

void setupで学ぶGit GitHub

永谷研究室 2022/04/20
作成者 田邊 匠

はじめに

- ・本勉強会ではGit, GitHubのを使った開発の流れをハンズオン形式で学習する
Git, GitHubを使ったことがない、アカウントを持っているが開発したことがない方を対象としている
すでにGit, GitHubを使ってバージョンの管理やチームでの開発を行ったことのある方は新たに得られる知識や習得できるものはないと思われるので本勉強会の内容を行わなくても良い
- ・void setupとは
本勉強会で取り扱うvoid setupは太鼓の達人に導入されているナムコオリジナルの曲でありProcessingで書かれたソースコードがそのまま歌詞になっている
歌唱 愛原圭織
作詞(プログラミング) 祇羽
作曲・編曲・トラックダウン、譜面作成 増渕裕二

参考：太鼓の達人譜面とかWiki

<https://www.wikihouse.com/taiko/index.php?%C6%F1%B0%D7%C5%D9%C9%BD%2F%A4%AA%A4%CB%2Fvoid%20setup>

Gitとは

- ・ソースコードをはじめとしたファイルの変更履歴(バージョン)を管理するためのシステム
- ・Gitの特徴として、「分散型」のバージョン管理システムである点が挙げられる
バージョン管理システムは大きく「集中型」と「分散型」に分けられる

「集中型」：特定の場所にあるリポジトリへの接続が必須になる

「分散型」：個々人のマシン上にリポジトリを作成して開発を行うことができる
現在のチーム開発における主流

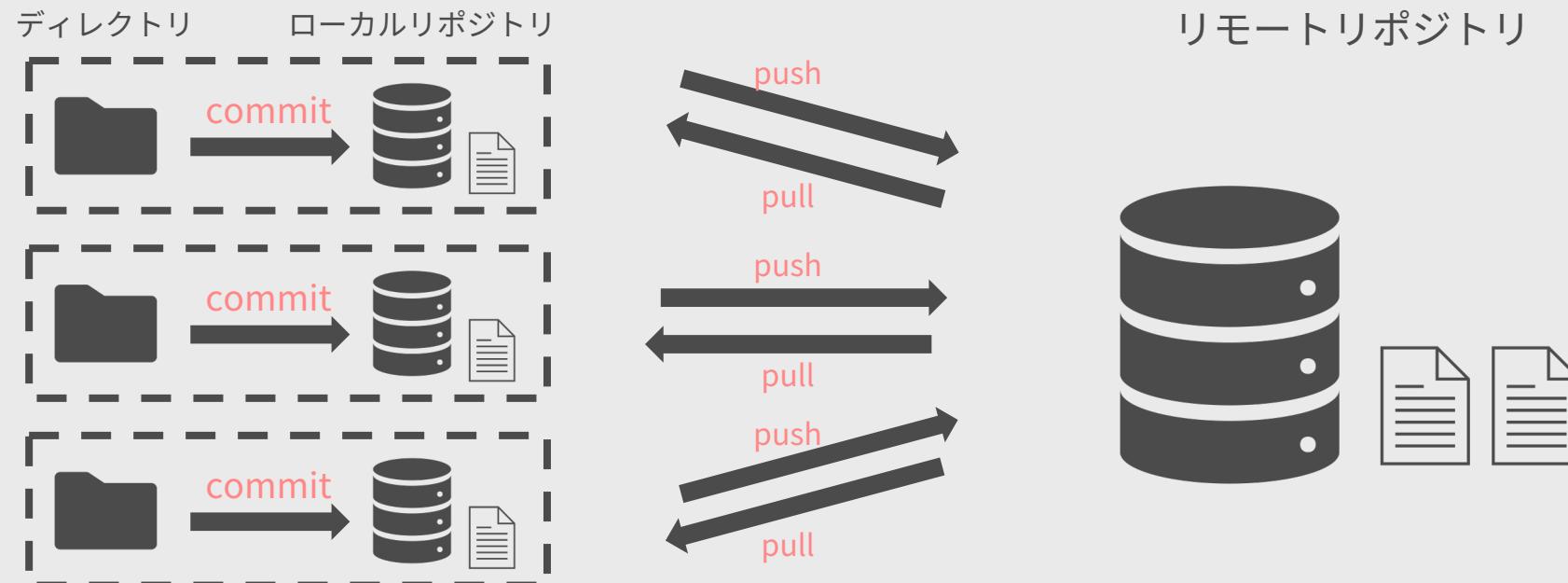


参考：【超入門】初心者のためのGitとGitHubの使い方

<https://tech-blog.rakus.co.jp/entry/20200529/git>

リポジトリとは

- ・バージョン管理によって管理されるファイルと履歴情報を保管する領域
- ・Gitの場合、個々人のマシン上にあるリポジトリ上で作業を行い、作業内容をサーバ上などにあるリポジトリに集約する流れで開発を進める



参考：【超入門】初心者のためのGitとGitHubの使い方

<https://tech-blog.rakus.co.jp/entry/20200529/git>

GitHubとは

- ・複数人のエンジニアがリモートリポジトリとして活用する他、チーム開発を行うための機能を提供するWebサービス
- ・リポジトリとしての機能をもつ他に、コードレビュー機能やWikiなどのコミュニケーションツールとしての機能を持っている



GitHub

参考：【超入門】初心者のためのGitとGitHubの使い方

<https://tech-blog.rakus.co.jp/entry/20200529/git>

勉強会の流れ

- ・本勉強会では以下の流れに沿って行う
- ・基本的にはCLI操作で行う、リモートリポジトリの作成などの一部操作はGUI操作で行う

- ・ Gitのインストール
- ・ Gitの初期設定
- ・ GitHubのアカウント作成
- ・ GitHubのアクセストークンの作成
- ・ リモートリポジトリの作成
- ・ ローカルリポジトリの作成
- ・ ローカルリポジトリにコミット
- ・ リモートリポジトリにプッシュ
- ・ ブランチの作成
- ・ ブランチでコミット・プッシュ
- ・ プルリクエスト・マージ
- ・ コンフリクト対応
- ・ ブランチの削除・復元
- ・ バージョン管理の例とおすすめのサイト

Gitのインストール1/2

- Macの場合

基本的にMacだと標準でインストールされいているのでインストール作業は不要
Gitのバージョンアップがめんどくさいなどの理由がある場合は適宜Homebrewなどで
インストールすること

- Windowsの場合

公式のページ(<https://gitforwindows.org/>)からインストーラをダウンロードし、インストーラの
指示に従ってインストールする
本勉強会で行う作業ではインストーラの設問は全てデフォルトのままでも問題ない

- Linuxの場合

Debian, Ubuntu系の場合 `sudo apt install git`

RedHat, CentOS系の場合 `sudo yum install git`

Gitのインストール2/2

- Gitのインストールが完了したら以下のコマンドでGitがインストールされていることを確認
Windowsの場合はGit Bashを起動してコマンドを入力

```
$ which git  
/usr/bin/git
```

```
$ git --version  
git version 2.32.0 (Apple Git-132)
```

Gitの初期設定

- Gitではソースコードの変更履歴を確認できるが、「誰が」変更したかを確認するための情報が必要となる

作業者を識別するための情報として、ユーザ名とメールアドレスを登録する必要がある
すでに登録している方はこの作業はスキップ

- 以下のコマンドを入力して登録を行う

ユーザ名の登録

```
$ git config --global user.name 任意のユーザ名
```

メールアドレスの登録

```
$ git config --global user.email 任意のメールアドレス
```

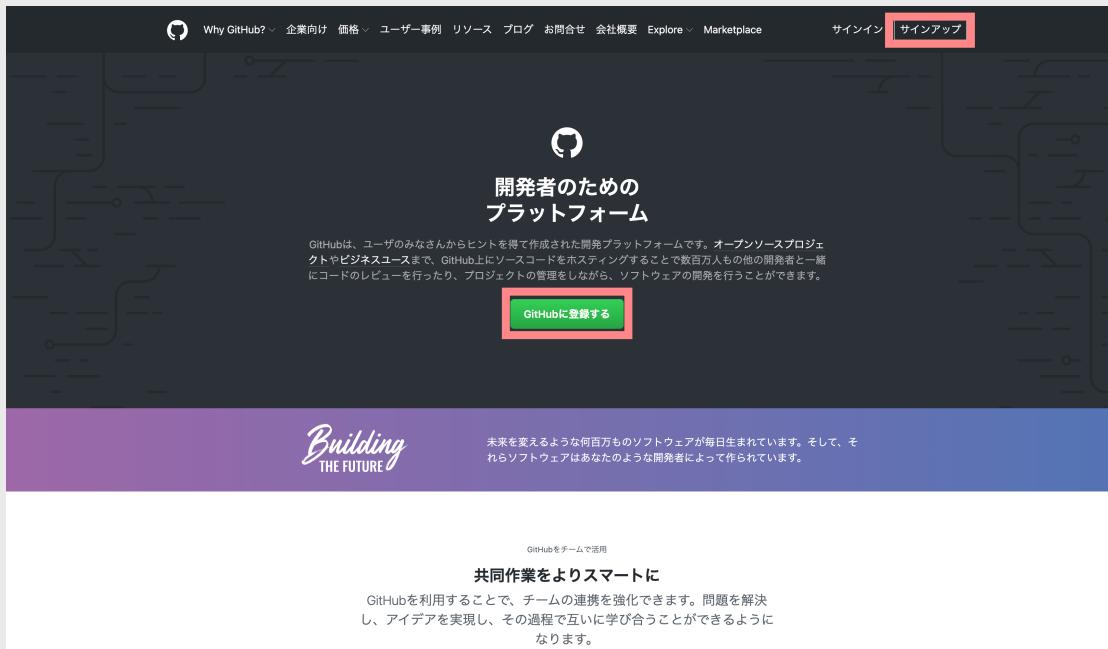
登録内容の確認

```
$ git config --list
credential.helper=osxkeychain
user.name=設定したユーザ名
user.email=設定したメールアドレス
core.repositoryformatversion=0
```

•
•
•

GitHubのアカウント作成

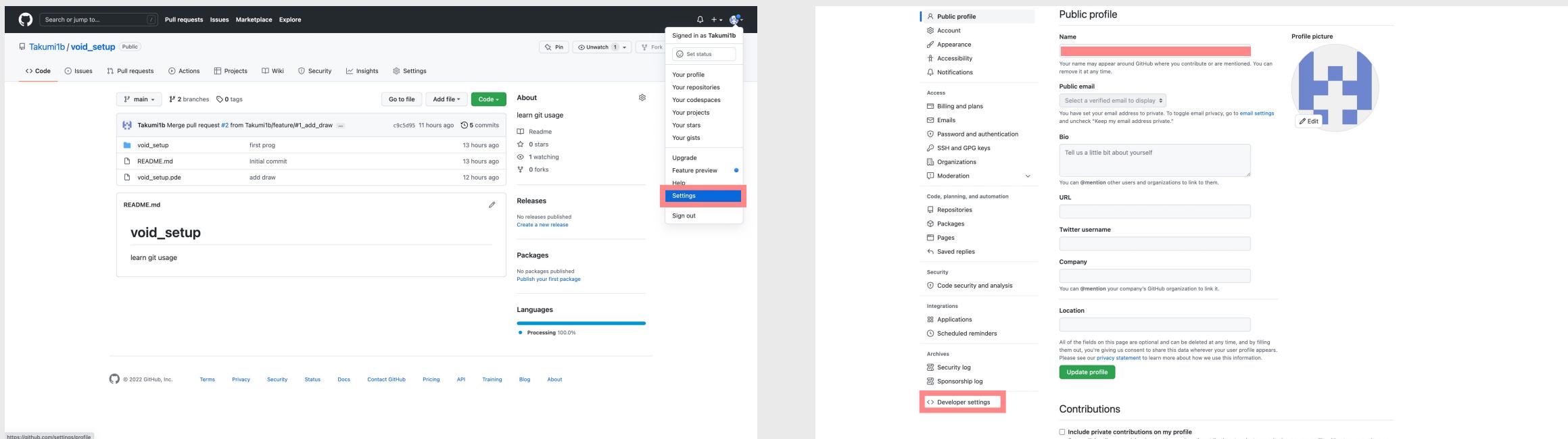
- GitHubのアカウントを作成する 公式ページ(<https://github.co.jp/>)
すでにアカウントを持っている方はこの作業をスキップ
アカウント作成の際にメールアドレスに確認コードが送られる可能性があるので
それぞれ都度対応すること



The screenshot shows the "Create your account" form on GitHub. The form is enclosed in a red box. It has three input fields: "Username", "Email address", and "Password", each with a green checkmark icon. Below the password field is a note: "Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)". There are two sections below the inputs: "Email preferences" (with an unchecked checkbox) and "Verify your account" (with a text input field containing Japanese text and a "Verify" button). At the top of the form, there's a "Join GitHub" header and a "Create your account" sub-header.

GitHubのアクセストークンの作成1/3

- リモートリポジトリにpushする際に必要となる場合がある
すでに取得している方はこの作業をスキップ
- 右上のプロフィールアイコンからsettingを選択し、プロフィール編集ページのサイドメニューをスクロールしてDeveloper settingsを選択する



GitHubのアクセストークンの作成2/3

- 開発者用の認証設定ページが開くのでサイドメニューからPersonal access tokensを選択しGenerate new tokenを選択
新しいトークン作成ページが開くので必要事項を記入
Noteはトークンの説明、Expirationは有効期間
本勉強会ではリポジトリを操作するのみであるのでrepoのみのチェックで十分

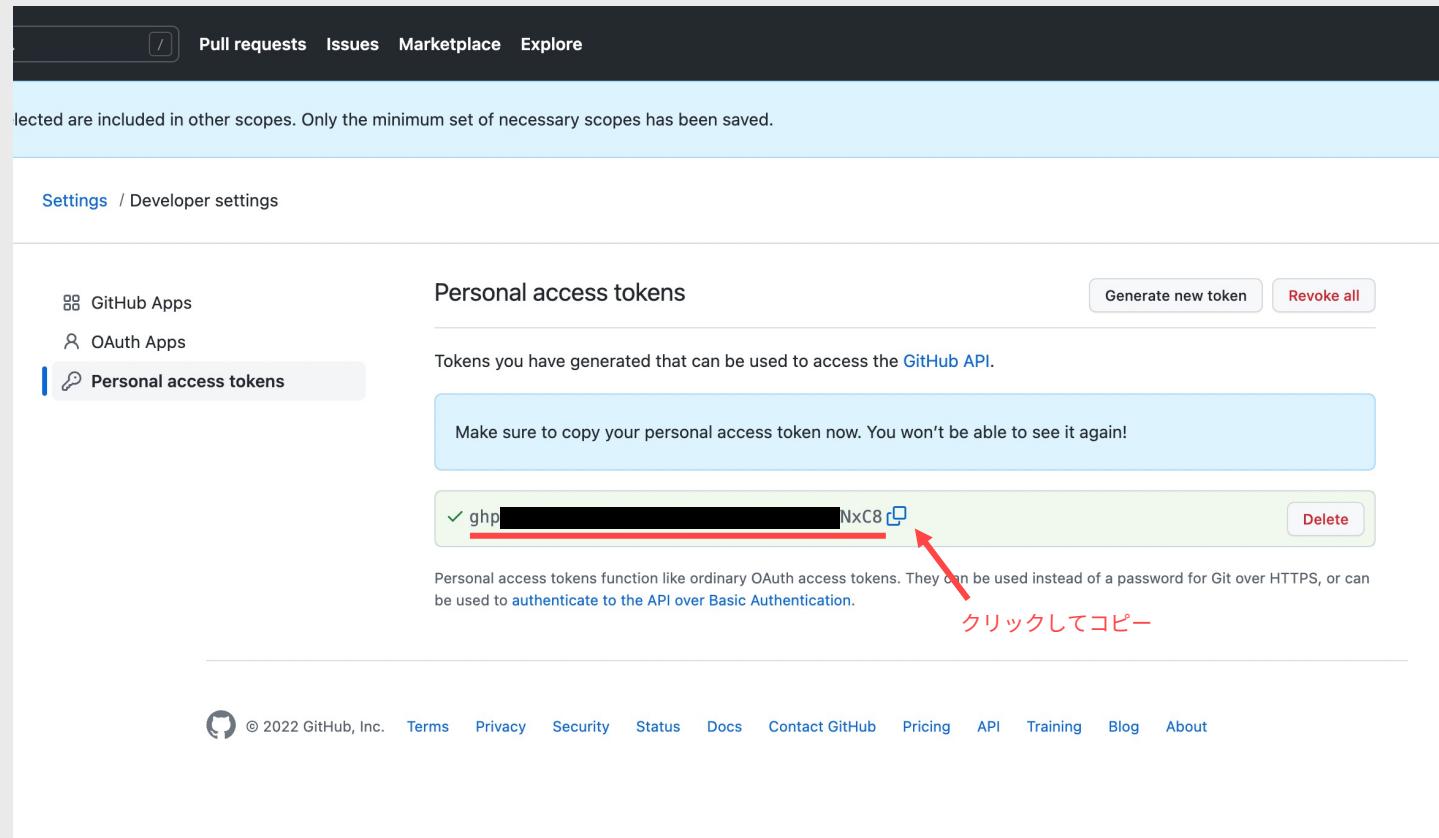
The screenshot shows the GitHub developer settings page. In the sidebar, 'Personal access tokens' is highlighted with a red box. At the top right, there is a 'Generate new token' button. Below it, there is a note about what personal access tokens are used for. The bottom of the page includes standard GitHub footer links.

The screenshot shows the 'New personal access token' configuration page. It includes fields for 'What's this token for?' (set to 'for_repository'), 'Expiration' (set to '90 days'), and a large 'Select scopes' section. The 'repo' scope is checked, granting full control of private repositories. Other scopes listed include 'repo-status', 'repo_deployment', 'public_repo', 'repo:write', 'security_events', 'workflow', 'write:packages', 'read:packages', 'delete:packages', 'admin:org', 'write:org', 'read:org', 'admin:public_key', 'write:public_key', and 'read:public_key'. Descriptions for each scope are provided on the right.

GitHubのアクセストークンの作成3/3

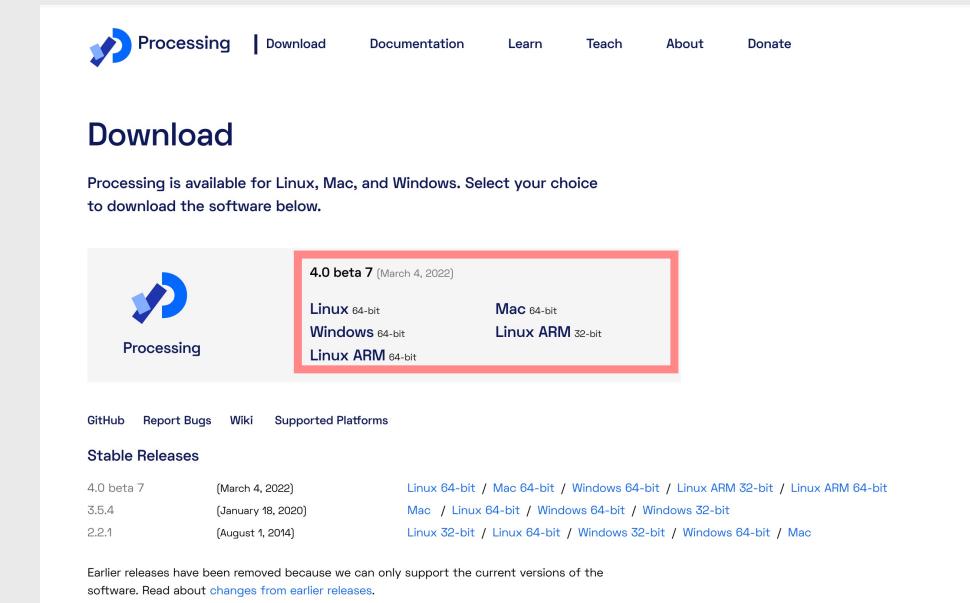
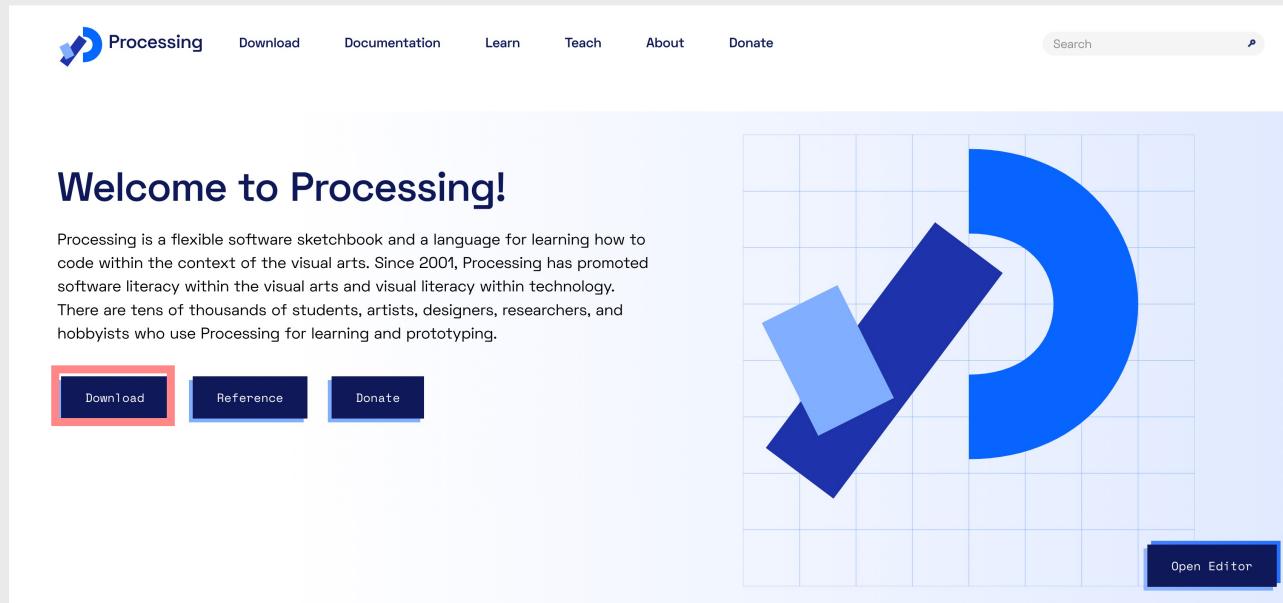
- 必要事項の入力完了後、Generate tokenをクリックすると下記のようにアクセストークンが生成される

注：トークンは作成時にしか表示されないので、忘れた際は再取得する必要がある



Processingのインストール

- 本勉強会で題材として扱うプログラムはProcessingで記述する
公式ページ(<https://processing.org/>)からProcessingのインストーラをダウンロードする
永谷研究室に在籍している方はすでにインストールされていると思うのでこの作業はスキップ



リモートリポジトリの作成

- GitHubにアクセスし、プロフィール画面(トップ画面からでも可)からリモートリポジトリを作成
 - Repository name : リポジトリの名前
 - public : 他のユーザがソースコードを閲覧可
 - Add a README file : READMEファイルの追加
 - Choose the license : ライセンスの指定
- Description : リポジトリの説明
- private : 非公開設定
- Add .gitignore : 管理外ファイルの設定

The screenshot shows the GitHub profile page for 'Takumi1b'. The top navigation bar includes 'Search or jump to...', 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the navigation is a search bar with 'Find a repository...' and a red 'New' button. The main area displays a large circular profile picture with a blue and white pixelated logo. Below the picture, the user's name 'Takumi Tanabe' and handle 'Takumi1b' are shown, along with a 'Edit profile' button. A message states 'Takumi1b doesn't have any public repositories yet.' At the bottom, there are links for 'Edit profile', 'Joined 1 hour ago', and standard GitHub footer links like 'Terms', 'Privacy', 'Security', etc.

The screenshot shows the 'Create a new repository' form. The top section asks 'Create a new repository' and provides a link to 'Import a repository'. The 'Owner' dropdown is set to 'Takumi1b' and the 'Repository name' field contains 'void_setup'. The 'Description (optional)' field has 'learn git usage' entered. The 'Visibility' section shows 'Public' selected (radio button highlighted), while 'Private' is also available. The 'Initialize this repository with:' section includes 'Add a README file' (checkbox checked), 'Add .gitignore' (checkbox unchecked), and 'Choose a license' (checkbox unchecked). A note at the bottom says 'This will set `main` as the default branch. Change the default name in your settings.' At the bottom right is a green 'Create repository' button.

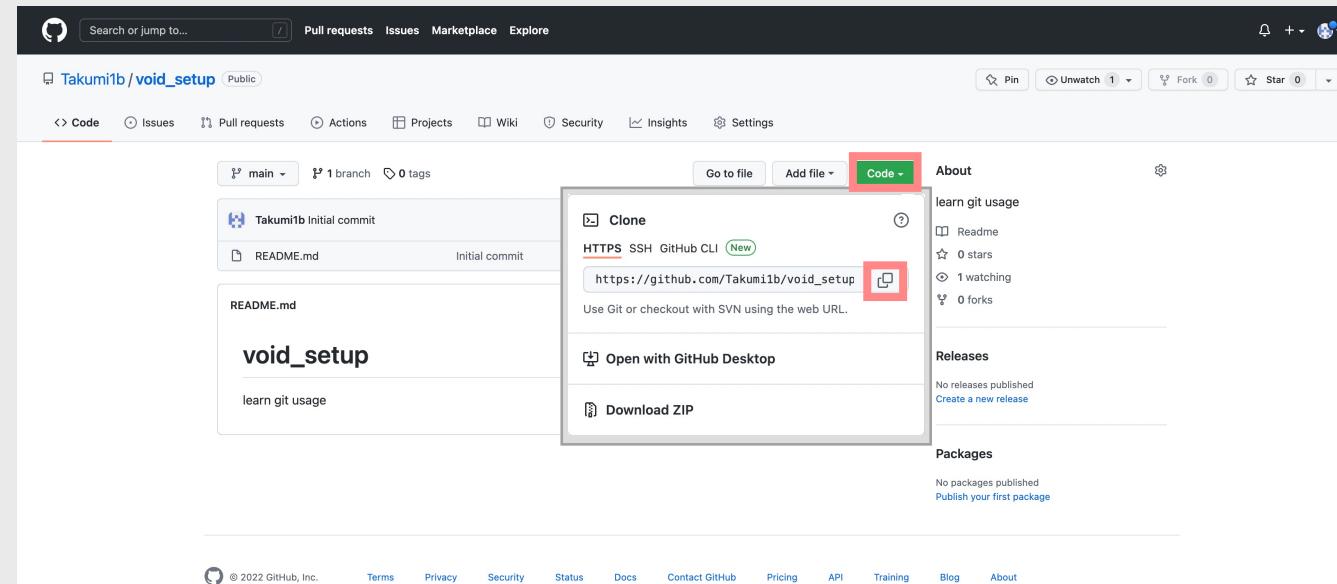
ローカルリポジトリの作成1/2

- 以下のコマンドを入力してリモートリポジトリと同じものをローカルに作成
ローカルリポジトリはcloneコマンドを入力したディレクトリ直下に生成される
リポジトリのURLはリポジトリページCodeを選択すると確認できる

```
$ git clone リポジトリURL
```

```
$ git clone https://github.com/ユーザ名/リポジトリ名.git
```

例 \$ git clone https://github.com/Takumi1b/void_setup.git



ローカルリポジトリの作成2/2

- 作成したローカルリポジトリ内で以下のコマンドを入力してリポジトリとして初期化

```
$ git init
```

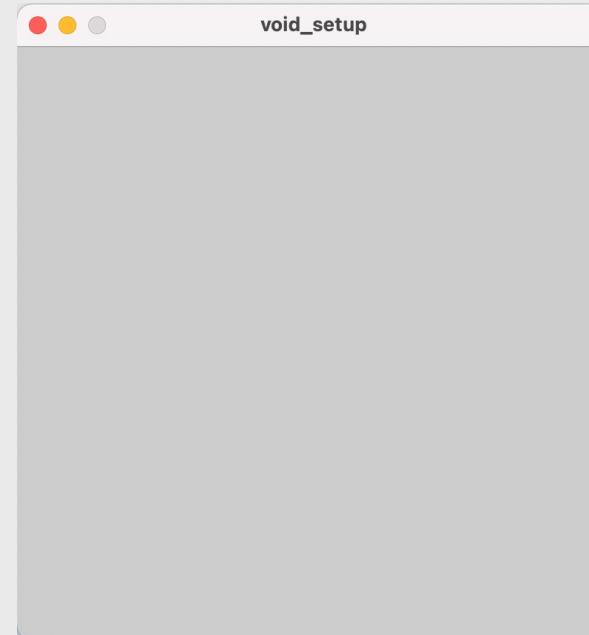
- ローカルリポジトリに追加するファイルを作成

Processingを開き次のコードを入力して保存

ディレクトリにはProcessingの内容を保存した時に生成されたディレクトリもしくは
.pdeファイルだけでも良い

Processing

```
1: void setup(){  
2:   size(400,400);  
3: }
```



実行結果

400×400のウィンドウ
が表示される

ローカルリポジトリにコミット

- ・コミット(commit)とは、ファイルの追加・変更をローカルリポジトリに反映する操作を意味します、以下のコマンドを入力し、インデックスへファイルを追加
インデックスとはコミット前に変更内容を一時的に保存する領域を指す
インデックスに追加されたファイルのみがコミットの対象となる

```
$ git add ファイル名 例 $ git add void_setup.pde
```

次に以下のコマンドを入力することでインデックスに存在するファイルがローカルリポジトリへ追加される

```
$ git commit -m “commit message”
```

-mはコミットメッセージを入力するためのオプションでありダブルクォーテーションで囲った部分に変更内容などのメッセージを記し、より詳細な履歴情報を残すことができる

リモートリポジトリにプッシュ

- ローカルリポジトリの内容をリモートリポジトリに反映させるため、以下のコマンドを実行する

```
$ git push origin main
```

コマンド実行後にGitHubのユーザ名とパスワードの入力を求められる

普通にGitHubのユーザ名とパスワードを入力すると、以下のような警告が出て
リモートリポジトリへの反映ができない

```
remote: Support for password authentication was removed on August 13, 2021. Please use a personal access token instead.
```

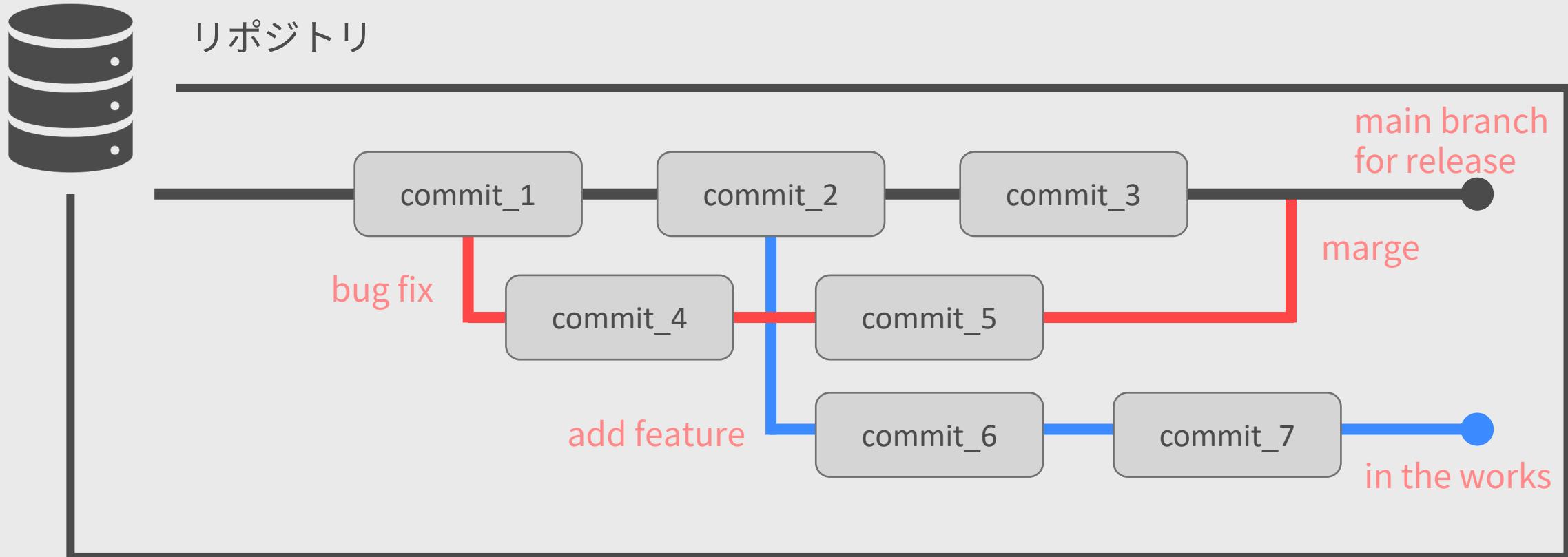
```
remote: Please see https://github.blog/2020-12-15-token-authentication-requirements-for-git-operations/ for more information.
```

```
fatal: Authentication failed for 'https://github.com/ユーザ名/リモートリポジトリ名.git/'
```

GitHubのパスワードを入力する際に生成したアクセストークンを入力することで実行できる
コマンドを実行すると、GitHubのページでも変更内容が反映されていることが確認できる

ブランチの作成1/3

- ・ ブランチ(branch)とは開発を並行して行うための仕組み
ある時点でのリポジトリの状態を複製して別名で保存したもの



ブランチの作成2/3

- 作業の開始時にどんな理由で作業しているのかを記録するためにissueを作成する
GitHubのリポジトリのページからissueタブを開き、New issueを選択
issueのタイトル、説明文を入力して Submit new issue を選択
issueが作成されると番号が振られる
その番号をブランチ名で使用すると今どの作業を行なっているのかが明確になる

The screenshot shows the 'Issues' tab of a GitHub repository named 'Takumi1b / void_setup'. A modal window titled 'Label issues and pull requests for new contributors' is displayed, informing the user that GitHub will help potential first-time contributors discover issues labeled with 'good first issue'. Below the modal, the main interface shows a search bar with the query 'is:issue is:open', a filter section with '0 Open' and '1 Closed', and a message stating 'There aren't any open issues.' At the bottom, there's a 'ProTip!' link and a footer with standard GitHub links.

The screenshot shows the 'New issue' dialog box on the same GitHub repository. The title field contains 'draw関数の追加' and the description field contains 'void_setupのdraw関数を追加する'. The right sidebar shows fields for 'Assignees', 'Labels', 'Projects', 'Milestone', and 'Development'. At the bottom right of the dialog is a red-bordered 'Submit new issue' button.

ブランチの作成3/3

- ・ ブランチの作成は以下のコマンドを実行することで行う
今回は作成したissueに従い機能の追加目的でブランチを作成する

\$ git branch ブランチ名

例 \$ git branch feature/#1_add_draw

ブランチの命名規則として、以下のようなものがオススメ

- ・ バグ修正目的の場合 : bugfix/#issueNo_作業名
- ・ 機能追加目的の場合 : feature/#issueNo_作業名

- ・ 作成したブランチで作業を行うために以下のコマンドを実行しブランチを移動

\$ git checkout ブランチ名

例 \$ git checkout feature/#1_add_draw

ブランチでコミット・プッシュ1/2

- ・ ブランチ内のProcessingコードを編集し、インデックスへのファイル追加とリモートリポジトリへの反映を行う
- ・ 右に示すようにプログラムに追加を行い、保存したのち以下のコマンドを実行する

\$ git add ファイル名

例 \$ git add void_setup.pde

\$ git commit -m “commit message”

\$ git push origin ブランチ名

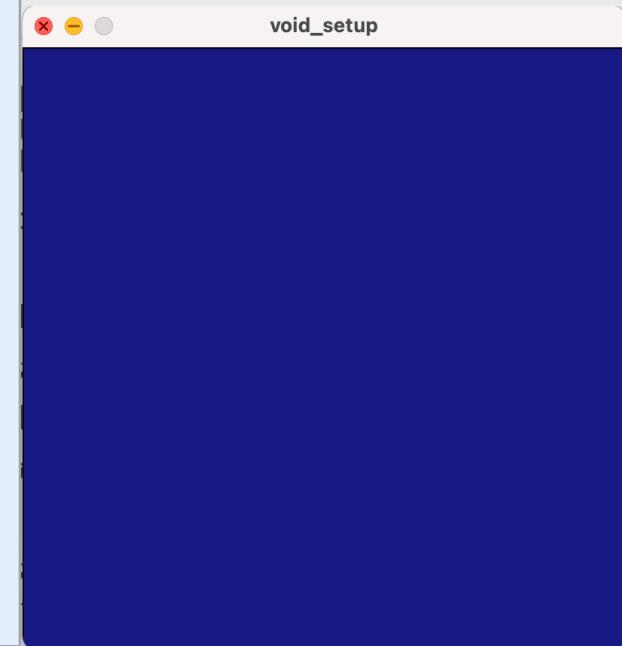
例 \$ git push origin feature/#1_add_draw

Processing

```
1: void setup(){
2:   size(400,400);
3: }
4:
5: float a = 0;
6:
7: void draw(){
8:   fill (#021050, 5);
9:   rect (0, 0, width, height);
10:  translate (200+a*10,200+a*10);
11:  rotate (a);
12:  scale (a/15);
13:  a+= 0.05;
14:  if (a>30) a-= 50;
15: }
```

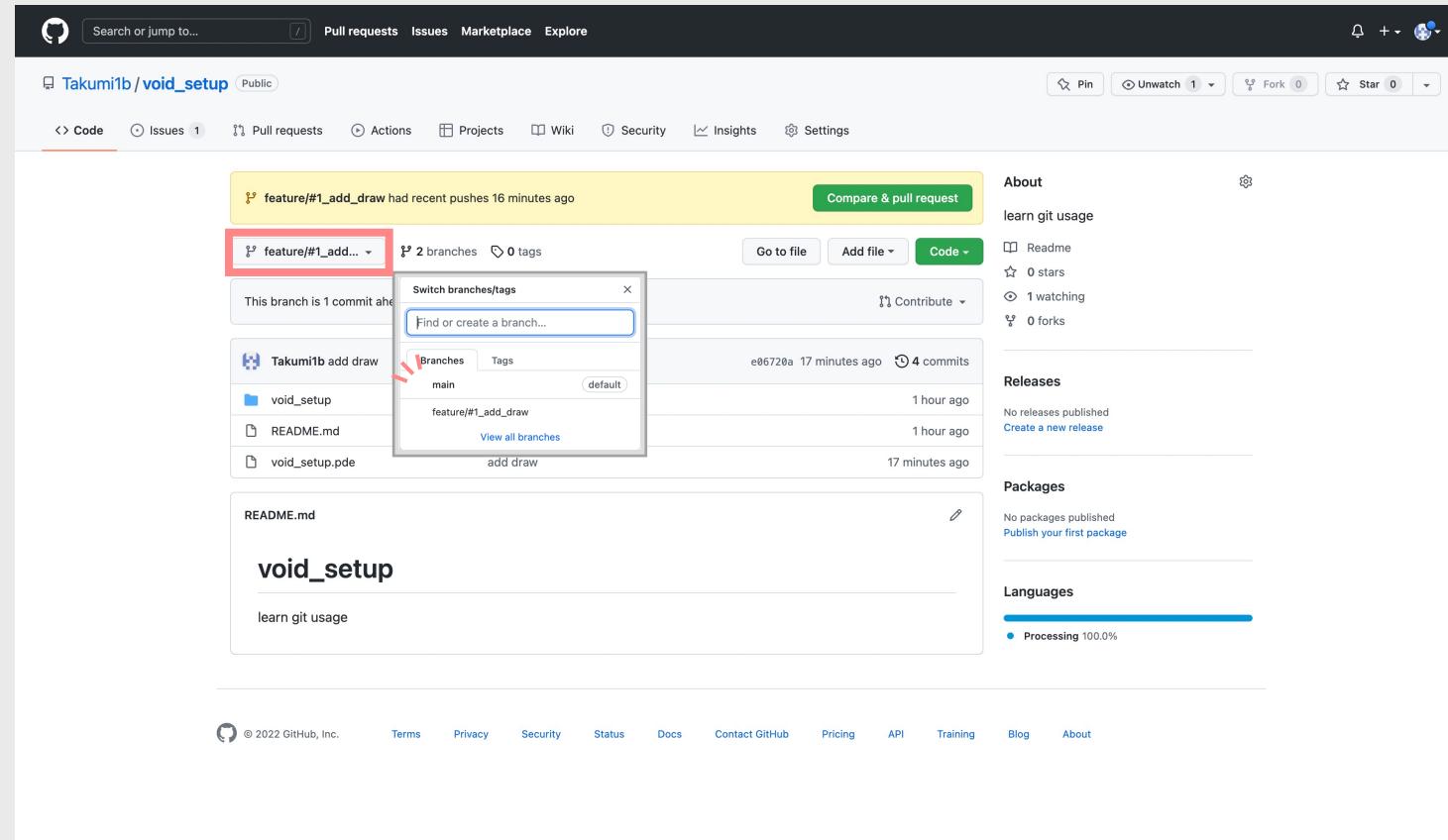
実行結果

400×400の青いウィンドウが表示される



ブランチでコミット・プッシュ2/2

- GitHubにアクセスすると2つのブランチが存在していることが確認できる
作成したブランチには変更内容が反映されていることが確認できる
逆に、mainブランチには変更内容が反映されていないことが確認できる



プルリクエスト・マージ1/4

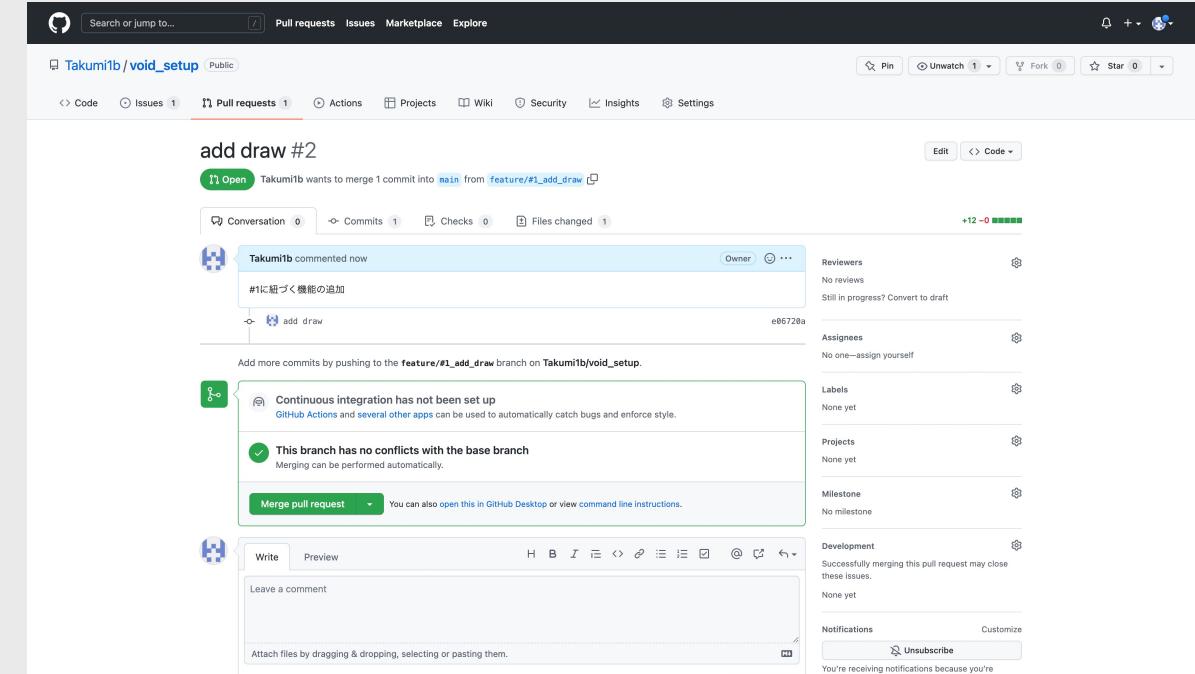
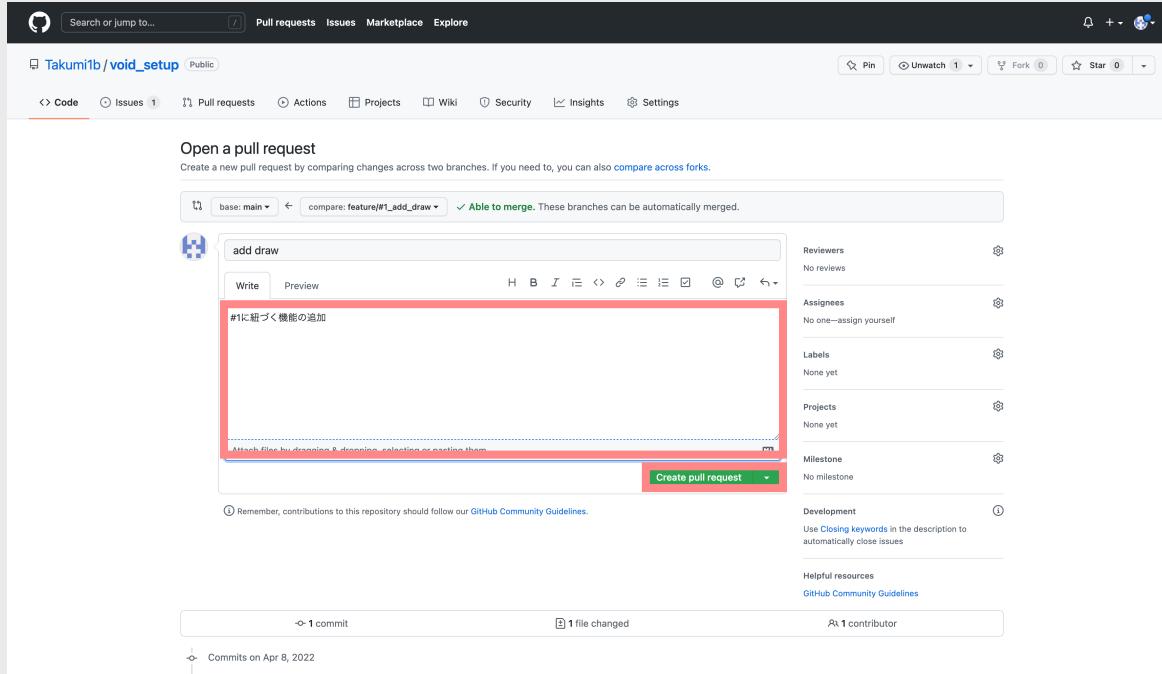
- ・マージとはmainブランチに作成したブランチを統合すること
- ・マージを行う際に、GitHub上ではプルリクエストを作成し、コードのレビューを行う
codeタブにあるCompare & pull requestもしくはpull requestタブのNew pull requestを選択

A screenshot of a GitHub repository page for 'Takumi1b/void_setup'. The repository is public and has 2 branches. A yellow box highlights the 'Compare & pull request' button in the top right corner of the main content area. Below it, the code tab shows a commit history with one commit ahead of the main branch. The void_setup branch contains files like README.md, void_setup.pde, and void_setup.pde. The void_setup file has a 'learn git usage' section.

A screenshot of the GitHub pull requests page for the same repository. A red box highlights the 'New pull request' button in the top right corner of the search bar. The page displays a message: 'Label issues and pull requests for new contributors. Now, GitHub will help potential first-time contributors discover issues labeled with good first issue.' Below the search bar, there are filters for 'Labels' and 'Milestones', and a summary showing 0 open and 1 closed pull requests. A message at the bottom states: 'There aren't any open pull requests. You could search all of GitHub or try an advanced search.'

プルリクエスト・マージ2/4

- ・ プルリクエスト名と説明文を入力し、Create pull requestを選択
説明文に「#No」のように入れるIssueとプルリクエストを紐づけることができる



プルリクエスト・マージ3/4

- 変更の差分はFiles Changedタブを選択することで下記のように変更内容の一覧が確認できる

The screenshot shows a GitHub pull request page for a repository named 'Takumi1b/void_setup'. The pull request is titled 'add draw #2' and is marked as 'Open'. It shows one commit from the 'feature/#1_add_draw' branch into the 'main' branch. The 'Files changed' tab is selected, showing a single file named 'void_setup.pde'. The code in the file is:

```
1 void setup(){  
2     size(400,400);  
3  
4     float a = 0;  
5  
6     void draw(){  
7         fill (#021050, 5);  
8         rect (0, 0, width, height);  
9         translate (200+a*10,200+a*10);  
10        rotate (a);  
11        scale (a/15);  
12        a+= 0.05;  
13        if (a>30) a= -50;  
14    }  
15}
```

This screenshot shows the same GitHub pull request page as the first one, but with a red box highlighting the code diff in the 'void_setup.pde' file. The code diff highlights the addition of the 'draw()' function and its contents.

プルリクエスト・マージ4/4

- Pull requestタブにあるMerge pull requestを選択
紐づいたissueを終わらせるためにコメントに「close #No」と入力しConfirm mergeを選択

The screenshot shows a GitHub pull request page for a repository named 'Takumi1b/void_setup'. The pull request is titled 'add draw #2' and is currently open. It shows one commit from the 'feature/#1_add_draw' branch being merged into the 'main' branch. The commit message is '#1に紐づく機能の追加'. The pull request has 0 reviews, 0 assignees, and 0 labels. The 'Development' section notes that successfully merging this pull request may close some issues. A red box highlights the 'Merge pull request' button at the bottom.

The screenshot shows the same GitHub pull request page after a comment has been added. The comment text is 'close#1 add draw test ok'. A red box highlights this comment. A modal dialog box is open over the pull request interface, containing the text 'Merge pull request #2 from Takumi1b/feature/#1_add_draw' and two buttons: 'Confirm merge' (highlighted in red) and 'Cancel'. The 'Development' section at the bottom of the page also contains the same merge comment.

コンフリクト対処1/8

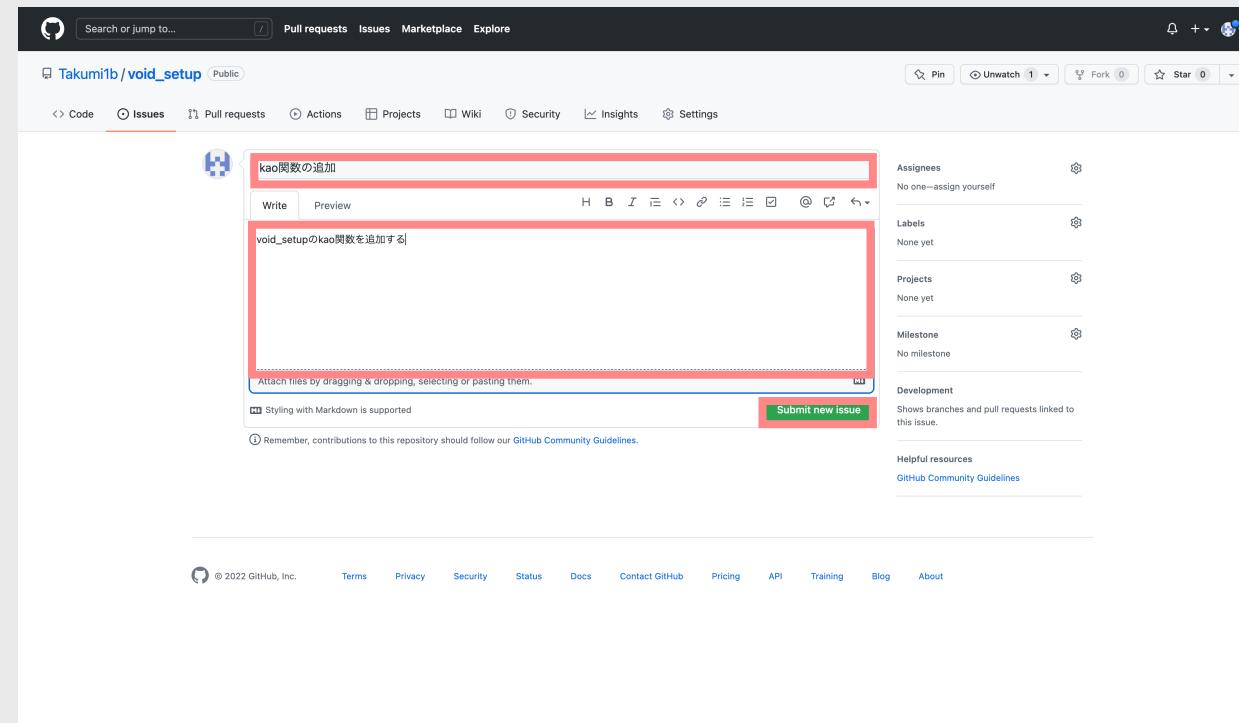
- Gitにおけるコンフリクトとは複数人が同じ箇所を変更した場合に発生するエラー



feature/1とfeature/2の
どちらを変更を優先したらいいの？

コンフリクト対処2/8

- 実際にコンフリクトを起こしてみる
- githubのページからissueを作成
- ブランチを2つ作成 例 \$ git branch feature/#2_add_kao,
\$ git branch feature/#2_add_kao_function



コンフリクト対処3/8

- 作成した1つ目のブランチで作業するためにチェックアウト

例 \$ git checkout feature/#2_add_kao

.pdeファイルに以下の内容を追記

Processing

- 変更した内容を
リモートリポジトリに反映

例

\$ git add void_setup.pde

\$ git commit -m "add kao function"

\$ git push origin feature/#2_add_kao

```
1: void setup(){
2:   size(400,400);
3: }
4:
5: float a = 0;
6:
7: void draw(){
8:   fill (#021050, 5);
9:   rect (0, 0, width, height);
10:  translate (200+a*10,200+a*10);
11:  rotate (a);
12:  scale (a/15);
13:  a+= 0.05;
14:  kao (0,-50);
15:  kao (1,50);
16:  if (a>30) a-= 50;
17: }

18:
19: void kao (int don, float b){
20:   stroke (0);
21:   strokeWeight (2);
22:   fill (#e7eedd);
23:   ellipse (b,0,50,50);
24:   noStroke ();
25:   fill (#68c0c0 + #8f8768 * don);
26:   ellipse (b,0,38,38);
```

コンフリクト対処4/8

- 作成した2つ目のブランチで作業するためにチェックアウト

例 \$ git checkout feature/#2_add_kao_function

.pdeファイルに以下の内容を追記

Processing

- 変更した内容を
リモートリポジトリに反映

例

\$ git add void_setup.pde

\$ git commit -m “add kao function”

\$ git push origin (下に続く)

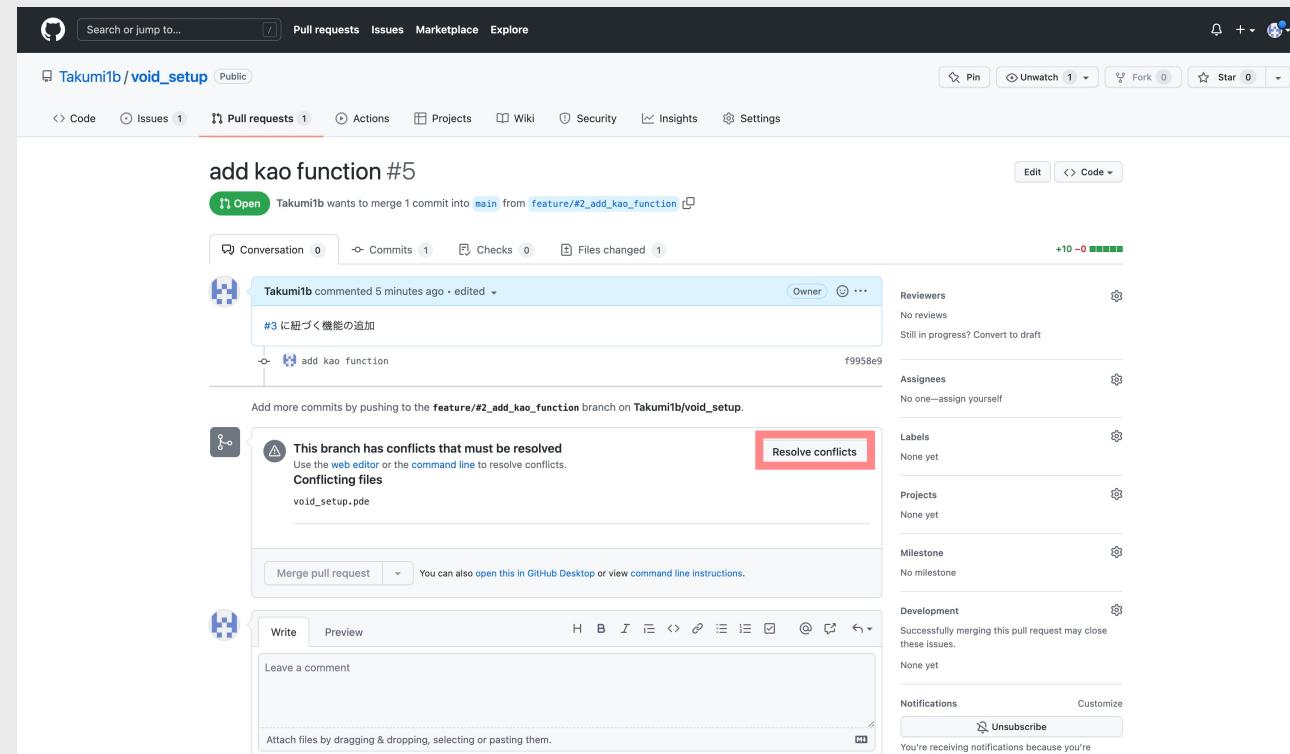
feature/#2_add_kao_function

```
1: void setup(){
2:   size(400,400);
3: }
4:
5: float a = 0;
6:
7: void draw(){
8:   fill (#021050, 5);
9:   rect (0, 0, width, height);
10:  translate (200+a*10,200+a*10);
11:  rotate (a);
12:  scale (a/15);
13:  a+= 0.05;
14:  if (a>30) a-= 50;
15: }

16:
17: stroke (0);
18: strokeWeight (1.4);
19: arc (b + 4.26,5,8.5,7.3,0.3,PI,OPEN);
20: arc (b - 4.26,5,8.5,7.3,0,PI - 0.3,OPEN);
21:
22: fill (0);
23: ellipse (b - 11, -2,6.56,7);
24: ellipse (b + 11, -2,6.56,7);
25: }
```

コンフリクト対応5/8

- ・2つのブランチのプルリクエストを作成
- ・1つ目のブランチをマージ
- ・2つ目のブランチをマージしようとすると以下のようにコンフリクトを直すように促される
- ・Resolve conflictsを選択しコンフリクトを直す



コンフリクト対応6/8

- Resolve conflictsを選択すると以下のようなエディタが開くので黄色くなっているところを直す

add kao function #5

Resolving conflicts between feature/#2_add_kao_function and main and committing changes → feature/#2_add_kao_function

1 conflicting file	void_setup.pde
void_setup.pde	void_setup.pde
	1 void setup(){ 2 size(400,400); 3 } 4 5 float a = 0; 6 7 void draw(){ 8 fill (#021050, 5); 9 rect (0, 0, width, height); 10 translate (200+a*10,200+a*10); 11 rotate (a); 12 scale (a/15); 13 a+= 0.05; 14 kao (0,-50); 15 kao (1,50); 16 if (a>30) a= 50; 17 } 18 19 <<<< feature/#2_add_kao_function 20 stroke (0); 21 strokeWeight (1.4); 22 arc (b + 4.26,5.8.5,7.3,0.3,PI,OPEN); 23 arc (b - 4.26,5.8.5,7.3,0,PI - 0.3,OPEN); 24 25 fill (0); 26 ellipse (b - 11, -2.6.56,7); 27 ellipse (b + 11, -2.6.56,7); 28 } 29 ===== 30 void kao (int don, float b){ 31 stroke (0); 32 strokeWeight (2); 33 fill (#e7edd); 34 ellipse (b,0,50,50); 35 noStroke (); 36 fill (#68c0c0 + #f8768 * don); 37 ellipse (b,0,38,38); 38 >>>> main 39

add kao function #5

Resolving conflicts between feature/#2_add_kao_function and main and committing changes → feature/#2_add_kao_function

1 conflicting file	void_setup.pde
void_setup.pde	void_setup.pde
	1 void setup(){ 2 size(400,400); 3 } 4 5 float a = 0; 6 7 void draw(){ 8 fill (#021050, 5); 9 rect (0, 0, width, height); 10 translate (200+a*10,200+a*10); 11 rotate (a); 12 scale (a/15); 13 a+= 0.05; 14 kao (0,-50); 15 kao (1,50); 16 if (a>30) a= 50; 17 } 18 19 void kao (int don, float b){ 20 stroke (0); 21 strokeWeight (2); 22 fill (#e7edd); 23 ellipse (b,0,50,50); 24 noStroke (); 25 fill (#68c0c0 + #f8768 * don); 26 ellipse (b,0,38,38); 27 28 stroke (0); 29 strokeWeight (1.4); 30 arc (b + 4.26,5.8.5,7.3,0.3,PI,OPEN); 31 arc (b - 4.26,5.8.5,7.3,0,PI - 0.3,OPEN); 32 33 fill (0); 34 ellipse (b - 11, -2.6.56,7); 35 ellipse (b + 11, -2.6.56,7); 36 37 38



コンフリクト対応7/8

- 修正が終わりMark as resolvedを選択するとCommit margeが選択できるようになる
- Commit margeを選択するとプルリクエスト画面でMarge pull requestが選択できる
- Marge pull requestを選択し、必要事項を入力してマージする

add kao function #5
Resolving conflicts between feature/#2_add_kao and main and committing changes → feature/#2_add_kao

1 conflicting file

void_setup.pde	void_setup.pde
----------------	----------------

```
1 void setup(){  
2     size(400,400);  
3 }  
4  
5 float a = 0;  
6  
7 void draw(){  
8     fill (##210150, 5);  
9     rect (0, 0, width, height);  
10    translate (200+a*10,200+a*10);  
11    rotate (a);  
12    scale (a/15);  
13    a+= 0.05;  
14    kao (0,-50);  
15    kao (1,50);  
16    if (a>30) a-= 50;  
17 }  
18  
19 void kao (int don, float b){  
20     stroke (0);  
21     strokeWeight (2);  
22     fill (##e7e0d);  
23     ellipse (b,0,50,50);  
24     noStroke ();  
25     fill (#68c0c0 + #ff7f7f * don);  
26     ellipse (b,0,30,30);  
27  
28     stroke (0);  
29     strokeWeight (1.4);  
30     arc (b + 4.26,5,8.5,7.3,0,PI,OPEN);  
31     arc (b - 4.26,5,8.5,7.3,0,PI - 0.3,OPEN);  
32  
33     fill (0);  
34     ellipse (b - 11, -2,6.56,7);  
35     ellipse (b + 11, -2,6.56,7);  
36 }  
37  
38
```

1 conflict Prev ▲ Next ▼ ⚙️ Mark as resolved

add kao function #5
Resolving conflicts between feature/#2_add_kao and main and committing changes → feature/#2_add_kao

1 conflicting file

void_setup.pde	void_setup.pde
----------------	----------------

✓ Resolved

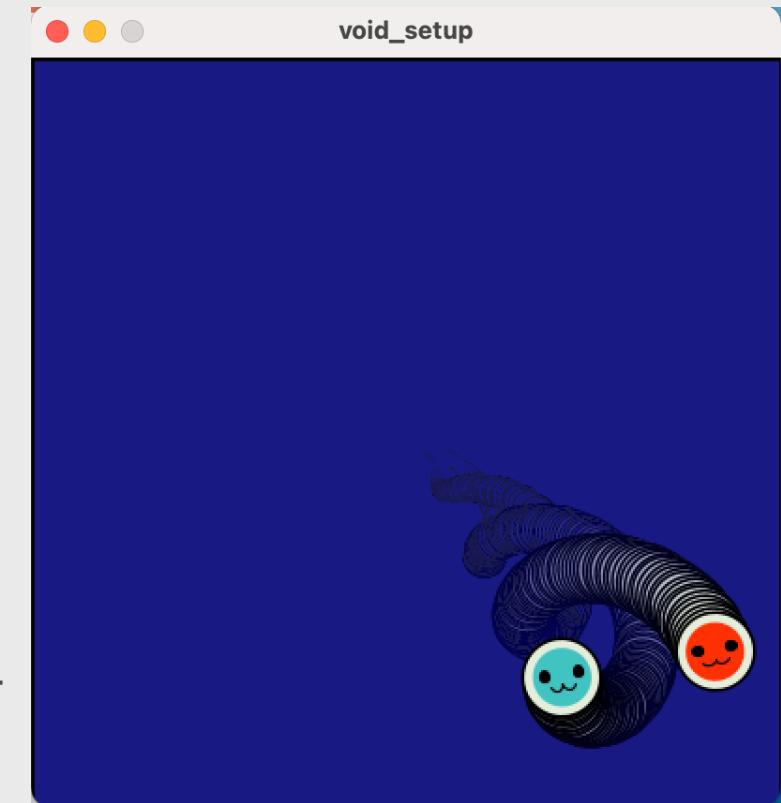
```
1 void setup(){  
2     size(400,400);  
3 }  
4  
5 float a = 0;  
6  
7 void draw(){  
8     fill (##210150, 5);  
9     rect (0, 0, width, height);  
10    translate (200+a*10,200+a*10);  
11    rotate (a);  
12    scale (a/15);  
13    a+= 0.05;  
14    kao (0,-50);  
15    kao (1,50);  
16    if (a>30) a-= 50;  
17 }  
18  
19 void kao (int don, float b){  
20     stroke (0);  
21     strokeWeight (2);  
22     fill (##e7e0d);  
23     ellipse (b,0,50,50);  
24     noStroke ();  
25     fill (#68c0c0 + #ff7f7f * don);  
26     ellipse (b,0,30,30);  
27  
28     stroke (0);  
29     strokeWeight (1.4);  
30     arc (b + 4.26,5,8.5,7.3,0,PI,OPEN);  
31     arc (b - 4.26,5,8.5,7.3,0,PI - 0.3,OPEN);  
32  
33     fill (0);  
34     ellipse (b - 11, -2,6.56,7);  
35     ellipse (b + 11, -2,6.56,7);  
36 }  
37  
38
```

✓ Commit merge

コンフリクト対応8/8

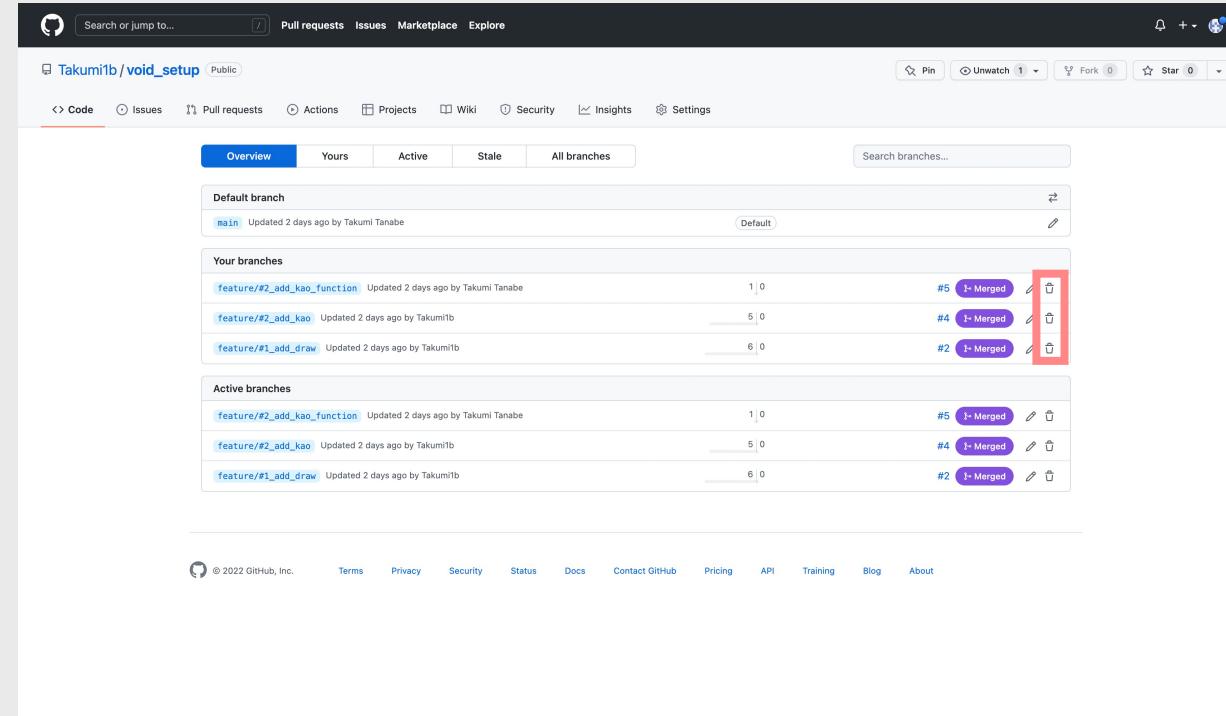
- ・リモートリポジトリのmainリポジトリをローカルに落として動作を確認する
- ・\$ git checkout main とコマンドを入力すると
Your branch is behind 'origin/main' by 7 commits, and can be fast-forwarded.
(use "git pull" to update your local branch)
のようにローカルブランチをアップデートしようと出てくる
- ・\$ git pull でローカルブランチをアップデート
- ・void_setup.pdeを起動して変更内容が
右のように反映されていることを確認する

実行結果：
和田どんと和田かつが
螺旋状にくるくる回って
前後する



ブランチの削除・復元1/3

- ・マージが完了したブランチはコードレビューでOKをもらっているはずなので消しても構わない
- ・ローカルリポジトリ内のブランチの場合 \$ git branch -D 消したいブランチ名とすることで消すことができる
- ・リモートリポジトリにあるブランチの場合、GitHubのbranchタブからブランチを選択して消すことができる



ブランチの削除・復元2/3

- 誤ってブランチを削除してしまった場合でも履歴から復元することができる
- リモートのブランチを消してしまった場合

\$ git branch 復元したいブランチ名 コミットのハッシュ

で復元することができる

コミットのハッシュはGitHubのInsightsタブからNetworkを選択し、復元したいブランチのコミットを選択すると確認できる

The screenshot shows the GitHub Insights interface for a repository named 'Takumi1b/void_setup'. The 'Network' tab is selected. A network graph displays the timeline of commits, with the most recent ones at the top. The graph shows a series of nodes connected by arrows, representing the flow of changes between commits. The commits are ordered by their push date, with the latest commit being from April 17.

The screenshot shows the GitHub commit details for a specific commit. The commit hash is 4feabec. The commit message is "Takumi1b committed 2 days ago". The commit has 1 parent, 9d825c9, and 1 commit, 4feabec. The commit message is "add draw function". The code diff shows the addition of a PDE script named void_setup.pde. The code defines a void setup() function that initializes a canvas and a void draw() function that rotates and scales a shape. The commit has 0 comments.

ブランチの削除・復元3/3

- ローカルブランチを消してしまった場合

```
$ git reflog
```

とコマンドを入力すると以下のようなレスポンスが帰ってくる

```
24321c1 (HEAD -> main, origin/main, origin/HEAD) HEAD@{0}: pull: Fast-forward  
9d825c9 HEAD@{1}: checkout: moving from feature/#2_add_kao_function to main  
f9958e9 HEAD@{2}: commit: add kao function
```

```
·  
·  
·
```

- 先頭のハッシュ値を参照するか、HEAD@{ログ番号}を指定することでブランチを復元できる

```
$ git branch 復元したいブランチ名 先頭のハッシュ値
```

```
$ git branch 復元したいブランチ名 HEAD@{ログ番号}
```

バージョン管理の例とおすすめのサイト

- ・ 永谷研究室におけるGit, GitHubのおすすめの使い方
 - ・ マイコンのプログラムやグラフの可視化プログラムなどをリポジトリに上げる
 - ・ STLファイルなどの3Dデータを修正したらその都度リポジトリに上げる
→ローカル環境だとちょっとした修正が増えるとだんだんわからなくなる
 - ・ 進捗で報告した画像データや動画データをリポジトリに上げる
→昔のデータを見せてと頼まれた場合に有効
- ・ おすすめのサイト
 - ・ サル先生のGit入門 <https://backlog.com/ja/git-tutorial/contents/>
→逆引きGitはやりたいことからコマンドを逆引きできる
 - ・ Git公式 <https://git-scm.com/>
→ここを見れば全てが載っている