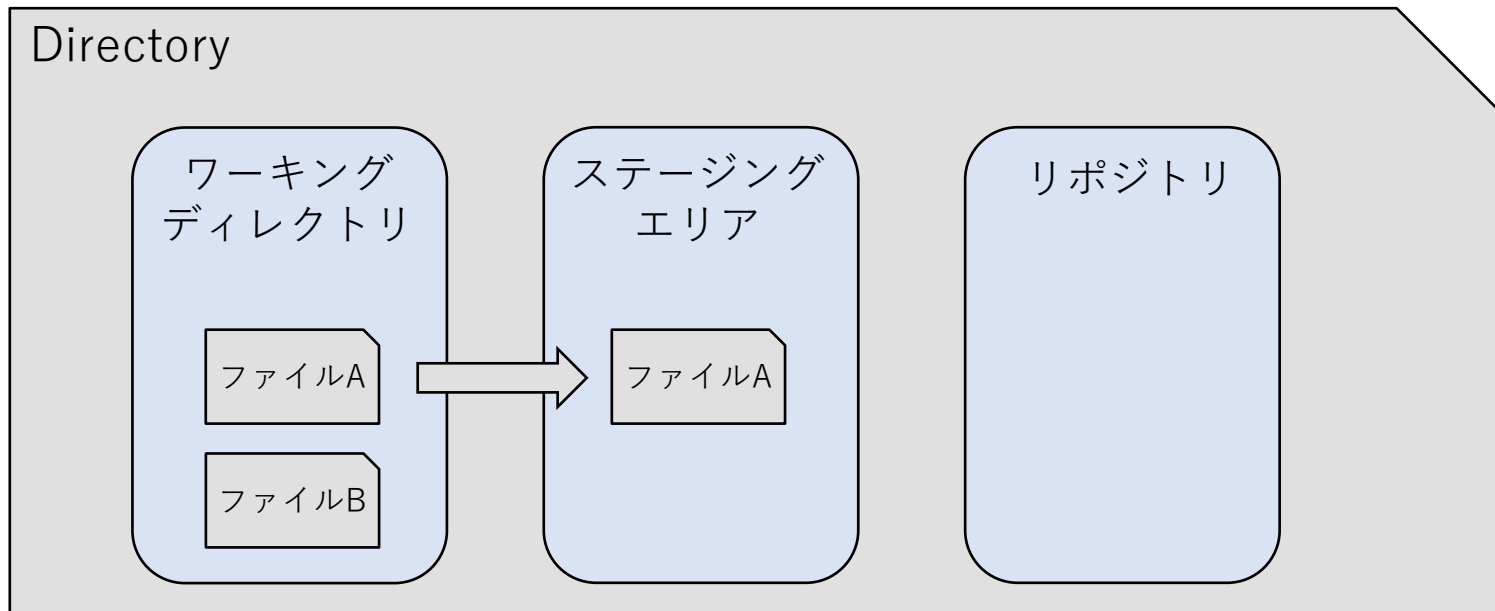
- Gitで管理している、実際の作業を行うディレクトリのことをワーキングディレクトリという。



# Gitの概要

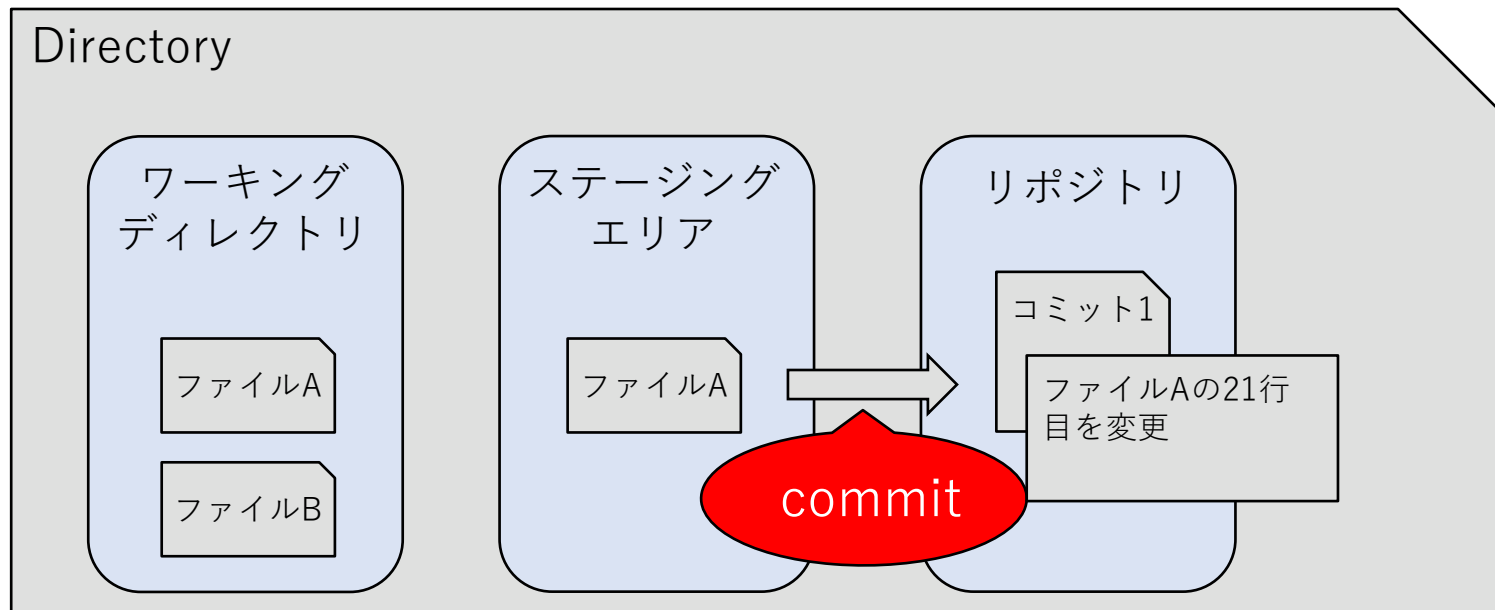
## commitの手順

- 変更履歴として保存するファイルを選択し、置いておく場所を **ステージングエリア** という。
- コミットの前に、ワーキングディレクトリのファイルの中から履歴として保存したいファイルを選びステージングする。



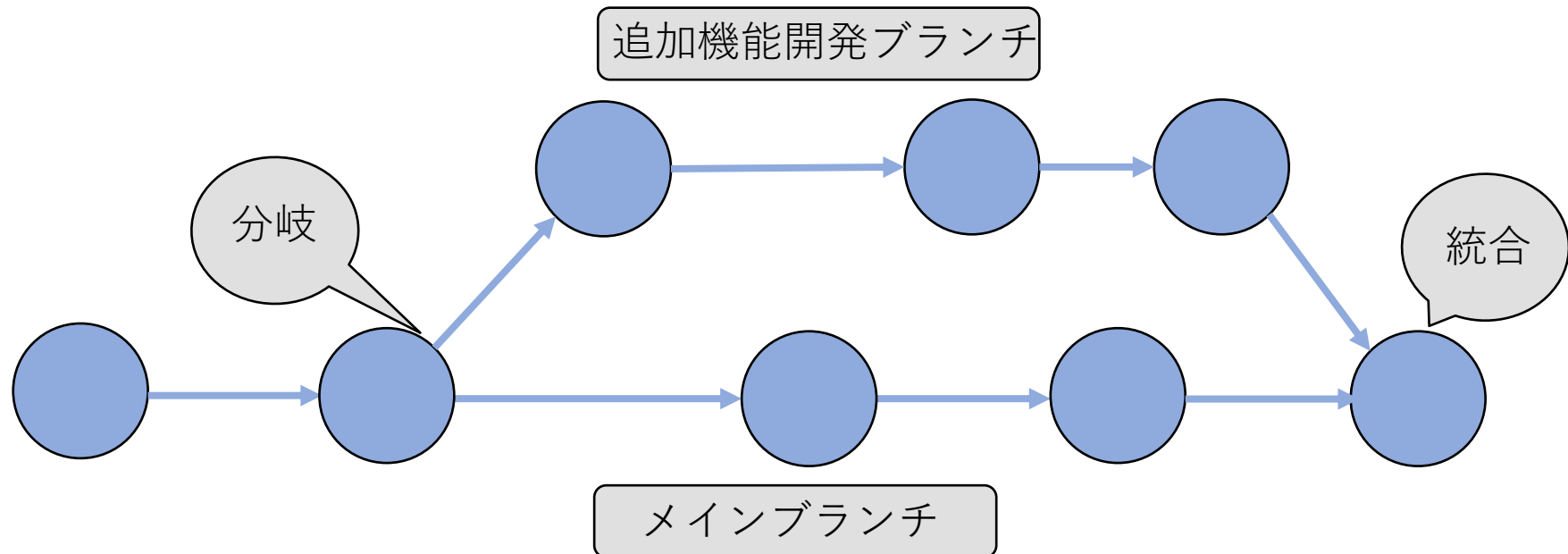
## commitの手順

- commitをすると、ステージングエリアに置かれているファイルの変更履歴が、リポジトリに保存される。
- ステージングエリアがあることで、ワーキングディレクトリの中の保存しておきたいファイルだけを選んで、commitすることができる。



## branch

- 変更履歴の流れを分岐することで並行での編集を支援する機能をbranchという。
- あるブランチへの変更は、他のbranchに影響しない。
- ブランチ同士の統合（merge）も可能。



## ②Gitチュートリアル

## Gitの利用

- **Windowsの場合**
  - Git for Windows をインストールする
  - WSL (Windows Subsystem for Linux) を利用する
    - Cygwin などのLinuxライクな環境を用意する互換レイヤーをインストールする
- **Mac の場合**
  - Xcode Command Line Tools をインストールする
    - 「git -version」などとした際に未インストールならインストール画面へ  
(近年では、install 済みであることが多い)
- **Linux の場合**
  - 「sudo apt install git-all」など

## Gitの初期設定

```
$ git config --global user.name "JOHO KYODAI"  
$ git config --global user.email "foo@bar.com"
```

- Gitの操作はターミナル上で行う。
- `git config --global` コマンドによりユーザー名とメールアドレスを登録する。
- 登録したユーザーがcommitの際に、記録される
- 私の例：

```
$ git config list  
user.email=m_nagano@i.kyoto-u.ac.jp  
user.name=nagano28
```

## Gitリポジトリを初期化する

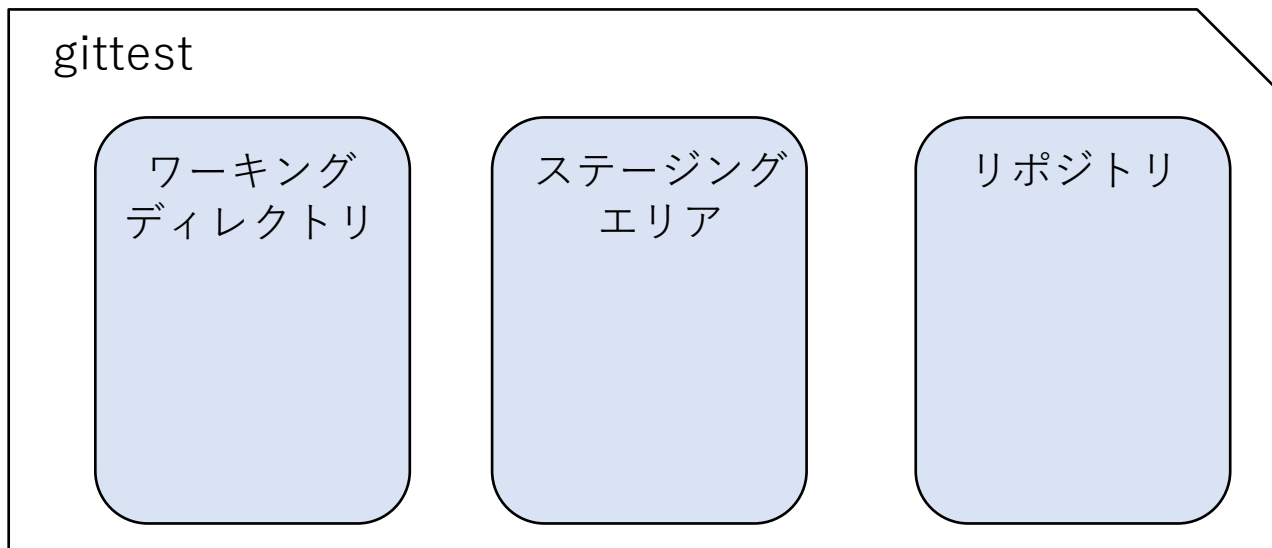
```
git init
```

現在のディレクトリをGitリポジトリとして初期化する。

## Gitリポジトリを初期化する

```
$ mkdir gittest ←フォルダを作成
$ cd gittest ←フォルダへ移動
$ git init ←リポジトリの初期化
Initialized empty Git repository in <ディレクトリ>
/gittest/.git
```

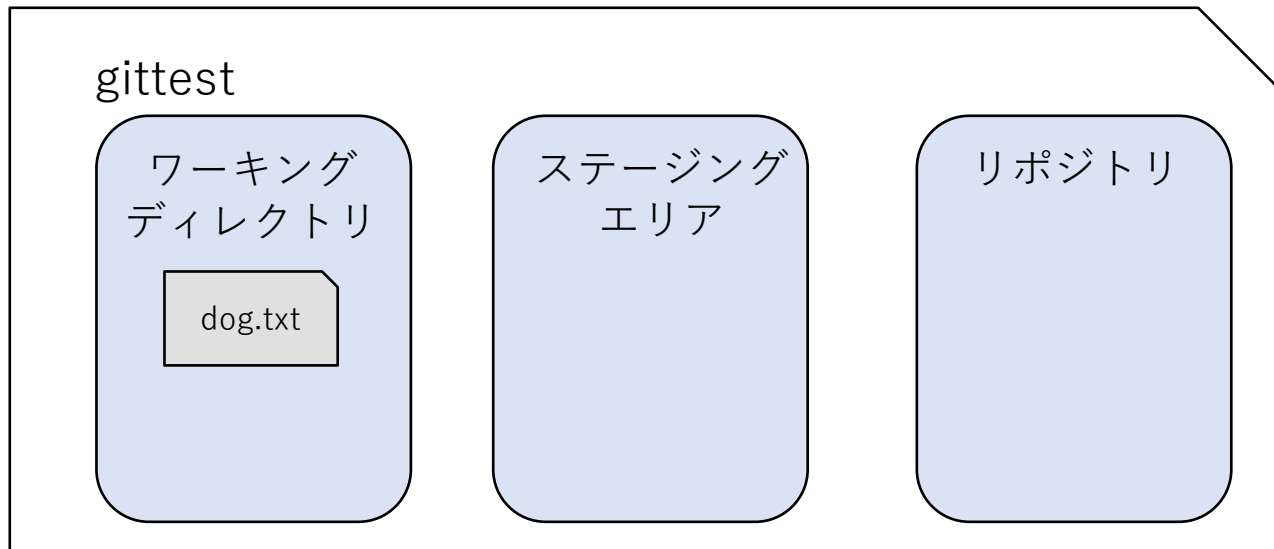
- gittestディレクトリがGitリポジトリとして認識される。
- ディレクトリには.gitディレクトリが作成されている。



## ディレクトリ内にテキストファイルを作成する。

```
$ echo "bow wow" >> dog.txt
```

(Linuxにおいて、テキスト作成・編集の一般的なコマンド)  
※ファイルの中身の確認がしたければ、`$ cat dog.txt`



ワーキングディレクトリにdog.txtが追加される。

## 現在のディレクトリ内の状態を確認する

```
git status
```

ディレクトリ内の状態を確認する。

## 現在のディレクトリ内の状態を確認する

```
$ git status
On branch main

Initial commit

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

    dog.txt

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

- 新しく作成したファイル (dog.txt) がUntracked files:の一覧に表示される。
- Untracked filesには一度もステージングしたことがないファイルが表示される

指定したファイルをステージングエリアへ追加する

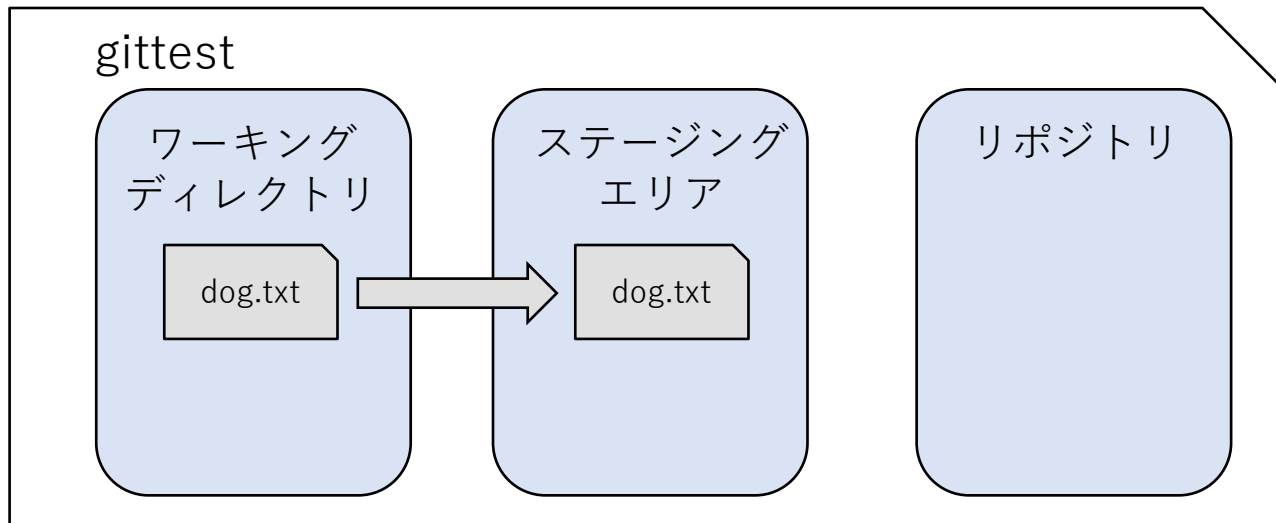
```
git add <ファイル名>
```

<ファイル名>をステージングエリアに追加する。

## 指定したファイルをステージングエリアへ追加する

```
$ git add dog.txt
```

- dog.txtがステージングエリアに追加される。



## ステージングエリア追加後の状態を確認する

```
$ git status
...
Changes to be committed:
(Use "git rm -cached <file>..." to unstage)

    new file:   dog.txt
```

- ステージング後に、再びgit statusコマンドにより状態を確認する。
- Changes to be committed:の一覧にdog.txtがnew fileとして表示されている。ステージングエリアに新たに追加されたことを示している。

ステージングされたファイルをcommitする。

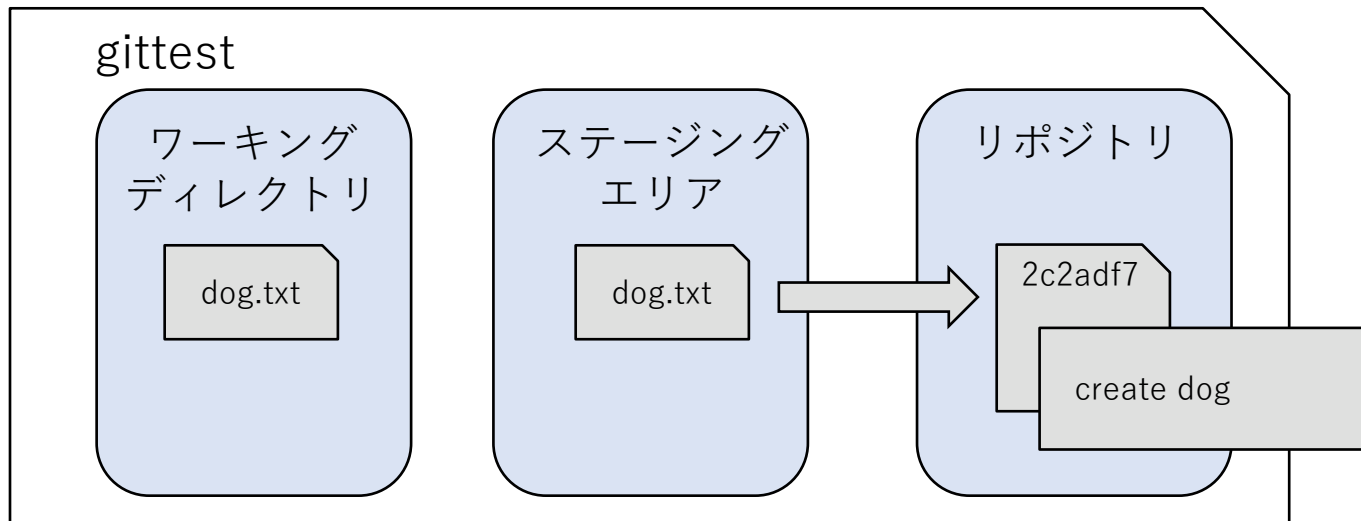
```
git commit -m <コメント>
```

- ステージングされているファイルをcommitする。
- -mオプションをつけると1行のコメントをつけることができる。
  - どのような処理をしたのか、  
分かり易い見出しがオススメ。

## ステージングされたファイルをcommitする。

```
$ git commit -m "create dog"  
[master (root-commit) 2c2adf7] create dog  
1 file changed, 1 insertion(+)
```

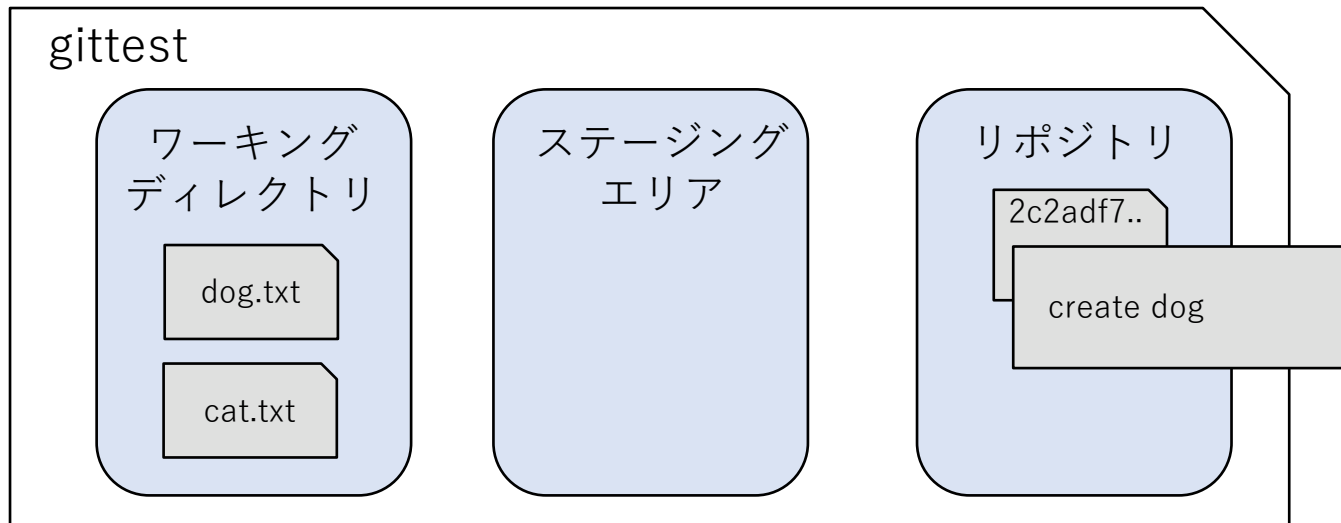
- create dogというコメントをつけてcommitする。
- ステージングされていたdog.txtがcommitされる。



## dog.txtの変更と、新規ファイルの追加。

```
$ echo "mew" >> cat.txt  
$ echo "wan wan" >> dog.txt
```

- dog.txtを編集し、新たにcat.txtを追加する。

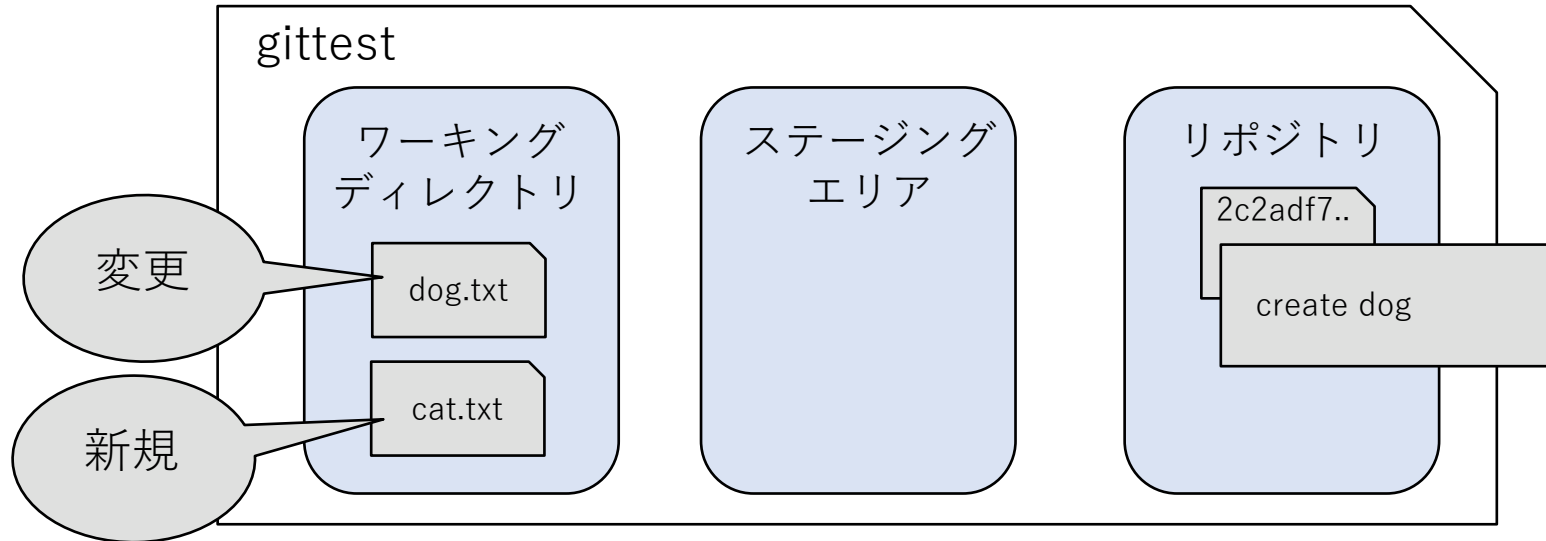


## dog.txtの変更と、新規ファイルの追加。

```
$ git status
On branch main
Changes not staged for commit:
...
    modified: dog.txt
Untracked files:
...
    cat.txt
```

- Changes not staged for commitには過去にステージングしたことがあり、かつ最新のcommitから変更されているファイルが表示される。
- Untracked filesにはcat.txtが表示される。

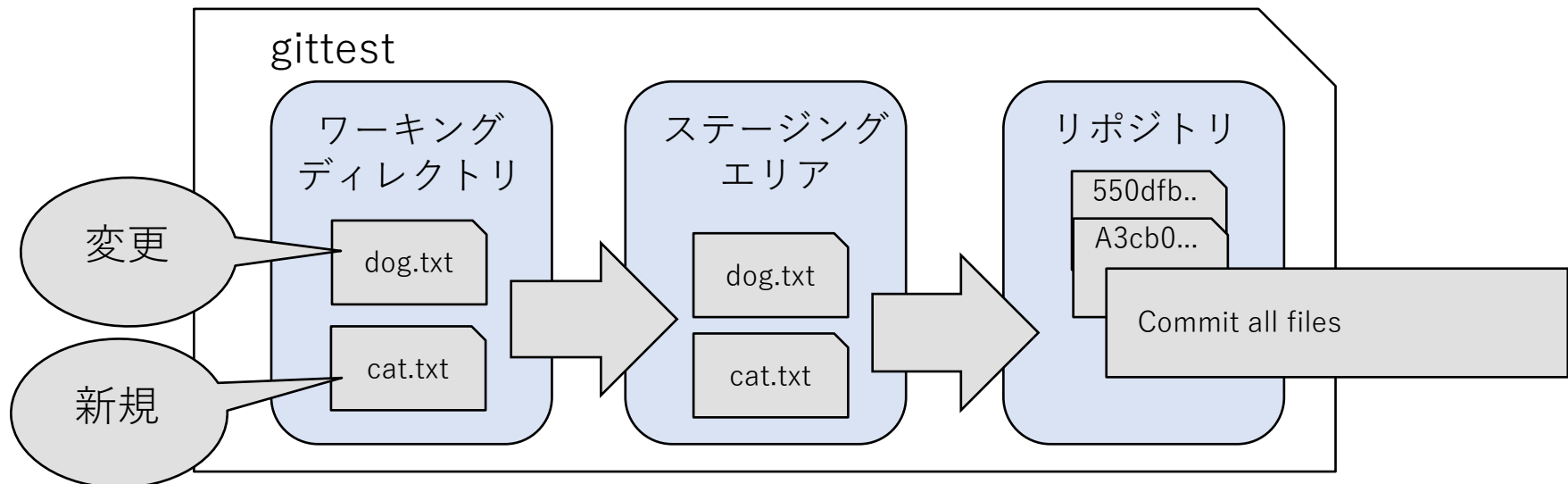
## dog.txtの変更と、新規ファイルの追加。



## 全ての変更をコミット。

```
$ git add *.txt  
$ git commit -m "commit all files"
```

- dog.txtの変更とcat.txtの追加が一つの変更としてコミットされる。
  - git addのファイル指定ではワイルドカードが使える。
  - "."を指定すると変更があった全てのファイルをステージングする。
- [+α]
- git add --allなどで全てのファイルをステージング
  - git reset xxxで、commitやaddを取り消し可能 (xxxは用途によって変更)



## commitの履歴を確認する

```
git log
```

リポジトリの保存されているcommitの履歴を確認する。

## commitの履歴を確認する

```
$ git log
commit 550dfb1f4de340c3abe04549f52c4a6598ad4cbf
Author: Kazunari Kato<kato@example.com>
Date: Fri Apr 27 14:01:23 2018 +0900

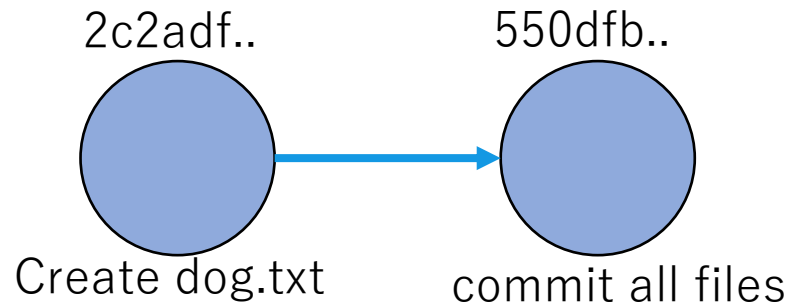
    commit all files

commit 2c2adf7291e16fd4301a625c8ac5589edc703f83
Author: Kazunari Kato<kato@example.com>
Date: Fri Apr 27 13:58:26 2018 +0900

    create dog.txt
```

- commit履歴が新しいものから順番に表示される。
- 1行目の文字列はcommit IDといって、commitを識別するためのもの。

## commitの履歴を確認する



以前のcommitの状態に戻す

```
git reset --hard <commit id>
```

ファイルを<commit id>で指定したcommitの状態に戻す。

## 以前のcommitの状態に戻す

```
$ echo "tweet" >> bird.txt
$ echo "meow meow" >> cat.txt
$ git status
// bird.txtの追加とcat.txtが変更されているメッセージが出る。
$ git reset --hard //コミットIDを省略すると最新のコミットに戻る
$ git status
// 変更が何もないというメッセージがでる。(bird.txtに関して以外)
```

- 新たにbird.txtを追加し、cat.txtを変更する。
- git reset --hardコマンドで、ファイルを最新のcommitの状態に戻す。
  - どちらかで、変更が何もないというメッセージがでる。
    - .gitignoreでbird.txtを追跡対象から外す。
    - git clean -fdコマンドで、未追跡ファイルを取り除く (bird.txt)
- lessコマンドなどで、ファイルを確認すると、元に戻っている。

Gitでトラッキング対象を管理する。

`.gitignore`

リポジトリ内の変更履歴を追跡するファイルを管理する。

## 一つのファイルの追跡を外す

```
$ touch .gitignore
$ echo "bird.txt" >> .gitignore
$ git status
// 変更が何もないというメッセージがでる。
```

- .gitignoreにbird.txtを管理しないように記載
- 変更履歴が不要なファイルや作業対象でないファイルなどを管理する。

branch一覧を表示する。

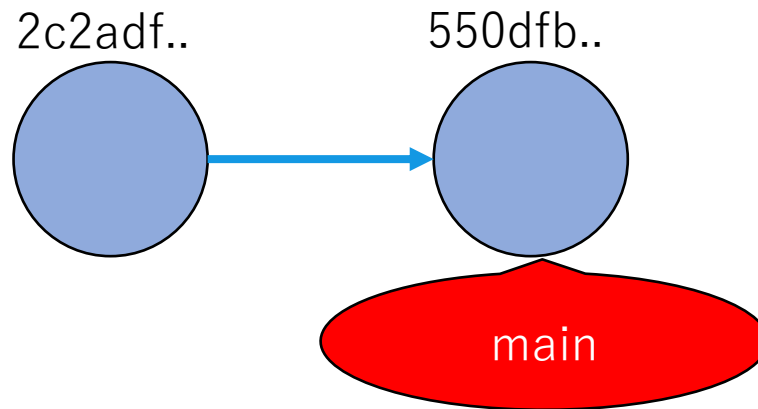
`git branch`

作成されているbranchの一覧を表示する。

## branchを利用する-branch一覧を表示する。

```
$ git branch  
* main
```

- 作成済みのbranchが表示される。
  - main branchは最初から存在している。
    - (デフォルトが master branch の場合も)
- \* は現在選択 (checkout) されているbranchを示している。



## 新しくbranchを作成する

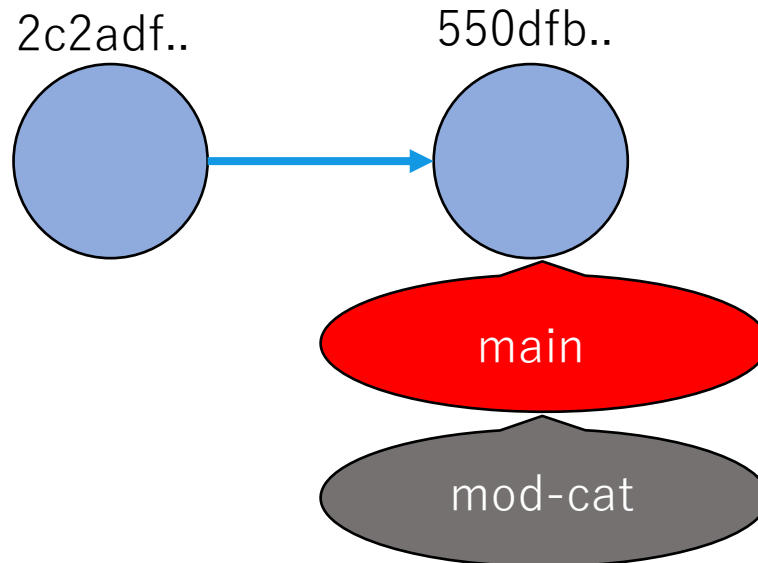
```
git branch <branch名>
```

新しくbranchを作成する。

## 新しくbranchを作成する

```
$ git branch mod-cat  
$ git branch  
* main  
  mod-cat
```

- mod-catというbranchを作成する。
- git branch で確認するとmod-cat branchが作成されているのがわかる。



## branchを選択する

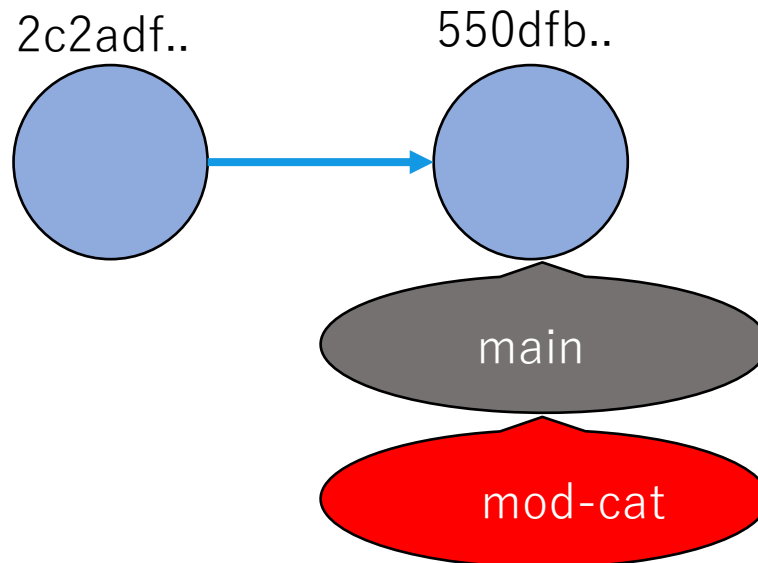
```
git checkout <branch名>
```

<branch名>で指定したブランチを選択する。

## branchを選択する

```
$ git checkout mod-cat  
Switched to branch 'mod-cat'  
$ git branch  
  main  
* mod-cat
```

mod-catブランチを選択して、git branchコマンドでmod-catが選択されていることを確認する。



(参考) branchを作成して選択する

```
git checkout -b <branch名>
```

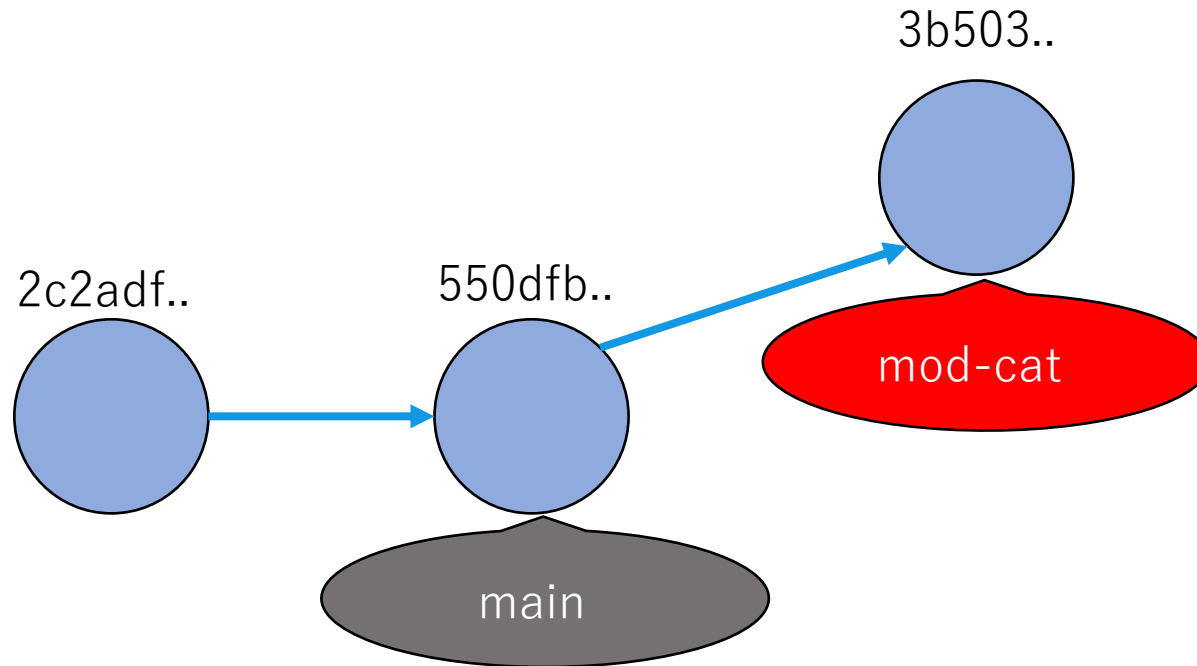
-b コマンドでを使うことで新規ブランチの作成と選択を同時に行うことも出来る。

## mod-catブランチでcommitする

```
$ echo "nya-" >> cat.txt
$ git add cat.txt
$ git commit -m "mod cat"
$ git log --oneline
* 3b503ba mod cat
* 55d0fb1f commit all files
* 2c2adf72 create dog
```

- cat.txtを変更して、commitする。
- git log コマンドに--onelineオプションをつけると履歴を省略形で表示する。
- 3つのcommitがあることを確認できる。

## mod-catブランチでcommitする



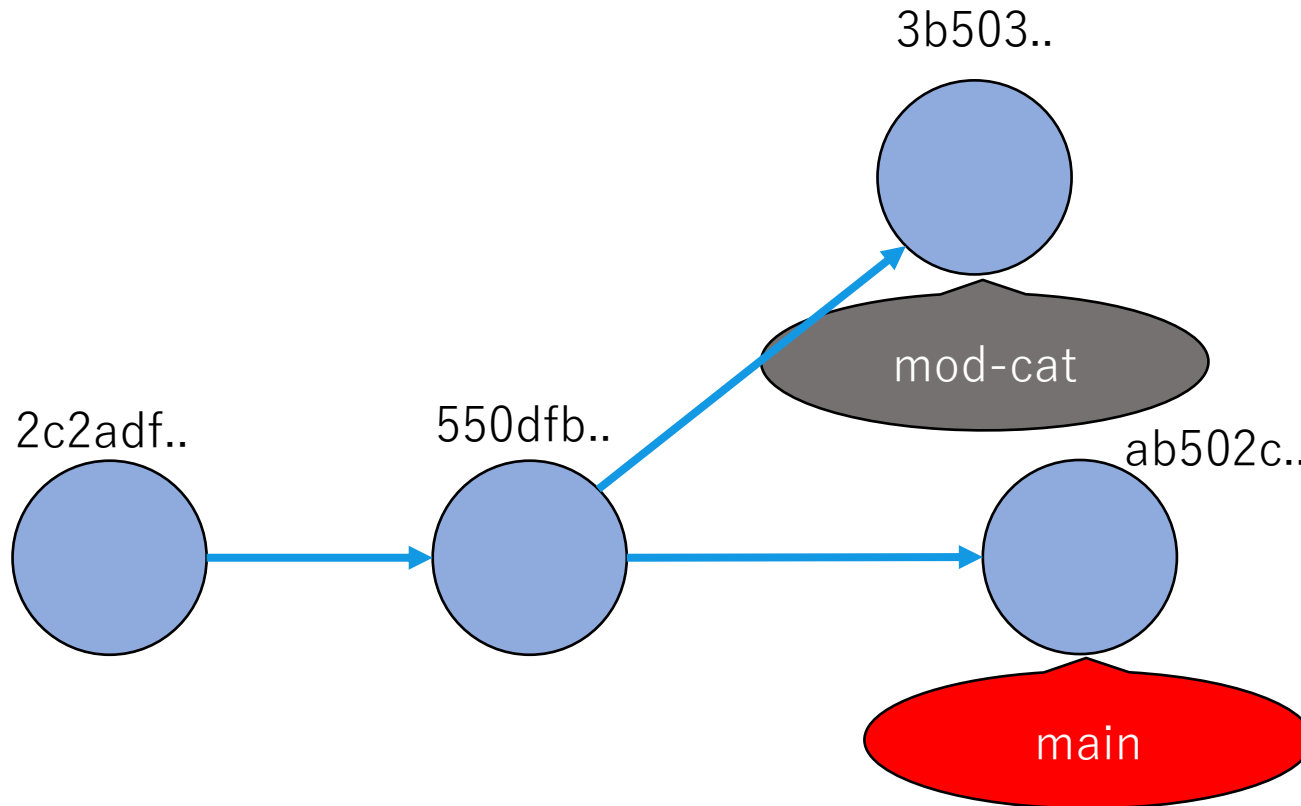
- 作成されたcommitはmod-catブランチで更新される。
- mainブランチは変わらずに550dfb..のcommitを保持している。

## mainブランチを更新する

```
$ git checkout main
$ echo "waon" >> dog.txt
$ git add dog.txt
$ git commit -m "mod dog"
$ git log
* e6fa391 mod dog
* 55d0fb1 commit all files
* 2c2adf7 create dog
```

- mainブランチを選択してdog.txtを変更しcommitする。
- commit履歴を確認するとmod-catブランチでのcommitは含まれておらず、mainブランチで行なったcommitのみが表示される。

## mainブランチでcommitを行う



- 新しく作成されたcommitはmainブランチで更新される。
- mod-catブランチで行われた変更はmainブランチには影響されない。

## branchをmergeする

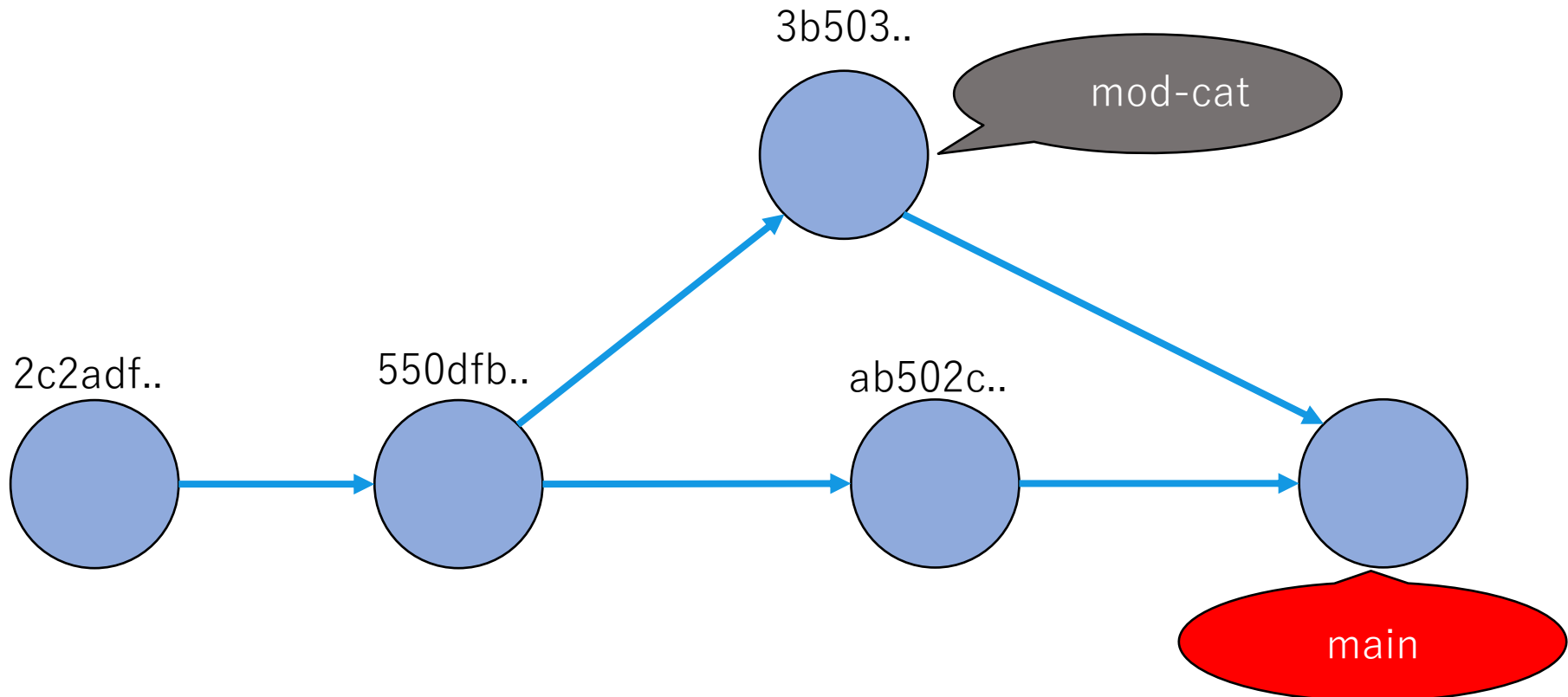
```
git merge <branch 名>
```

現在選択されているbranchに<branch 名>で指定したbranchをmergeする。

## mainブランチにmod-catブランチをmergeする

```
$ git checkout main  
$ git merge mod-cat
```

- mainブランチにcheckoutして、mod-catブランチをmergeする。



## mainブランチにmod-catブランチをmergeする

```
Merge branch 'mod-cat'
```

```
#.....
```

```
//省略
```

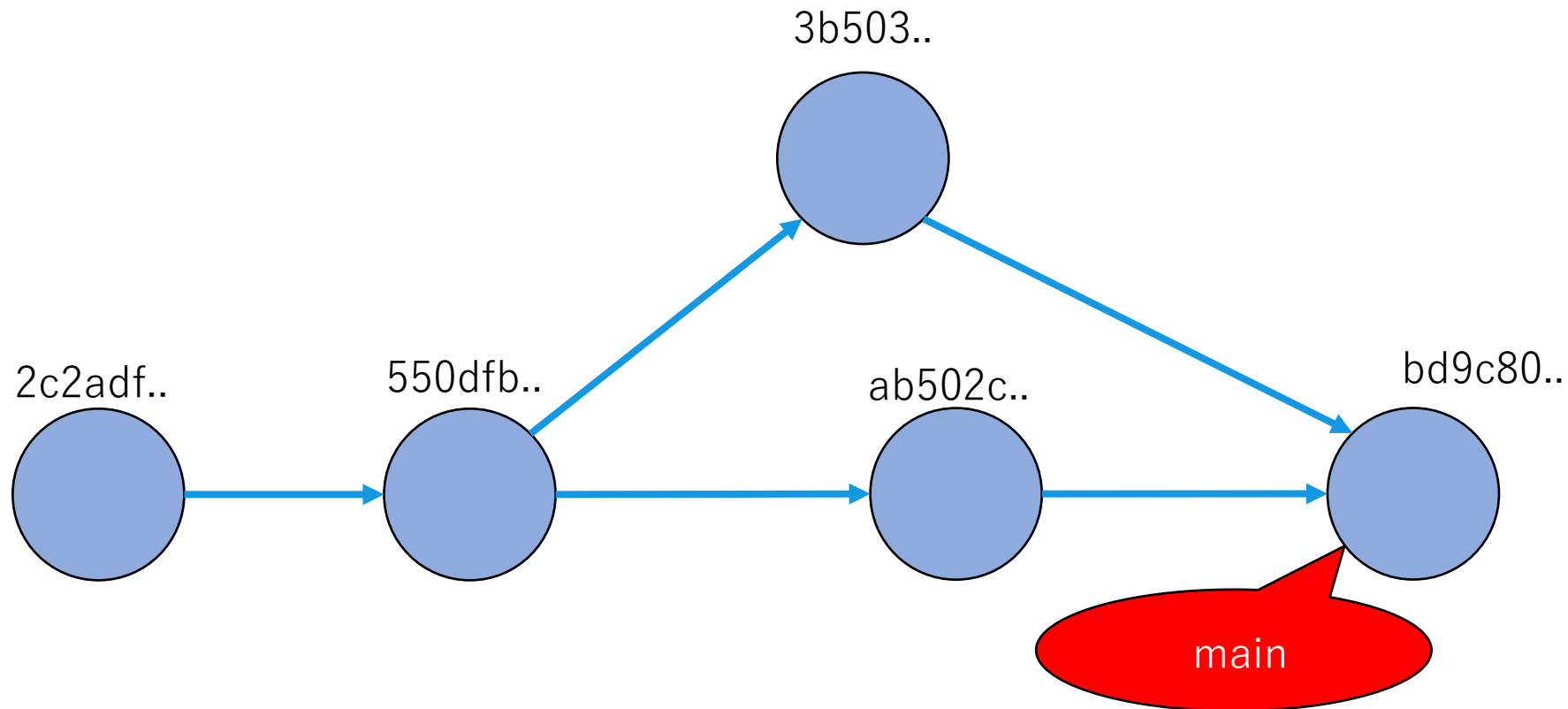
- merge commitを行うためのエディタが立ち上がるので、必要に応じてコメントを修正し、保存・終了する。
  - エディタはvimなので、vimのコマンドを参照

# Gitの使い方

## mainブランチにmod-catブランチをmergeする

```
$ git log --oneline --graph  
//省略
```

- git logコマンドに--graphオプションをつけると、履歴の分岐をグラフィカルに表示する。



## branchを破棄する

```
git branch -d <branch 名>
```

git branchに-dオプションをつけると<branch 名>で指定したbranchを破棄する。

## branchを破棄する

```
$ git branch -d mod-cat  
Delete branch mod-cat (was 23e1793)
```

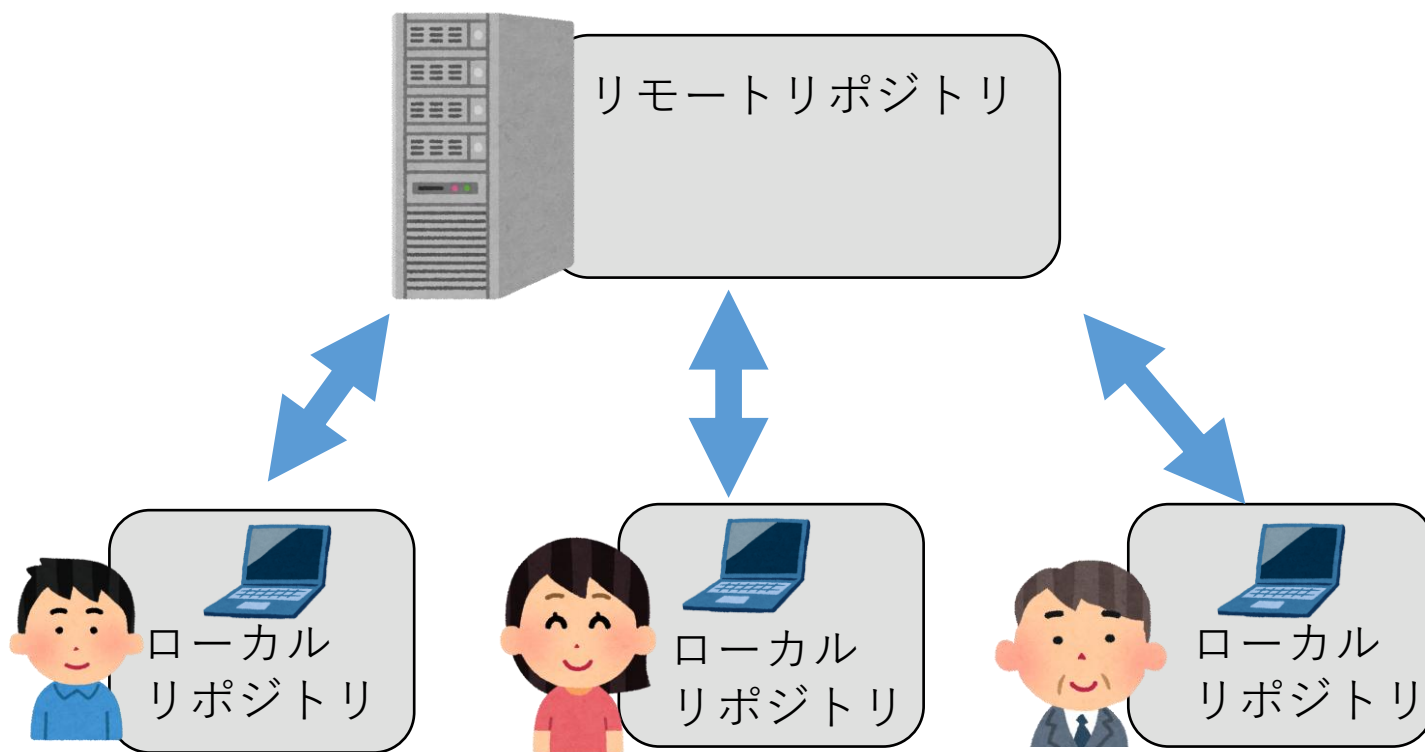
- 機能追加が完了し、不必要となったbranchは消去する。

## ③ リモートリポジトリ・ GitHubの概要

# リモートリポジトリの概要

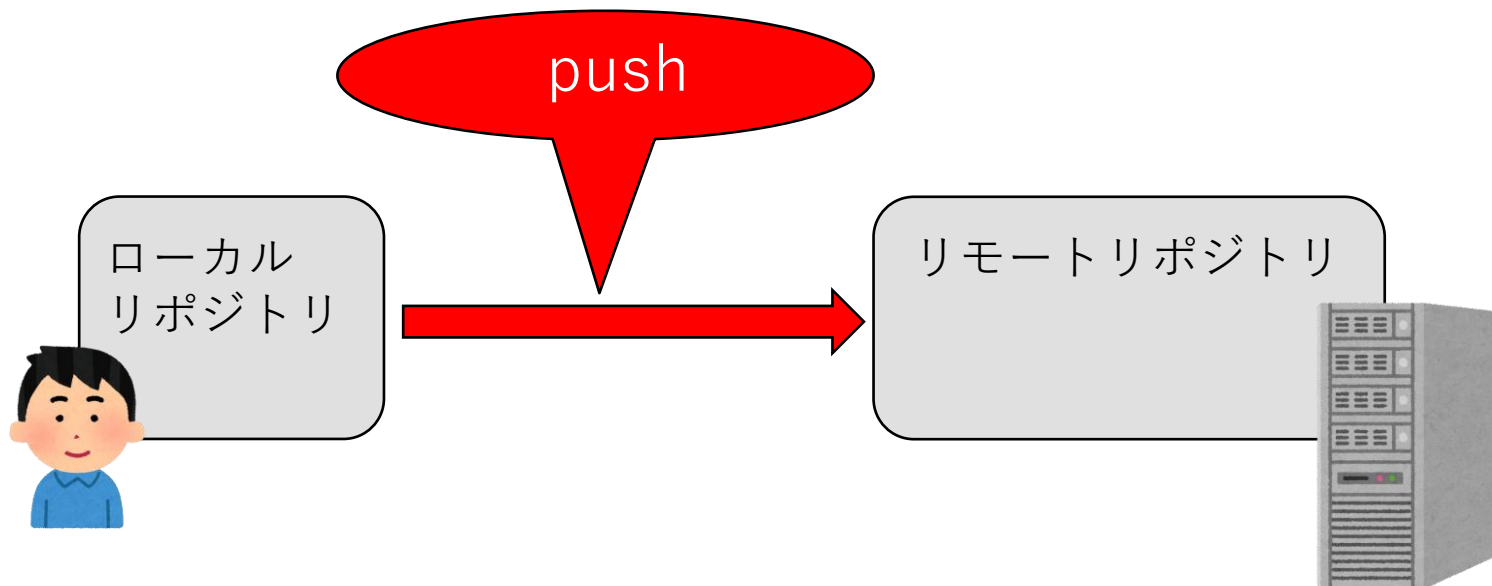
## ローカルリポジトリとリモートリポジトリ

- 自分の端末に存在し、個人で使用するリポジトリをローカルリポジトリ、サーバー上に存在し、複数の人が参照することのできるリポジトリを**リモートリポジトリ**という。



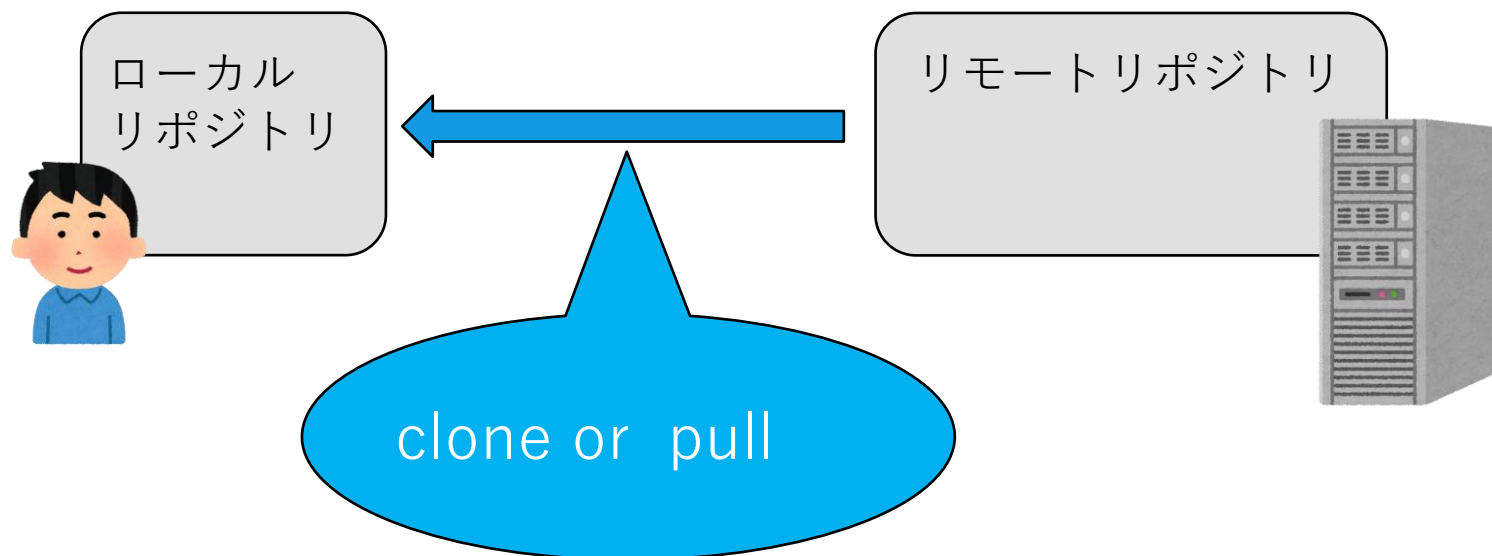
## Push

- リモートリポジトリに自分のローカルリポジトリにある変更履歴をアップロードすることをpushという。



## cloneとpull

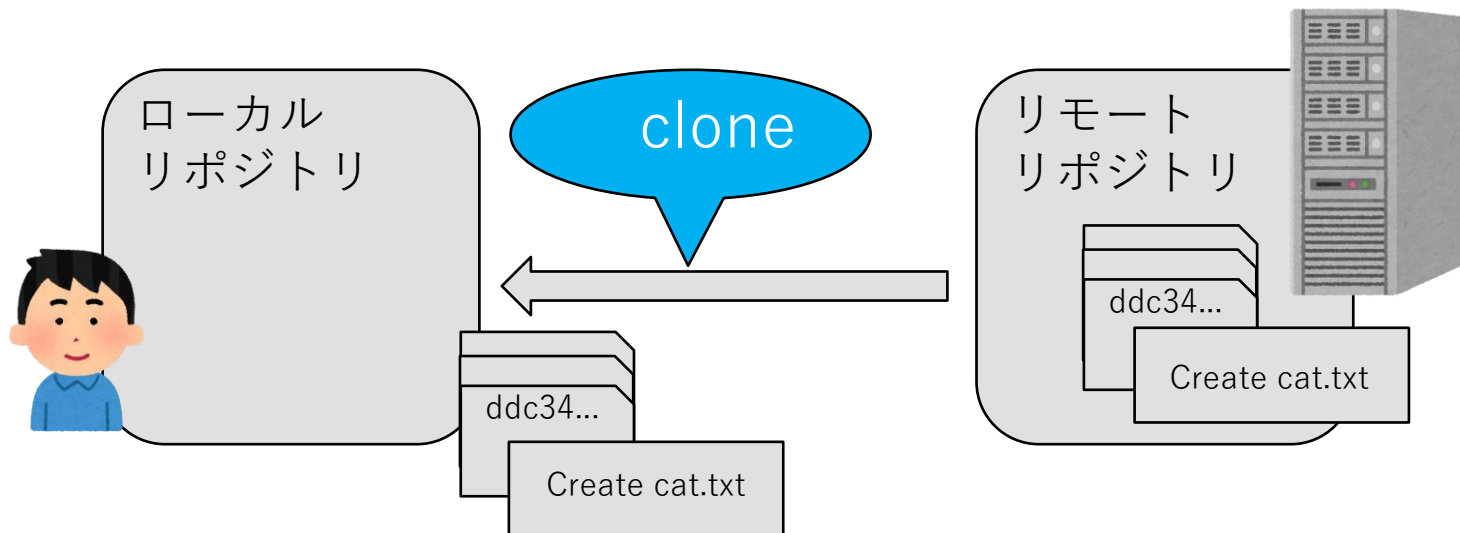
- リモートリポジトリにある変更履歴を自分のローカルリポジトリにダウンロードする方法に、**clone**と**pull**がある。



# リモートリポジトリの概要

## clone

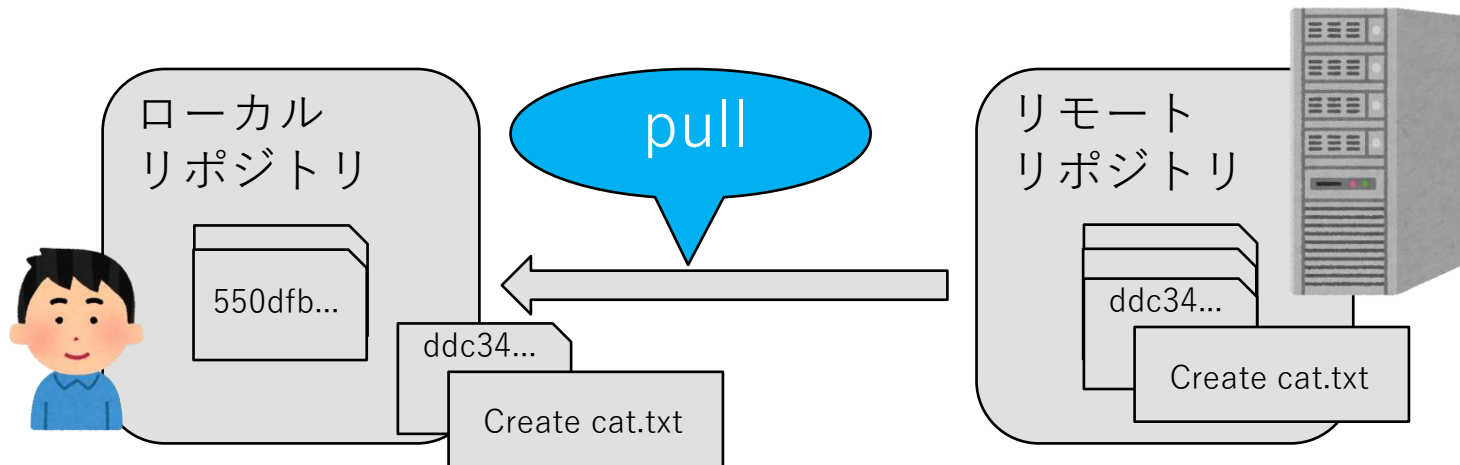
- **clone**は空のローカルリポジトリにリモートリポジトリのコピーを作成する時に使用。



# リモートリポジトリの概要

## pull

- pullはローカルリポジトリをリモートリポジトリの履歴で更新する時に使用。



## GitHub

- GitHubはGitの仕組みを利用して、プログラムのソースコードなどを共有・ホスティングできるサービス。リモートリポジトリとして使用する。

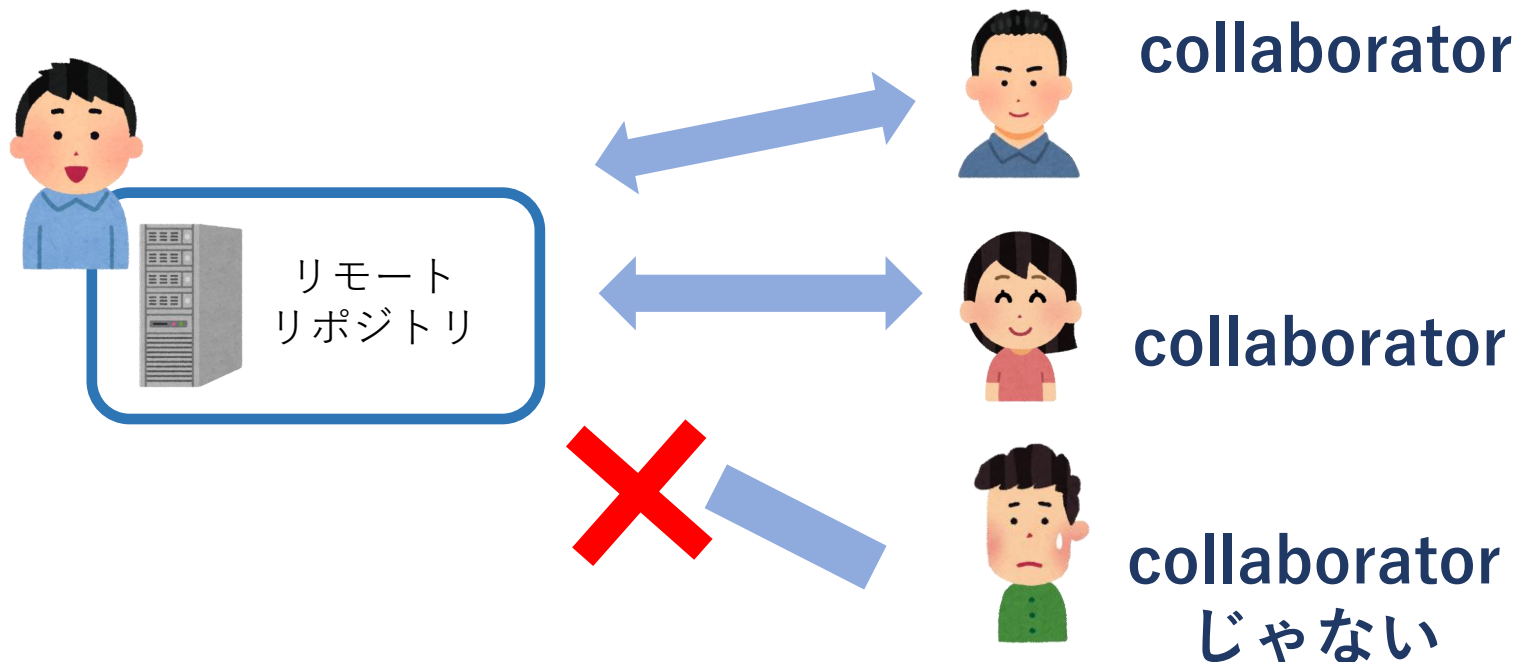
# GitHub

## リポジトリの公開設定

- GitHubでは自分のリポジトリをpublicとprivateいずれかに設定できる。
- Publicでは不特定多数の人がアクセス可能となる。
- Privateでは自分とcollaboratorに設定した人しかアクセスができない。
- **本実験で扱うリポジトリは必ずprivateにすること。**

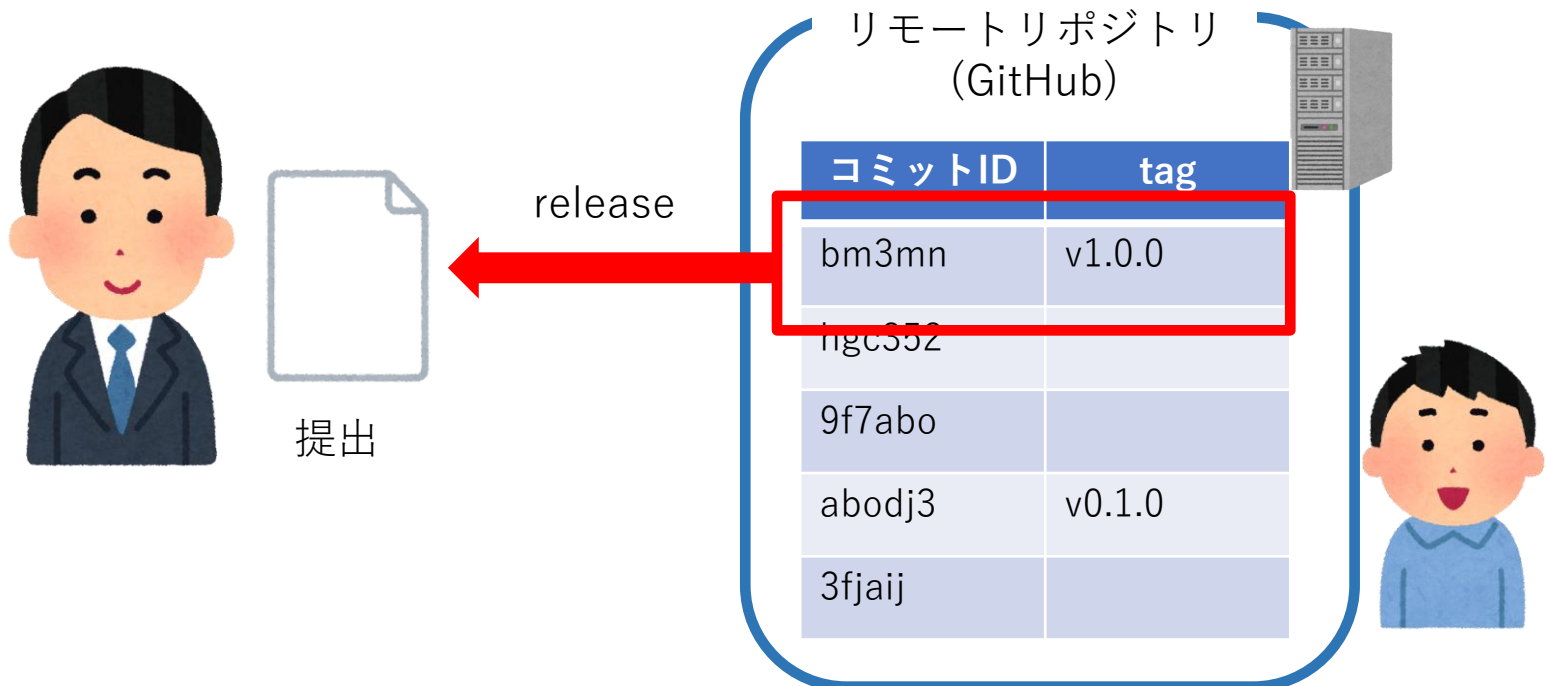
## collaborator

- 自分のprivateリポジトリに他のユーザを**collaborator**として登録すると、リポジトリを共有することができる。



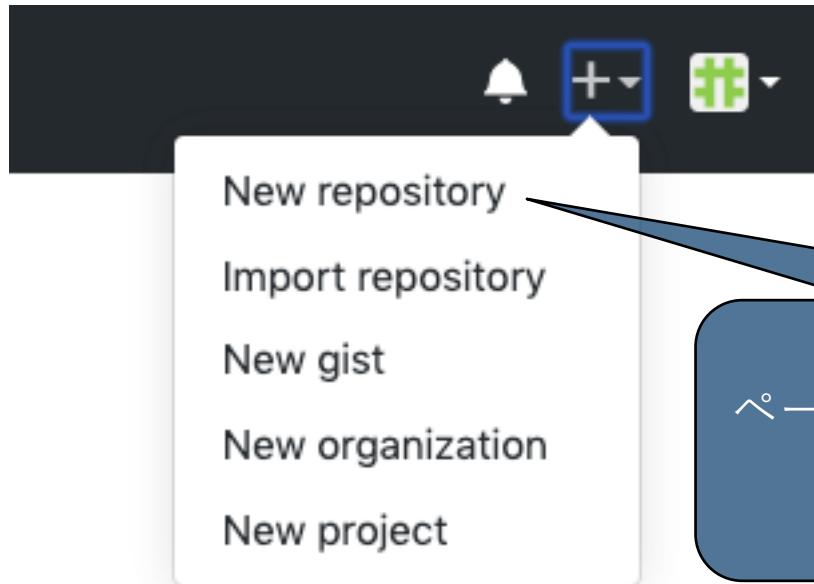
## tagとrelease

- 重要なコミットに分かりやすい名前をつけるための機能を **tag** という。
- tagがつけられたcommitを、他の人が使用できるように提供する機能を **release** という。 **本実験ではreleaseにより課題の提出を行う。**



## ④GitHubチュートリアル

## リモートリポジトリの作成



ページ右上の「+」 → 「New repository」を  
クリック

### 【リポジトリ名に関する注意】


- 導入課題では、2026-intro-<アカウント名>
  - 例) **2026-intro-nagano28**
- プロセッサ設計演習では、2026-simple-team<チーム番号>
  - 例) **2026-simple-team1**

## リモートリポジトリの作成

### Create a new repository

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

Owner

 kazunarikato ▾

Repository name \*

le3-test ✓

任意のリポジトリ  
の名前を入力

Great repository names are short and memorable. Need inspiration? How about [crispy-octo-succotash?](#)

Description (optional)

Privateにチェック  
を入れる

-  **Public**  
Anyone can see this repository. You choose who can commit.
-  **Private**  
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repo

- Initialize this repository with a README**  
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** ▾

Add a license: **None** ▾ ⓘ

「Create repository」  
をクリック

Create repository

## リモートリポジトリの作成

The screenshot shows the GitHub repository creation page for 'kazunarikato/le3-test'. The repository is marked as 'Private'. The page includes a 'Quick setup' section with options for 'Set up in Desktop', 'HTTPS', and 'SSH'. Below this, there are three sections for creating a repository on the command line: '...or create a new repository on the command line', '...or push an existing repository from the command line', and '...or import code from another repository'. A green rounded rectangle highlights the first two sections. Three blue callout boxes provide additional information: one points to the repository name and 'Private' status, another points to the repository URL, and a third points to the command line instructions.

kazunarikato / le3-test Private

Watch 0

Code Issues 0 Pull requests 0 Actions Projects 0 Wiki Security

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

Set up in Desktop or HTTPS SSH `https://github.com/kazunarikato/le3-test.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 "# le3-test" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/kazunarikato/le3-test.git
git push -u origin master
```

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

```
git remote add origin https://github.com/kazunarikato/le3-test.git
git push -u origin master
```

**...or import code from another repository**

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Import code

リポジトリの名前と“Private”になっていることを確認。

リポジトリのアドレス

説明に従って、ターミナルでコマンド。

## リモートリポジトリのアドレス登録

```
git remote add <エイリアス>  
                <URL>
```

<URL>を<エイリアス>という名前でリモートリポジトリとして登録する。

## リモートリポジトリのアドレス登録

kazunarikato / le3-test Private Watch 0 Star 0 Fork 0

<> Code Issues 0 Pull requests 0 Actions Projects 0 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/kazunarikato/le3-test.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 "# le3-test" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/kazunarikato/le3-test.git
git push -u origin master
```

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

```
git remote add origin https://github.com/kazunarikato/le3-test.git
git push -u origin master
```

**...or import code from another repository**

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[Import code](#)

GitHubのリポジトリページに  
リポジトリのURLがある

## リモートリポジトリのアドレス登録

```
$ git remote add origin <リモートリポジトリのURL>
```

- 一般的にエイリアス名には“origin”という名前をつける。
- リモートリポジトリとしてURLをoriginというエイリアス名で設定する。

## リモートリポジトリへのプッシュ

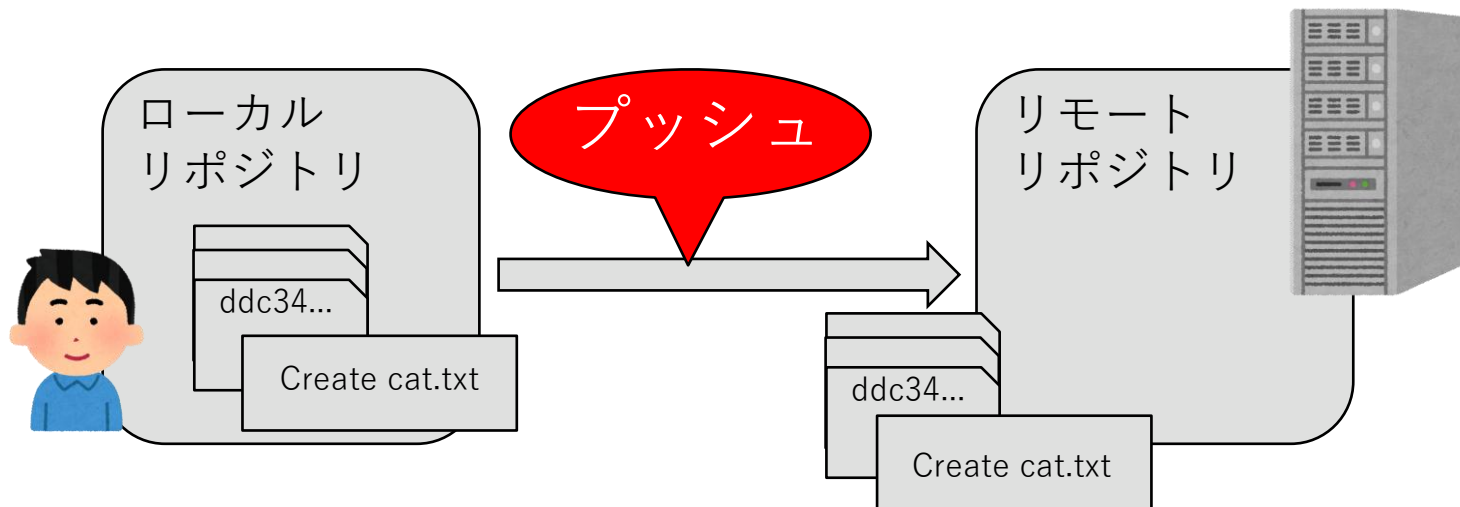
```
git push <エイリアス> <ブランチ>
```

<エイリアス>のリモートリポジトリにローカルの<ブランチ>をpushする。

## リモートリポジトリへのプッシュ

```
$ git push origin main
```

- originのリモートリポジトリにmainブランチをプッシュする。
  - エイリアスをしていなければ、URLを打ち込むことになる。



## リモートリポジトリからのpull

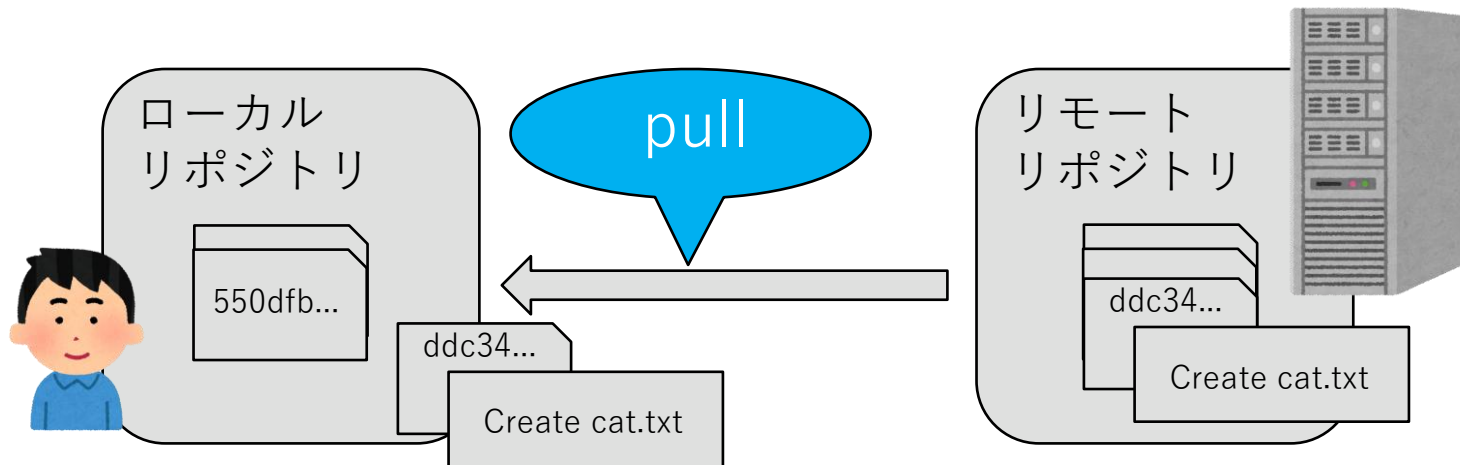
```
git pull <エイリアス> <ブランチ>
```

<エイリアス>のリモートリポジトリの内容でローカルの<ブランチ>を更新する。

## ローカルリポジトリを更新

```
$ git pull origin main
```

- リモートリポジトリの開発がローカルリポジトリよりも進んでいるとき、リモートリポジトリの内容でローカルリポジトリを更新する。



## リモートリポジトリからのclone

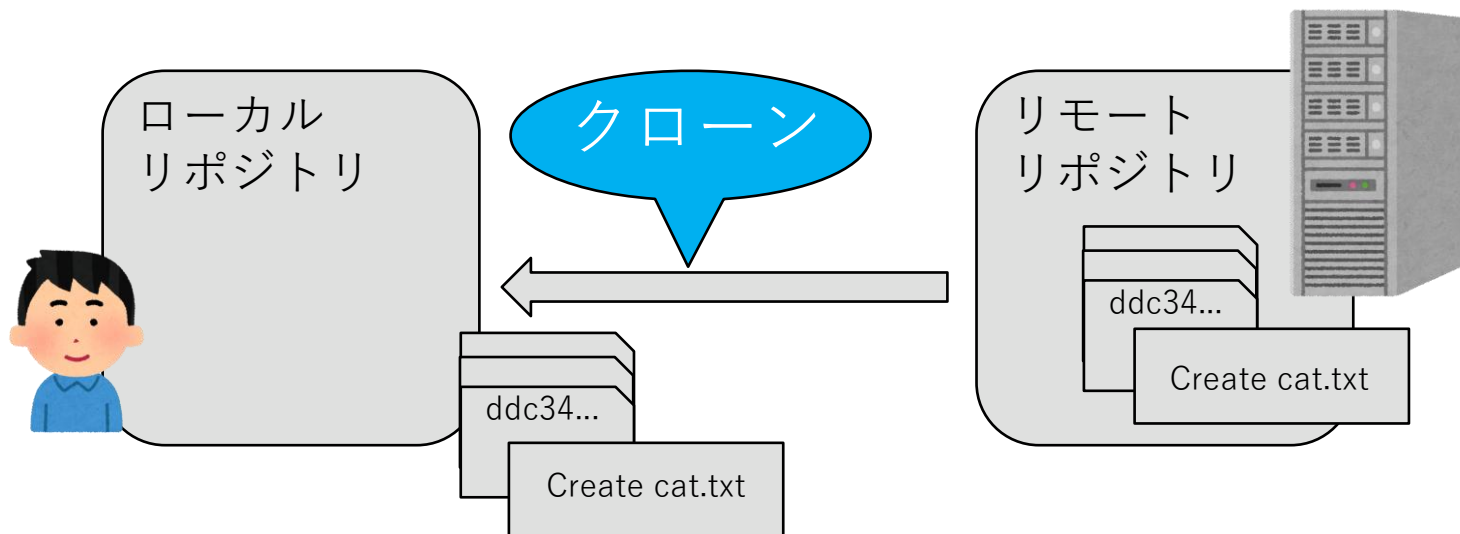
```
git clone <URL>
```

<URL>のリモートリポジトリの内容をローカルにコピーする。

## リモートリポジトリからのclone

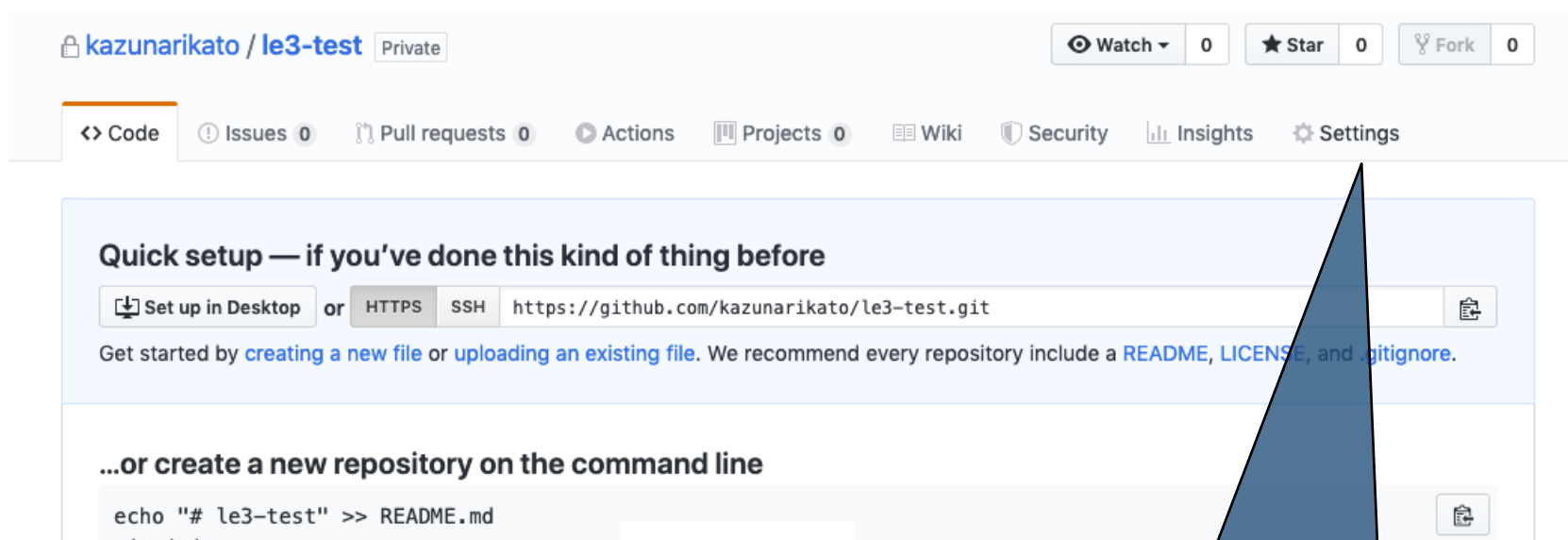
```
$ git clone <URL>
```

- <URL>で指定したリモートリポジトリをcloneする。



# GitHubの使い方

## collaboratorの登録



リポジトリのページで  
“Setting”をクリック

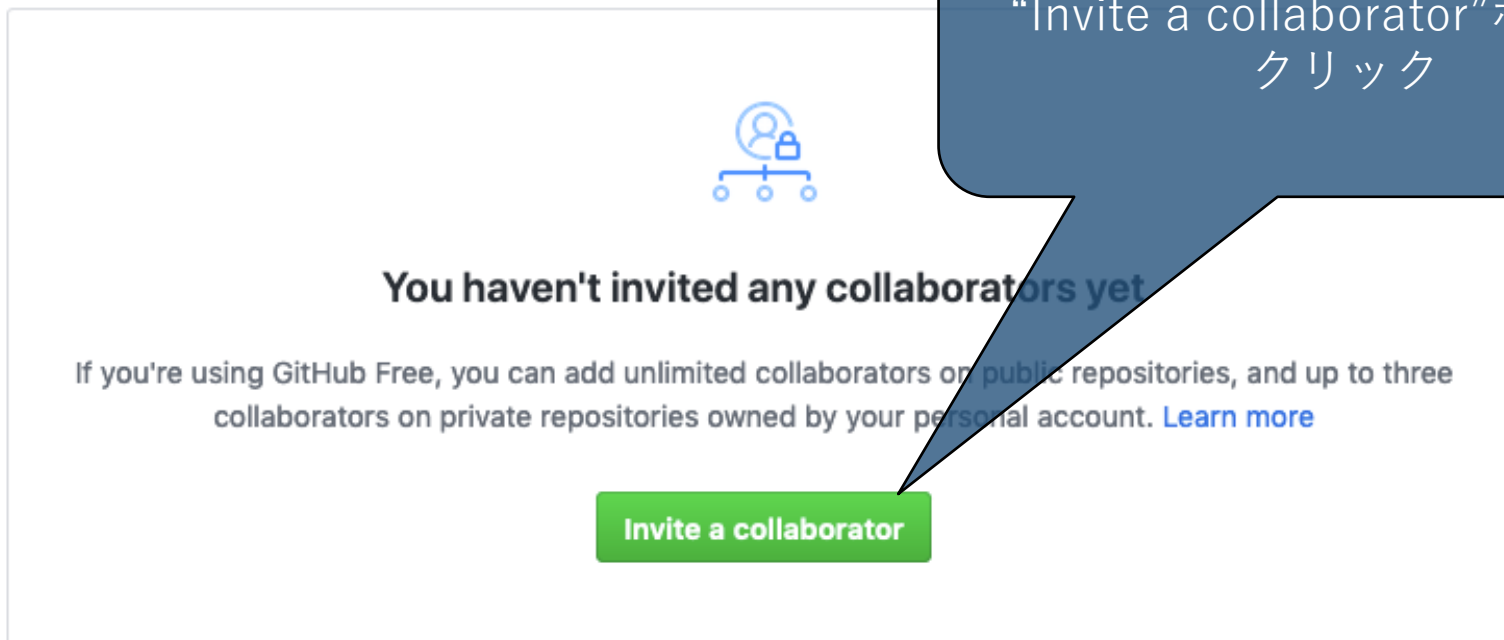
## collaboratorの登録

The screenshot shows the GitHub repository settings page for 'kazunarikato / le3-test'. The repository is marked as 'Private'. At the top, there are navigation links for 'Code', 'Issues 0', 'Pull requests 0', 'Actions', and 'Projects'. On the left, a sidebar menu lists various settings: 'Options', 'Manage access', 'Webhooks', 'Notifications', 'Integrations & services', 'Deploy keys', 'Secrets', and 'Actions'. The 'Manage access' item is highlighted with a blue bar and a pointer. The main content area is titled 'Settings' and includes a 'Repository name' field containing 'le3-test'. Below that, there is a checkbox for 'Template repository' which is currently unchecked. A blue callout box with a white triangle icon points to the 'Manage access' menu item.

画面左のメニューから”Manage access”をクリック

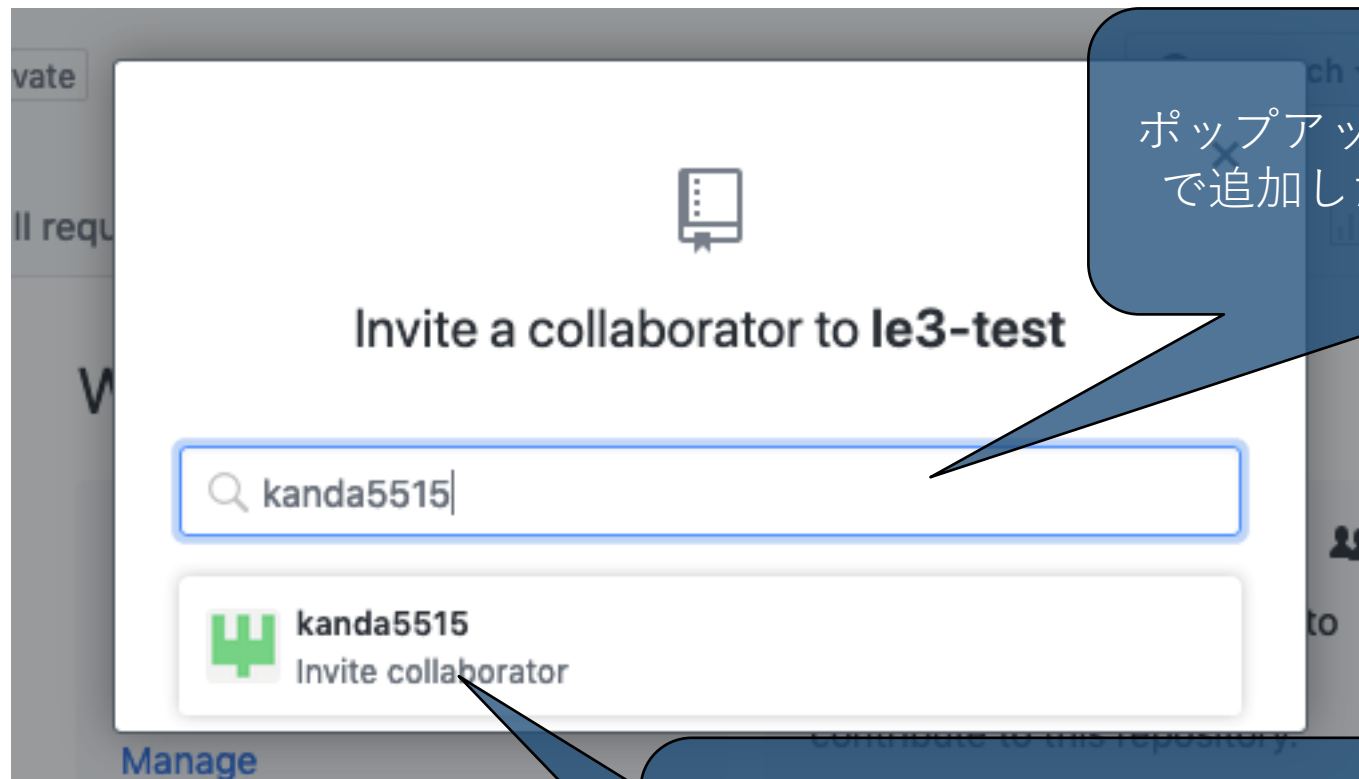
## collaboratorの登録

### Manage access



The screenshot shows the 'Manage access' page on GitHub. At the top, there is a blue icon representing a person with a lock and a tree structure. Below the icon, the text reads: 'You haven't invited any collaborators yet'. Underneath, there is a paragraph: 'If you're using GitHub Free, you can add unlimited collaborators on public repositories, and up to three collaborators on private repositories owned by your personal account. [Learn more](#)'. At the bottom center, there is a green button with the text 'Invite a collaborator'. A blue speech bubble points to this button with the text: '“Invite a collaborator”ボタンをクリック’.

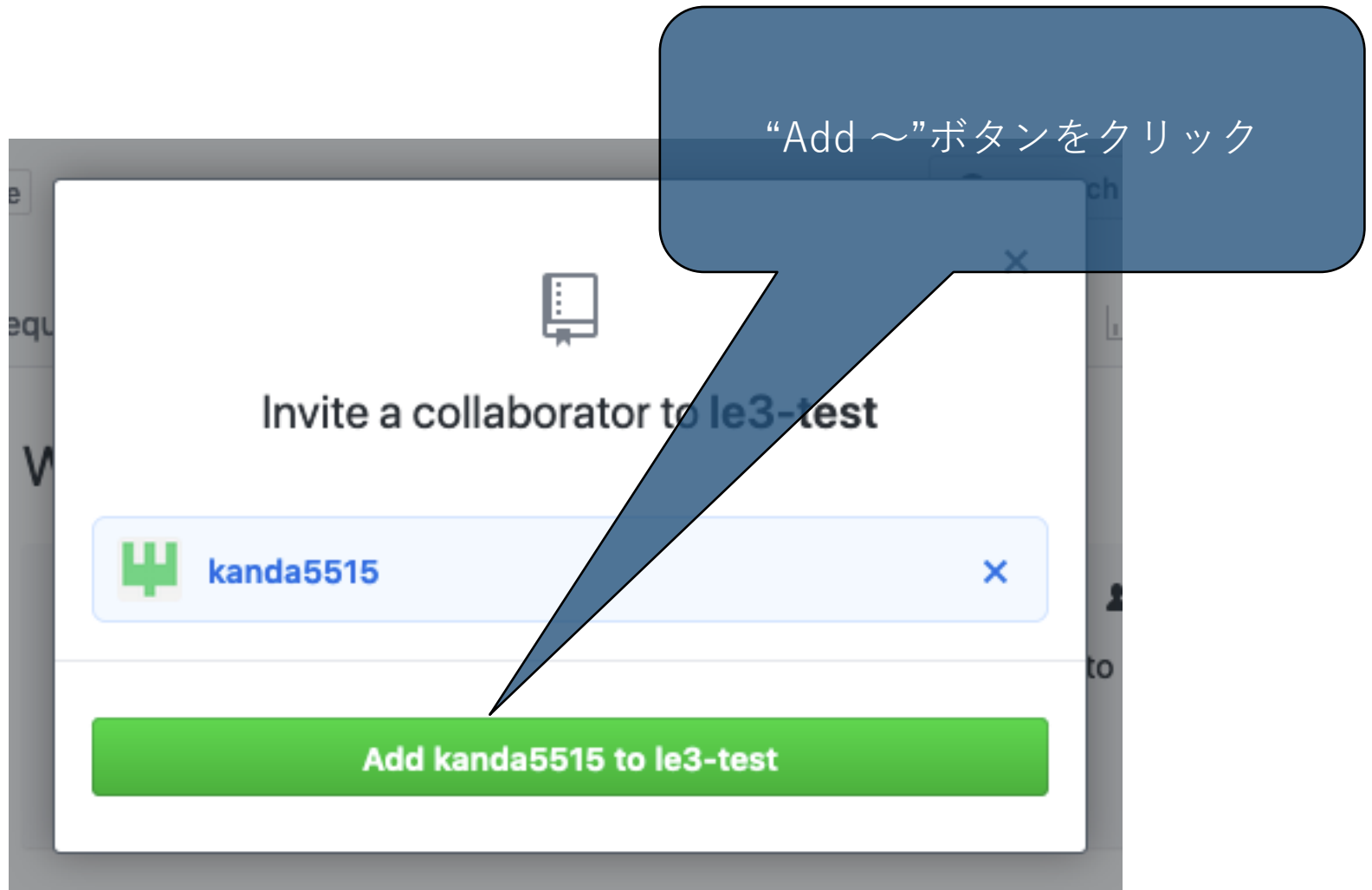
## collaboratorの登録



ポップアップウィンドウの検索窓  
で追加したいユーザー名を検索

検索されたユーザーをクリック

## collaboratorの登録



## collaboratorの登録




ユーザーが登録されていれば成功

### Manage access

Invite a collaborator

Select all Type ▾

Find a collaborator...

<input type="checkbox"/>	 <b>kanda5515</b> Awaiting kanda5515's response	Pending Invite 	
--------------------------	--	--	---

Previous

Next

## コミットにtagをつける

```
git tag -a <タグ名> -m <コメント>
```

最新のコミットに対して、tagをつける。

一般的にタグ名はセマンティックバージョンニングに準じてつけるが、**本実験**では課題ごとに指定されたタグ名をつける。

-a の後にタグ名、-m の後にコメントを記載

過去のコミットにタグをつけたい場合は、<コメント>の後に、コミットIDを書く。

## tagの確認

```
git tag
```

つけたタグの一覧を確認できる。

```
git show <タグ名>
```

指定したタグ名のコミットを確認できる。

## tagのpush

```
git push <エイリアス> --tag
```

git pushに--tagオプションをつけると、タグの情報がリモートリポジトリに反映される。

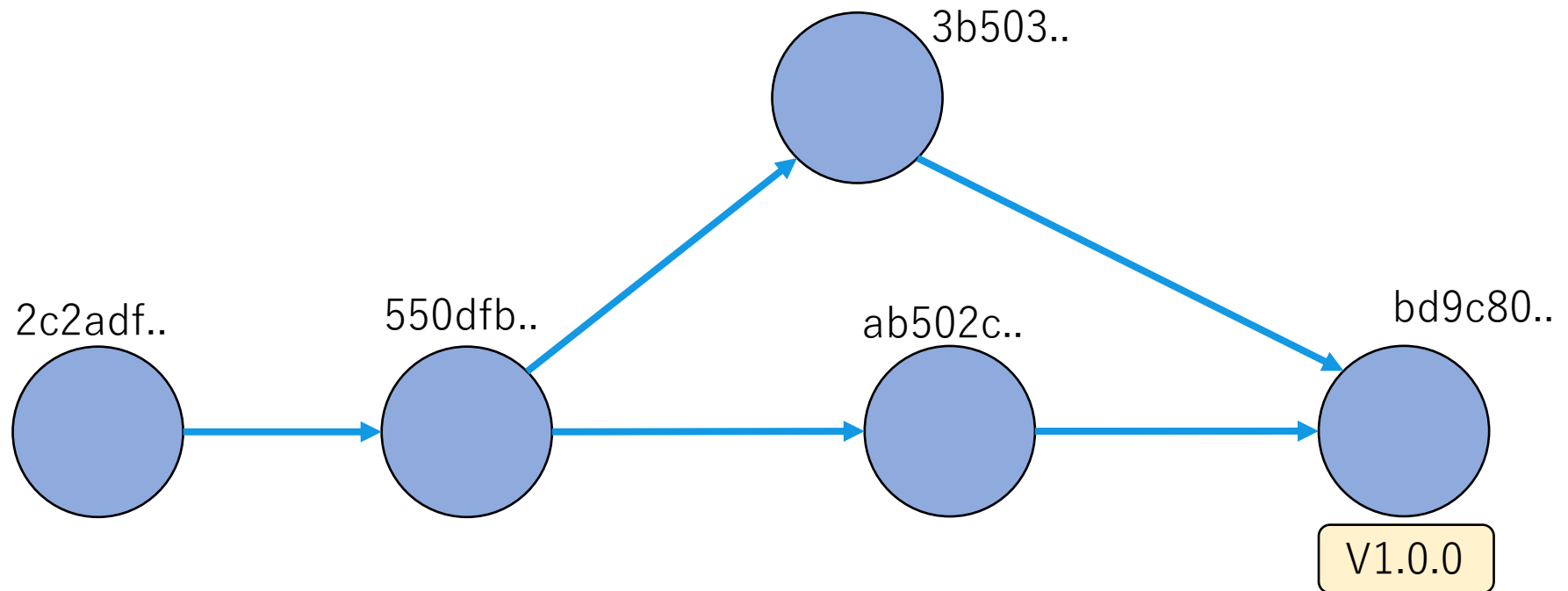
普通にpushしただけでは、リモートリポジトリにtagは反映されない。（逆にこのコマンドだけではブランチへのpushができない）。

必ず“git push <エイリアス> <ブランチ>”でブランチへのpushも完了しておくこと。

## コミットにtagをつける

```
git tag -a v1.0.0 -m 'add v1.0.0'  
git push origin --tag
```

\*必ず “git push <エイリアス> <ブランチ>” でブランチへのpushも完了しておくこと。

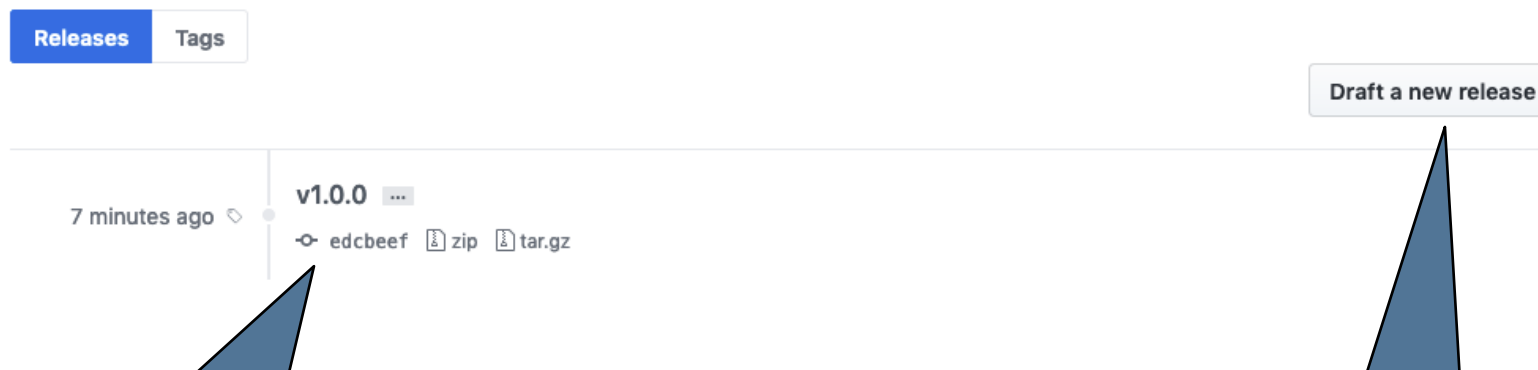


## releaseの作成

The screenshot shows the GitHub interface for a repository named 'smnimo/le3-test'. The repository is private. At the top, there are navigation links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The repository has 1 branch (main) and 1 tag. The commit history shows two commits: 'smnimo second commit' (7 minutes ago) and 'Initial commit' (10 minutes ago). The README file is displayed, containing the text 'le3-test'. On the right side, there are sections for 'About' (No description, website, or topics provided), 'Releases' (1 tags, Create a new release), and 'Packages' (No packages published, Publish your first package).

リポジトリのページで“release”内の  
“Create a new release”をクリック

## Releaseの作成



作成したタグが表示されている。

“Draft a new release”  
ボタンをクリック

## Releaseの作成

The screenshot shows the GitHub 'Create a new release' page. At the top, there are tabs for 'Releases' and 'Tags'. Below them is a 'Tag version' dropdown menu with a search icon and a 'Target: master' button. A note below the dropdown says 'Choose an existing tag, or create a new tag on publish'. Below that is a 'Release title' text input field. There are 'Write' and 'Preview' tabs. A large text area is labeled 'Describe this release'. Below the text area is a dashed box for attaching files with the text 'Attach files by dragging & dropping, selecting or pasting them.' and a separate dashed box for binaries with 'Attach binaries by dropping them here or selecting them.'. At the bottom, there is a checkbox labeled 'This is a pre-release' with the subtext 'We'll point out that this release is identified as non-production ready.'. At the very bottom are two buttons: 'Publish release' (highlighted in green) and 'Save draft'.

Releaseのバージョン番号を入力。  
バージョンはGit tagでつけたタグ  
名から選択。タグ名は課題ごとに  
指定されている。

リリースのタイトルを入力

リリースの説明を入力

“Publish release”をクリック

## Releaseの作成

The screenshot shows a GitHub release page for version v1.0.0. The page includes a 'Releases' tab, a 'Latest release' badge, and a 'Compare' button. The release is titled 'v1.0.0' and is attributed to 'kazunarikato'. The description states 'the first version'. Under the 'Assets' section, there are two files: 'Source code (zip)' and 'Source code (tar.gz)'. Two callout boxes are present: one pointing to the release title and user information, and another pointing to the source code assets.

Releases Tags

Edit release Delete

Latest release

v1.0.0

edcbeef

Compare

v1.0.0

kazunarikato released this now

the first version

Assets 2

Source code (zip)

Source code (tar.gz)

Releaseが作成される

ファイルがダウンロードできる形で提供される

⑤ GitHubに公開鍵を登録

## SSH用の公開鍵の登録

SSHで使うRSAキーの公開鍵をGitHub登録することでターミナルで「`git clone git@github.com:kuis-isle3hw/.....`」

詳しくは

<https://docs.github.com/en/authentication/connecting-to-github-with-ssh> を参考

ここではざっくり過程を見せる。

# SSH用の公開鍵の登録

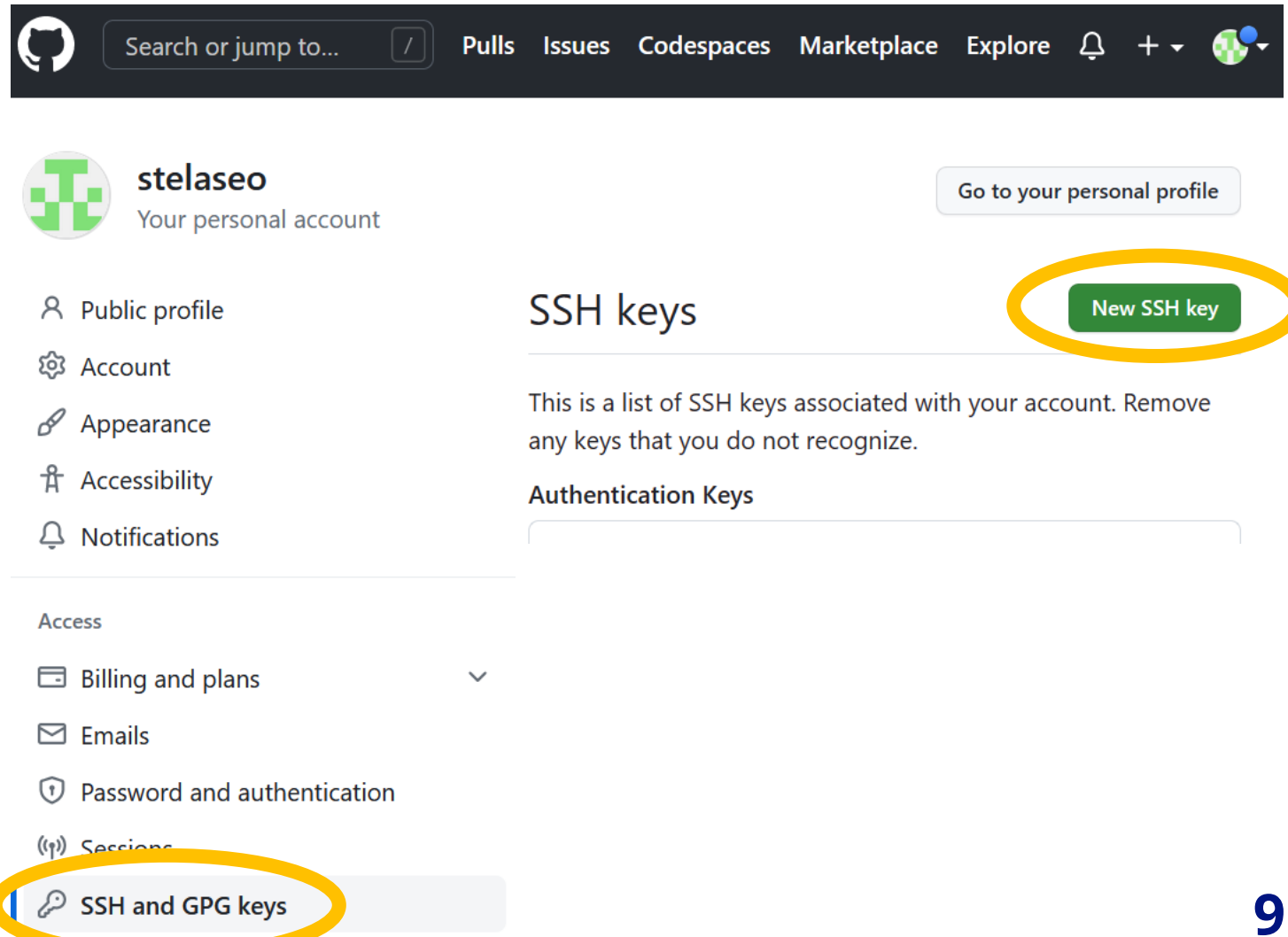
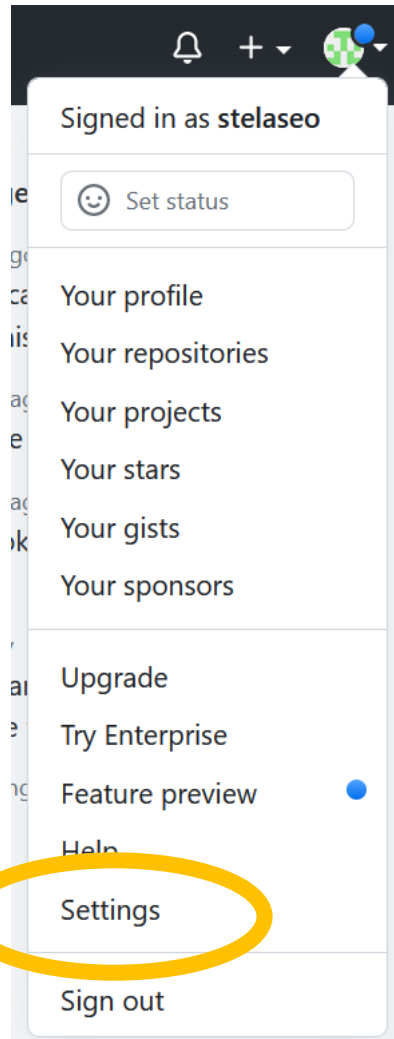
## ターミナルで「ssh-keygen」ツールを使ってキーを作成

```
stela@jupiter:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/stela/.ssh/id_rsa):
Created directory '/home/stela/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/stela/.ssh/id_rsa
Your public key has been saved in /home/stela/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:EtwevEOyzifh82x7F4+/A6MIwX4Fr0NTgDXtdl5Ztnc stela@jupiter
The key's randomart image is:
+---[RSA 3072]-----+
|
|  o+o
| ..o..o   o|
| .+ ==    +.|
| o*oo= . o.E|
| .+oS= o . o|
| +oo+.  =
| *o.o . *
|   *o o o o
|   .+o . .oo
+-----[SHA256]-----+
stela@jupiter:~$
```

# SSH用の公開鍵の登録

## 公開鍵をGitHubに登録する

Settings → SSH and GPG keys → New SSH Key



# SSH用の公開鍵の登録

## 公開鍵をGitHubに登録する

```
stela@jupiter:~$ cat ~/.ssh/id_rsa.pub  
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCoA9Wqej0cpRrdWwI58cJoD6EXy  
d0m70k5nQPX36FGougS1xj2h2dRMvZw24D7hgXJ8enwWZ5FJSpJ1Kf2jE5VrVX4J2  
OzLNPEfVgVtaq3kM5UNguFzVKyYhithMyuKZiWduAVinMSm4xk9Am1RglSHVVHNGG  
P/B03L5vSpuE/AzwLApJ3s0GE16wb/pxtEya7/vU2R1fKugsIT2KqXsi2qB4zUr  
Cf/+EJSIKUh3Z+/0F+HhThP...  
n4CAF54ok9ZpBkDg  
eLtroouPUseGZXM  
9dYxUXqIGAqA7c4  
IM/YAs8RGZAlmSPz
```

id\_rsa.pubの中身を  
コピー & ペースト

SSH key / Add new

Title

Key type

Authentication Key

Key

Begins with 'ssh-rsa', 'ecdsa-sha2-nistp256', 'ecdsa-sha2-nistp384', 'ecdsa-sha2-nistp521', 'ssh-ed25519', 'sk-ecdsa-sha2-nistp256@openssh.com', or 'sk-ssh-ed25519@openssh.com'

Add SSH key

## ターミナルでgit repositoryをclone

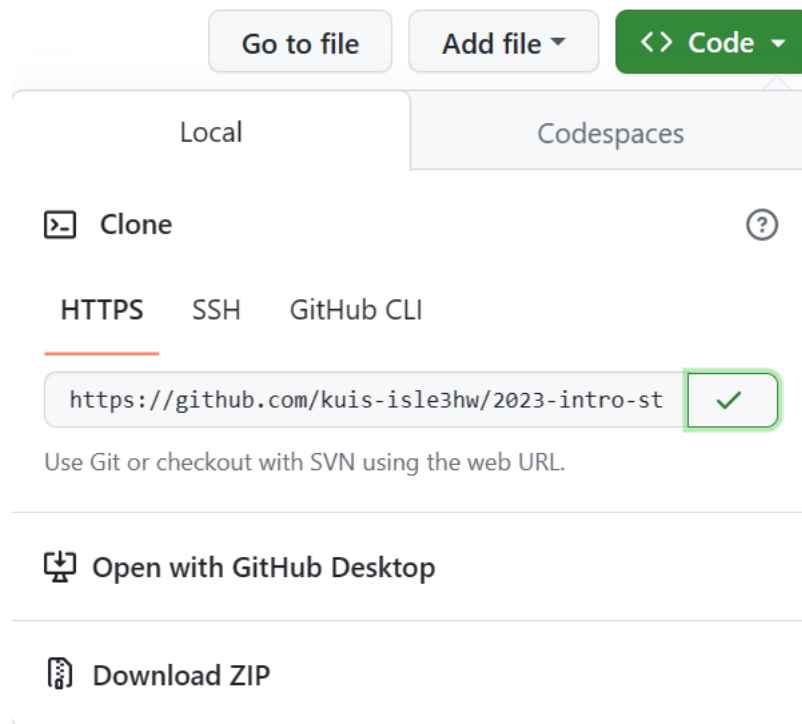
```
stela@jupiter:~$ git clone git@github.com:kuis-isle3hw/2023-intro-stelaseo.git
Cloning into '2023-intro-stelaseo'...
Enter passphrase for key '/home/stela/.ssh/id_rsa':
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 1 (delta 0), pack-reused 0
Receiving objects: 100% (3/3), done.
stela@jupiter:~$
```

## ⑤ GitHubをVS2022内で アクセス

# Visual Studio 2022内でGitを使う

公開鍵をGitHub登録せずIDE内で使いたい場合

GitHubからrepositoryのHTTPSをコピー



## VSでcloneを始める



### Clone a repository

Get code from an online repository like GitHub or Azure DevOps

### Clone a repository


Enter a Git repository URL

Repository location

Path

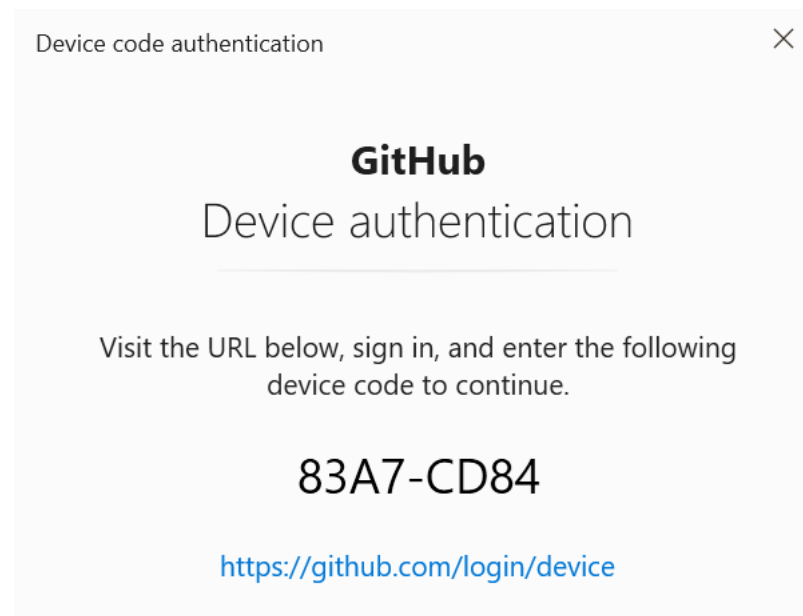
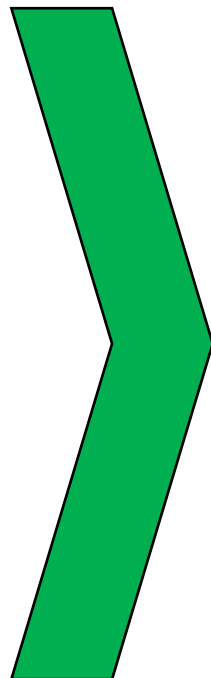
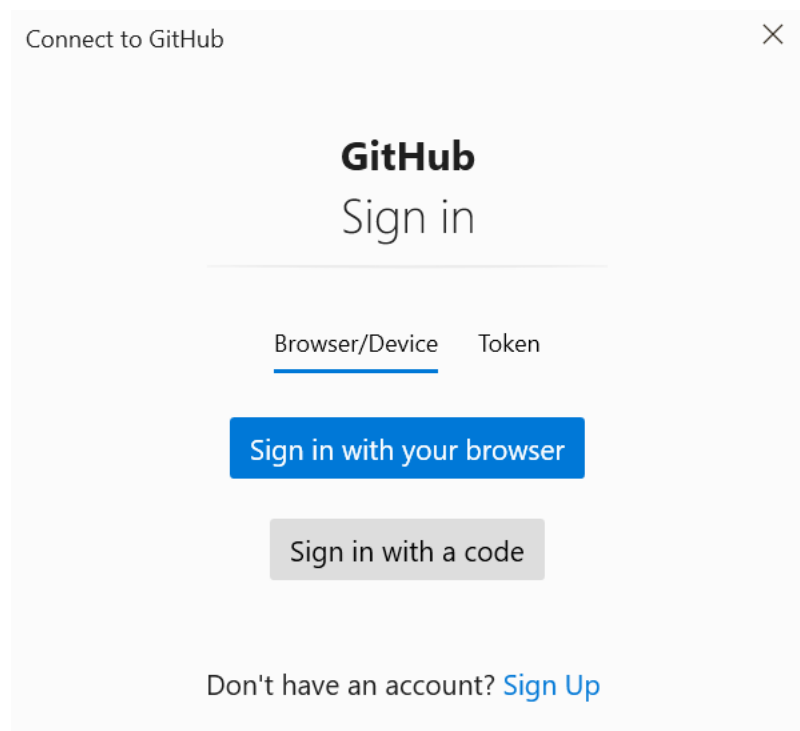


### Browse a repository

 Azure DevOps

 GitHub

## Popupから「Sign in with a code」を選択



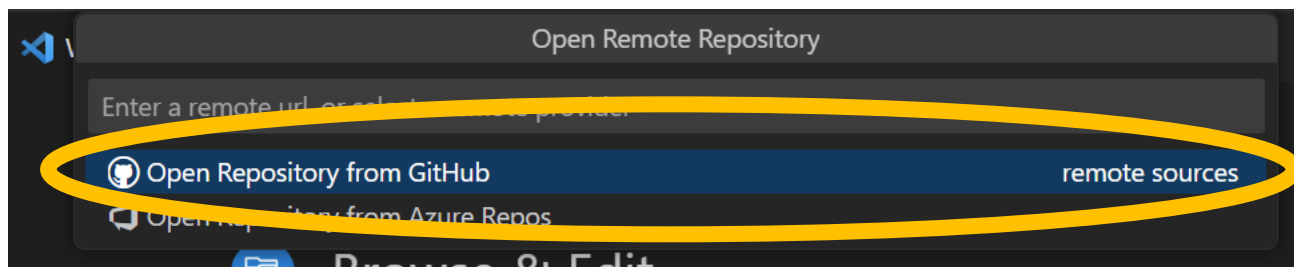
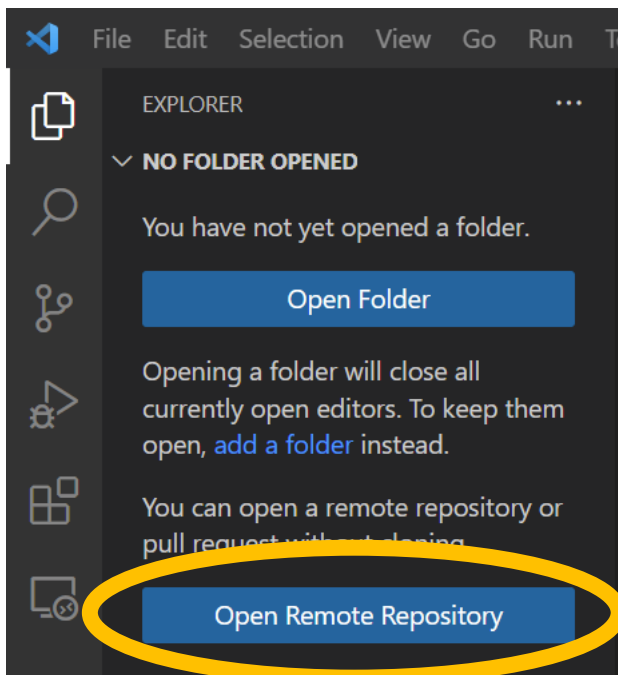
<https://github.com/login/device> のページに  
コードを入力する

## ⑤ GitHubをVS Code内で アクセス

# Visual Studio CodeでGitを使う

## Open Remote Repositoryを選択



## Open Repository from GitHubを選択



# Visual Studio CodeでGitを使う

## ログイン後

## Authorize Visual-Studio-Codeを押す




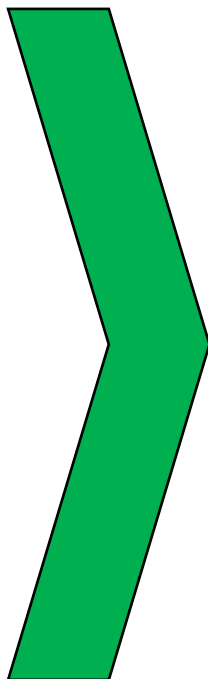
Sign in to GitHub  
to continue to GitHub for VS Code

Username or email address


Password [Forgot password?](#)


**Sign in**


New to GitHub? [Create an account.](#)




Authorize GitHub for VS Code

 **GitHub for VS Code by Visual Studio Code**  
wants to access your stelaseo account

 **Personal user data** ▼  
Email addresses (read-only), profile information (read-only)

 **Repositories** ▼  
Public and private

 **Workflow** ▼  
Update GitHub Action Workflow files.

Authorizing will redirect to  
<https://vscode.dev>

## ⑥ GitHub Classroomの利用

## GitHub Classroom

- **GitHub Classroom は GitHub を講義で利用するためのプラットフォーム**
  - 個人用，グループ用の課題の作成が可能
  - 各学生の個人GitHub アカウントで作成したリポジトリとリンクさせることで課題用のリポジトリを一括管理
  - 本実験ではこのGitHub Classroom を利用し，  
課題は各リポジトリでRelease を作成することで提出

## GitHub Classroomの導入

- 招待リンクはPandA内のお知らせを参照
  - リポジトリは2種
    - 導入課題（個人課題）：2025-Intro
    - プロセッサ課題（グループ課題）：2025-Simple
- まずGitHub Classroom と GitHub の連携について authorizationを求められるので承認
  - GitHub アカウントを持っていない人はここで作成を

## 導入課題

- 導入課題は各学生 1 人に対して 1 つのリポジトリ



Join the classroom:

kuis-isle3hw-classroom-2021

To join the GitHub Classroom for this course, please select yourself from the list below to associate your GitHub account with your school's identifier (i.e., your name, ID, or email).

Can't find your name? Skip to the next step

Identifiers
1029296773
1029299434
1029302951
1029304955
1029307671
1029309236
1029310012
1029310040

Identifiersの中から  
自分の学生番号を探し出してクリック  
→ Accept でリポジトリが作成される  
2025-intro-XXX (Github アカウント名)

※自分の学生番号が見つからない  
場合はスタッフまで連絡を

リポジトリが作成されたら、"git clone"で自分端  
末にダウンロードし、そのフォルダでquartusのプ  
ロジェクトを作り課題を進める

## プロセッサ課題

- プロセッサ課題は基本的に **学生 2 人に 1 つのリポジトリ**
  - 3 人グループもある
- グループはこちらで割り振って連絡しますので後ほど PandA を確認してください。
- 1 人の学生がチームの作成（連動してリポジトリの作成）  
→ もう 1 人の学生は作成されたチームを探して参加
  - チーム名はグループ分けの番号と対応して「teamXX」（例：team01）としてください。
    - リポジトリは 2025-simple-teamXX が作成されます。

## プロセッサ課題

- リポジトリが作成されたら、グループの一人が[clone](#)で自分の端末にダウンロードし、そのフォルダでquartusのプロジェクトを作成し、push。
- 別のグループ員は上記作業後に[clone](#)して、以後それぞれの端末で課題を進める。

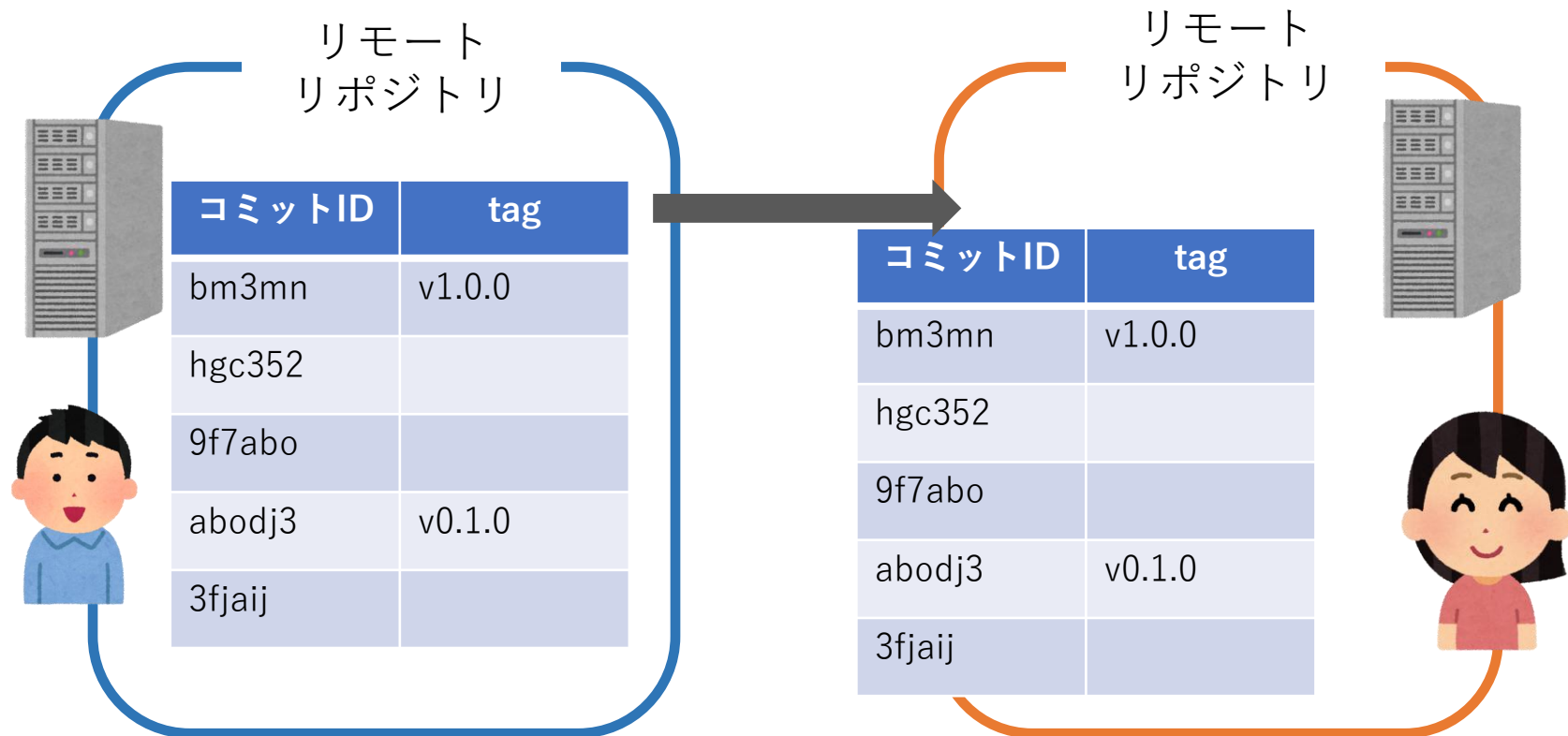
## 課題の提出

- 導入課題、プロセッサ課題の提出はGithubの**release**を使って行う。
- **release**については本資料の[p61](#)と[p81～88](#)を参照。

## ⑥ その他のGitHub機能

## fork

- 他の人のリモートリポジトリを自分のリモートリポジトリにコピーする。



## forkの方法

The screenshot shows the GitHub repository page for 'octocat / Spoon-Knife'. At the top right, there are buttons for 'Watch' (373), 'Star' (10.2k), and 'Fork' (109k). The 'Fork' button is highlighted with a blue arrow. Below the repository name, there are tabs for 'Code', 'Issues' (1,010), 'Pull requests' (5,000+), 'Actions', 'Projects' (0), 'Wiki', 'Security', and 'Insights'. A message states 'This repo is for demonstration purposes only.' Below this, there are statistics for '3 commits', '3 branches', '0 packages', '0 releases', and '1 contributor'. There are buttons for 'Branch: master', 'New pull request', 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. A commit history table is visible below, with columns for the commit author, message, and date.

Commit	Message	Date
octocat	Pointing to the guide for forking	13 Feb 2014
	Pointing to the guide for forking	6 years ago
	Created index page for future collaborative edits	6 years ago
	Create styles.css and updated README	6 years ago

コピーしたいリポジトリページの  
リポジトリ名の右側にある  
"Fork"をクリック

## forkの方法

The screenshot shows the GitHub interface for a repository named 'kazunarikato / Spoon-Knife'. It is a forked repository from 'octocat/Spoon-Knife'. The repository has 0 stars, 0 forks, and 109k forks. The repository is currently on the 'master' branch. A blue callout box points to the 'Clone or download' button, which is highlighted in green. The callout box contains the text: '自分のアカウントにコピーしたリポジトリができる'.

Commit	Author	Message	Time
Latest commit d0dd1f6	octocat	Pointing to the guide for forking	6 years ago
index.html		Created index page for future collaborative edits	6 years ago
style.css		Creates style.css and updated README	6 years ago

自分のアカウントに  
コピーしたリポジトリができる

### Pull request

- 新機能を追加し、ファイルをcommitした時、いきなりリモートリポジトリのmainにpushすると、そこにバグが含まれていた時に問題となる。
- **Pull request**を使うと、pushする前にローカルリポジトリでの変更内容をリモートリポジトリを共有している他の人に通知することができる。
- 変更内容を確認後、リモートリポジトリ上でmergeすることで、リモートリポジトリが更新される。
- コード・レビューがしやすくなる。

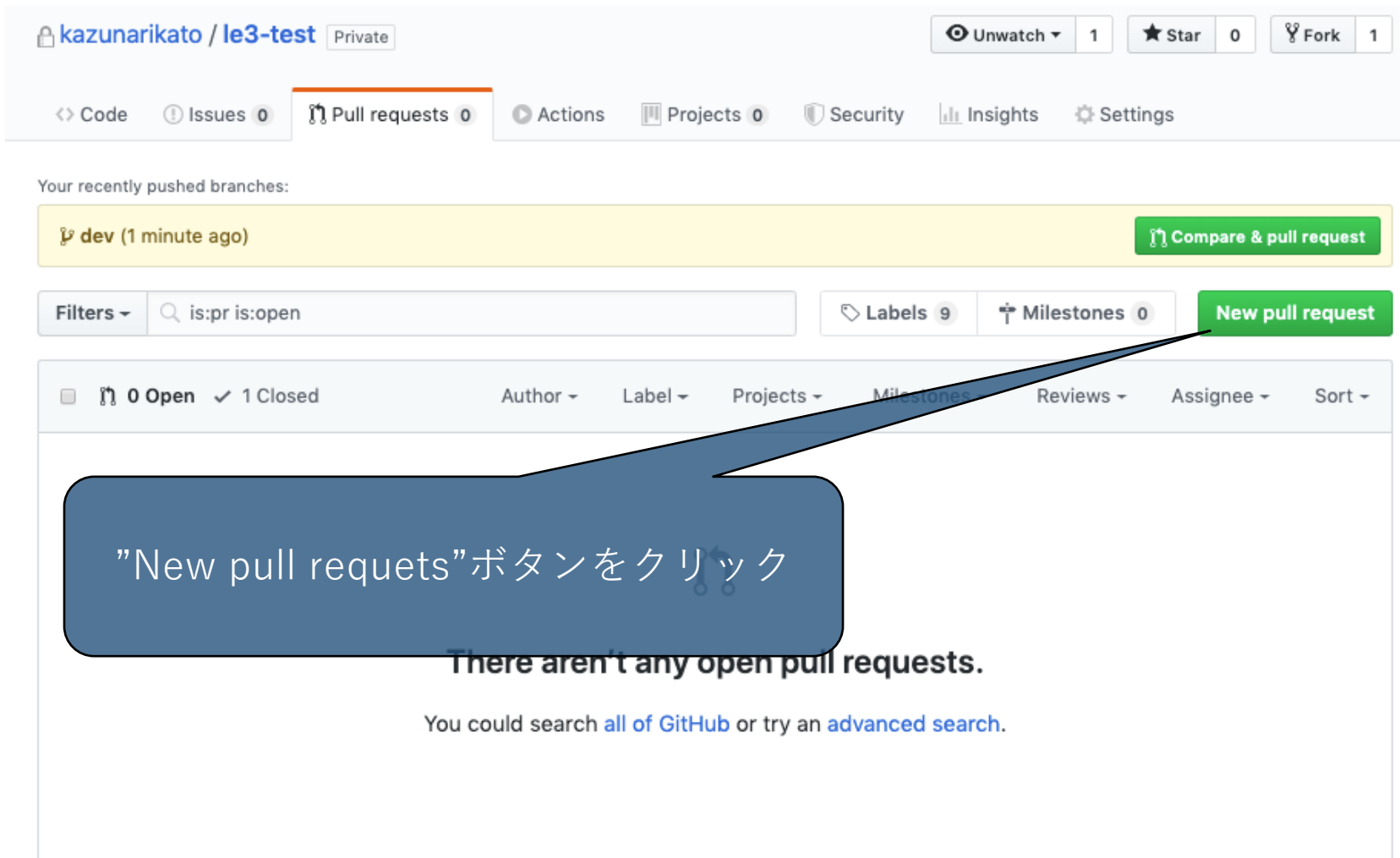
## Pull requestの作成

The screenshot shows the GitHub repository page for 'kazunarikato / le3-test'. The 'Pull requests' tab is selected in the navigation bar. A blue callout box points to the 'Pull requests' link. The repository has 4 commits, 1 branch, 0 packages, and 2 releases. The 'dev' branch is highlighted as the recently pushed branch. Below the navigation bar, there are buttons for 'Create new file', 'Upload files', 'Find file', and 'Clone or download'. The commit history shows a 'first commit' and a 'rev test.txt' commit, both 18 days ago.

File	Commit	Time
README.md	first commit	18 days ago
test.txt	rev test.txt	18 days ago

リポジトリページの  
"Pull requests"をクリック

## Pull requestの作成



The screenshot shows the GitHub interface for a repository named 'kazunarikato / le3-test'. The 'Pull requests' tab is selected, showing 0 open pull requests. A blue callout box points to the 'New pull request' button with the text: "New pull requests" ボタンをクリック

There aren't any open pull requests.  
You could search [all of GitHub](#) or try an [advanced search](#).

## Pull requestの作成

### Compare changes

Compare changes across branches, commits, tags, and more below. If you need to, you can also [compare across forks](#).

base: master ← compare: master

Choose a head ref

Find a branch

Branches Tags

✓ master default

dev

Create pull request

mergeするブランチを決める。  
“base:”にmerge先ブランチ  
“compare:”にmerge元ブランチを指定  
する。

### Compare and review just about anything

Branches, tags, commit ranges, and time ranges. [In the same repository and across forks.](#)

EXAMPLE COMPARISONS	
<a href="#">dev</a>	1 minute ago
<a href="#">master@{1day}...master</a>	24 hours ago

この例では“master”←“dev”のmergeを  
リクエストするので、compare:に“dev”  
を選択する。

## Pull requestの作成

### Comparing changes

Choose two branches to see what's changed or to start a new pull request. If you need to, you can also [compare across forks](#).

🔄 base: master ← compare: dev ✓ **Able to merge.** These branches can be automatically merged.

**Create pull request** Preview and review the changes in this comparison.

🔗 1 commit      📄 1 file changed      💬 0 commit comments      👤 1 contributor

“Create pull request”ボタンをクリックする。

📅 Commits on Apr 10, 2020

🏠 kazunarikato      mod readme.md      1a786eb

📄 Showing 1 **changed file** with 1 addition and 0 deletions.      Unified   Split

```
▼ 1 ■■■■■ readme.md 📄
...  ...  @@ -1 +1,2 @@
1   1     le3-testle3-testle3-test
2   2     + devで編集
```

## Pull requestの作成

### Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also compare across forks.

base: master ← compare: dev ✓ Able to merge. These branches can be automatically merged.

mod readme.md

Write Preview

AA B i “ <> @

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Create pull request

Reviewers

No reviews

Assignees

No one assigned

Labels

None yet

Projects

None yet

Milestone

No milestone

Linked issues

Use closing keywords in the

Pull requestの名前を記入する。

Pull requestの内容を記入する

“Create pull request”ボタンをクリックする。

## Pull requestの作成

The screenshot shows a GitHub Pull Request titled "mod readme.md #2" created by user "kazunarikato". The PR is open and ready for review. The interface includes a header with the PR title and an "Edit" button. Below the header, there are statistics for Conversation (0), Commits (1), Checks (0), and Files changed (1). A comment from "kazunarikato" is visible, stating "No description provided." Below the comment, a commit for "mod readme.md" is shown with the hash "1a786eb". A green box highlights the "Merge pull request" button and the status "This branch has no conflicts with the base branch". A blue callout box with the text "Pull requestが作成された。" (Pull request has been created.) is overlaid on the "Merge pull request" button. The right sidebar shows settings for Reviewers, Assignees, Labels, Projects, Milestone, and Linked issues. At the bottom, there is a comment input area with "Write" and "Preview" tabs, and buttons for "Close pull request" and "Comment".

## Pull requestの作成

### mod readme.md #2

Edit

 Open kazunarikato wants to merge 1 commit into master from dev 

 Conversation 0  Commits 1  Checks 0  Files changed 1 +1 -0 

Changes from all commits  File filter...  Jump to...  

0 / 1 files viewed  Review changes -

▼ 1  readme.md 

   Viewed 

... @@ -1 +1,2 @@

1 1 le3-testle3-testle3-test

2 + devで編集

 ProTip! Use  and  to navigate between files in a pull request.

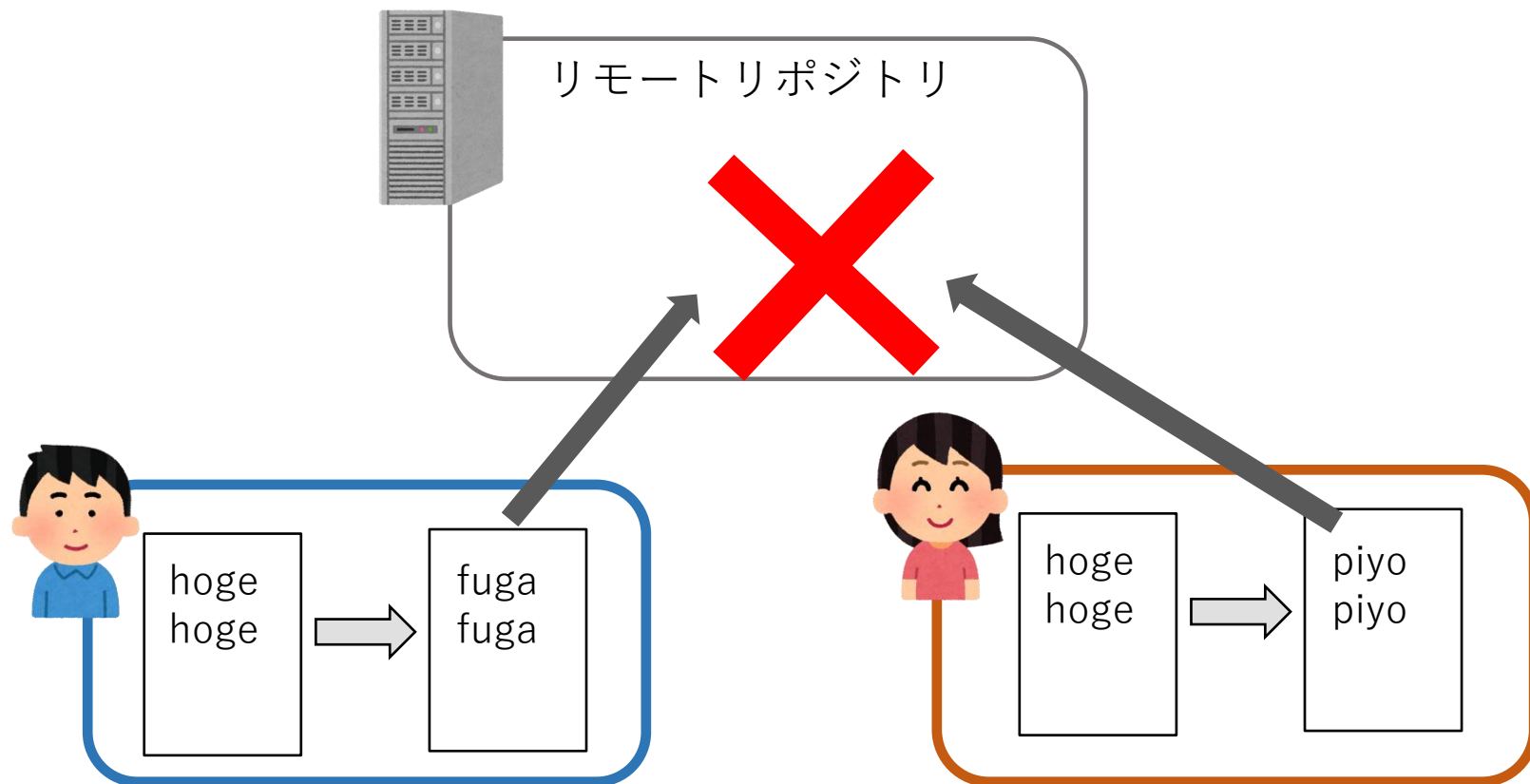
”File changed”をクリックすると、ファイルの変更箇所を確認できる。

## Pull requestの作成

The screenshot shows a GitHub pull request titled "mod readme.md #2" by user "kazunarikato". The interface includes a header with "Open", "1 commit", "0 checks", and "1 file changed". A comment from "kazunarikato" is visible. A callout points to the "Conversation" tab, stating: "“Conversation”をクリック。". Below the comment is a merge confirmation form with fields for "Merge pull request #2 from kazunarikato/dev", "mod readme.md", and the email "kato.kazunari5515@gmail.com". A callout points to the "Confirm merge" button, stating: "“Confirm merge”ボタンをクリックすると、mergeされる。". The right sidebar shows "Reviewers", "Assignees", "Labels", "Projects", "Milestone", and "Linked issues".

## コンフリクト

- 他の人が編集したファイルに重複した編集をした状態でpushやpullをするとコンフリクトが起きる。（1人で作業する場合でもmergeの時に起きる可能性あり）



### コンフリクト対策

- コンフリクトを未然に防ぐ
  - .gitignoreを設定して、ログファイルなど不要なファイルはGitの管理対象外にする。  
(<https://docs.github.com/ja/get-started/git-basics/ignoring-files>)
  - プルリクエストを使う。
    - まず、作業前（ファイル編集前）にpullする。
  - 二人で1つのファイルを同時に編集しないようにする。
    - ブランチを活用する。
  - こまめに、コミットする。

### コンフリクト対策

- コンフリクトしてしまったら
  - git status コマンドでコンフリクトしているファイルを確認。
  - 直接ファイルを開く。編集が重複している部分が“<<<<HEAD”、“=====”、“>>>>”でハイライトされているので、不要部分を削除する。
  - 編集したファイルをステージング、コミットする。

- サルでもわかるGit入門  
<http://www.backlog.jp/git-guide/>
- Pro Git book（日本語版）  
<https://git-scm.com/book/ja/v2>
- Git Cheat Sheet  
<https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf>
- GitHubヘルプドキュメント  
<https://help.github.com/ja/github>