



@IT eBookシリーズ Vol.32

こっそり始める Git/GitHub超入門

平屋真吾(クラスメソッド)





こっそり始める Git/GitHub超入門

本連載では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終えるころには、GitやGitHubの基本的な操作が身に付いた状態になっていると思います。

目次

- [01. 初心者でも Windows や Mac ができる、Git のインストールと基本的な使い方](#)
- [02. “はじめの Git”——超基本的な作業フローと 5 つのコマンド](#)
- [03. ポイント嫌いでも分かる Git ブランチの基本——作成、確認、切り替え、master にマージ、削除](#)
- [04. Git でコンフリクトしても慌てるな !! 解消に向けた 3 つの基本作業](#)
- [05. Git コミット現場あるある——やり直し、取り消し、変更したいときに使えるコマンド](#)
- [06. 「soft でも hard でも HEAD とブランチを付けたまま」——git reset で作業の取り消し](#)
- [07. はじまりはいつもプルリク? Git リポジトリホスティングサービス GitHub とは](#)
- [08. 2017 年、GitHub を始めるために最低限知っておきたい各機能](#)
- [09. これでもう怖くない、Git / GitHub におけるリモートリポジトリの作成、確認、変更、更新時の基本 5 コマンド](#)
- [10. GitHub を使うなら最低限知っておきたい、プルリクエストの送り方とレビュー、マージの基本](#)
- [11. 開発者のタスク管理がしやすくなる GitHub Issues の基本的な使い方](#)
- [12. 開発者のスケジュール管理に超便利、GitHub Issues、Label、Milestone、Projects 使いこなし術](#)
- [13. GitHub と Slack の連携の基本&知られざる便利機能 Wiki、Releases、Graphs、Pulse](#)
- [14. たった 3 つで共存できる、Git / GitHub と Subversion \(SVN\) の連携、移行に関する基本操作](#)
- [15. 【図解】git-flow、GitHub Flow を開発現場で使い始めるためにこれだけは覚えておこう](#)

初出：@ IT Test & Tools フォーラム

本書では、@ IT で公開した記事の内容をそのまま収録しています。各記事の日付は、最後に記事を更新した日を表しています。

【注意】

- ・本内容は記事執筆時点（2016年3月～2017年8月）のものです。
- ・本書に記載されている社名・商品名は一般に各社の商標または登録商標です。
- ・その他、免責事項については@ IT Web サイトのポリシーに準拠します。

<http://www.atmarkit.co.jp/aboutus/copyright/copyright.html>

01. 初心者でも Windows や Mac ができる、Git のインストールと基本的な使い方

(2016 年 03 月 31 日)

本連載では、バージョン管理システム「Git」と Git のホスティングサービスの 1 つ「GitHub」を使うために必要な知識を基礎から解説していきます。初回は Git を使い始めるための環境構築です。

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」と Git のホスティングサービスの 1 つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終えるころには、Git や GitHub の基本的な操作が身に付いた状態になっていると思います。

連載第 1 回目の本稿のテーマは「Git を使い始めるための環境構築」です。Git のインストールと動作確認を行う手順について、Windows と Mac (OS X) それぞれの環境に対して解説していきます。ぜひ、この機会に環境構築にチャレンジしてみてください。

Windows 環境での Git のインストール

先に Windows へのインストール手順を解説していきます。筆者が実行した環境は Windows 8.1 です。

Git for Windows 2.7.2 のダウンロード

下記 URL の公式サイトから Git for Windows の最新版をダウンロードします（2016 年 3 月の本稿執筆時点の最新版は、VERSION 2.7.2）。

- <https://git-for-windows.github.io/>

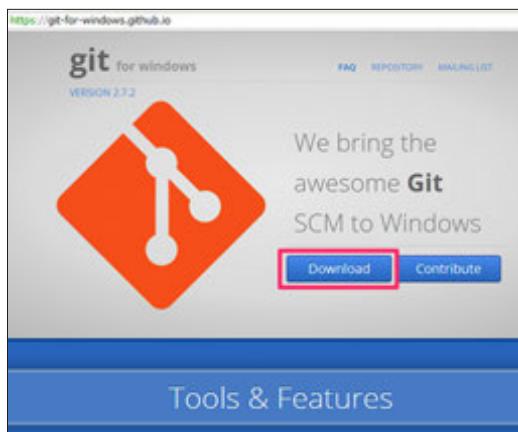


図 1 公式サイトで Git for Windows の最新版をダウンロード

インストーラーの起動からインストール完了まで

ダウンロードが完了したら、ダウンロードしたファイルを実行します。すると、インストールが開始されます。

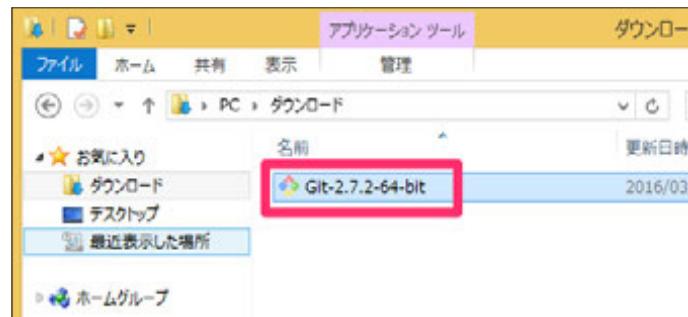


図 2 ダウンロードしたファイルを実行 (Windows)

Windows の設定によっては「ユーザー アカウント制御」のダイアログ表示されます。「はい」をクリックすると、次の操作に進めます。



図 3 「ユーザー アカウント制御」のダイアログ。「はい」をクリック (Windows)

次に、ライセンス画面が表示されます。ライセンスを読んで問題なければ「Next」をクリックします。



図 4 ライセンス確認。読んで問題なければ「Next」をクリック (Windows)

Git のインストール先を聞かれます。今回はデフォルトの設定でインストールします。そのまま「Next」をクリックします。

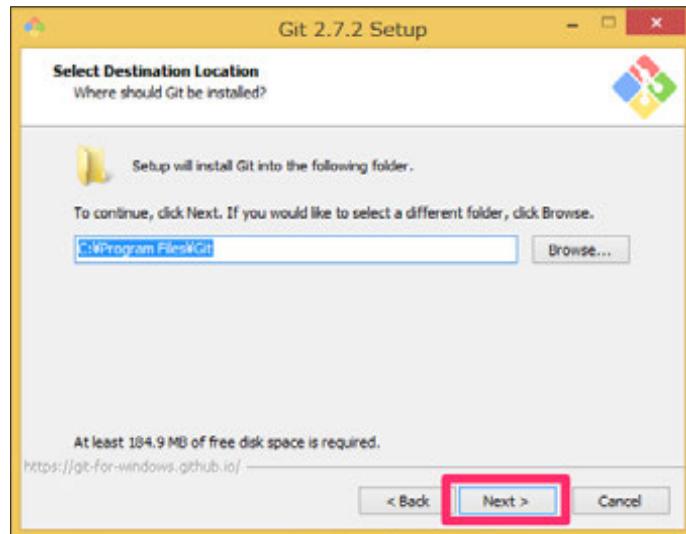


図 5 インストール先選択。そのまま「Next」をクリック（Windows）

インストールするコンポーネントを聞かれます。これもデフォルトの設定でインストールします。そのまま「Next」をクリックします。

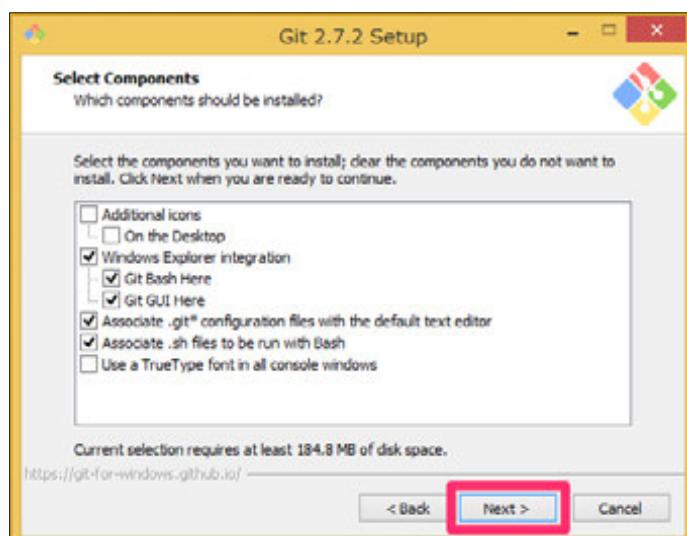


図 6 コンポーネント選択。そのまま「Next」をクリック（Windows）

スタートメニューに追加されるフォルダの名前を聞かれます。これもデフォルトの設定でインストールします。そのまま「Next」をクリックします。

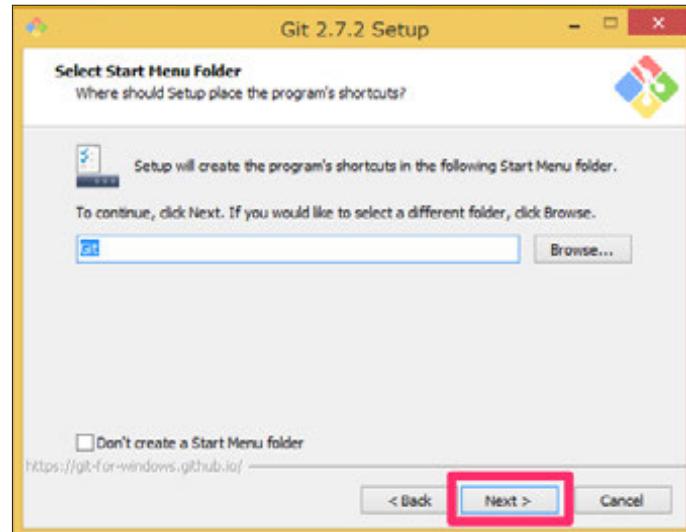


図 7 スタートメニューの設定。そのまま「Next」をクリック (Windows)

PATH 環境変数について聞かれます。デフォルトの設定でインストールします。そのまま「Next」をクリックします。

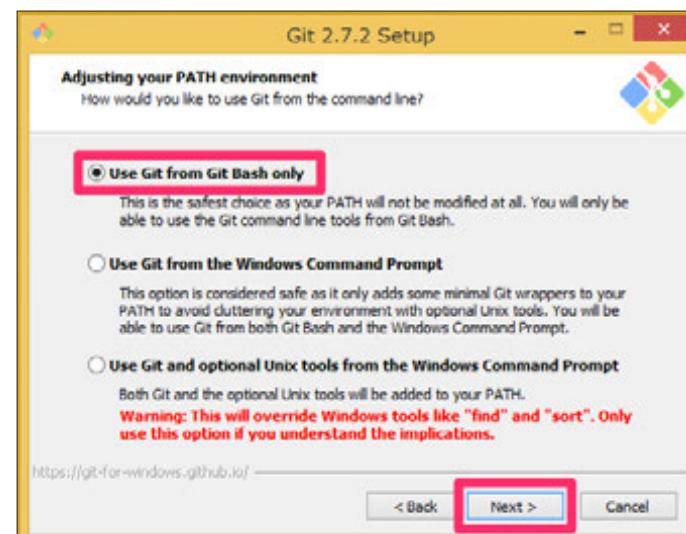


図 8 PATH 環境変数の設定。そのまま「Next」をクリック (Windows)

改行コードの設定について聞かれます。「Checkout as-is, commit as-is」にチェックを入れ、「Next」をクリックします。

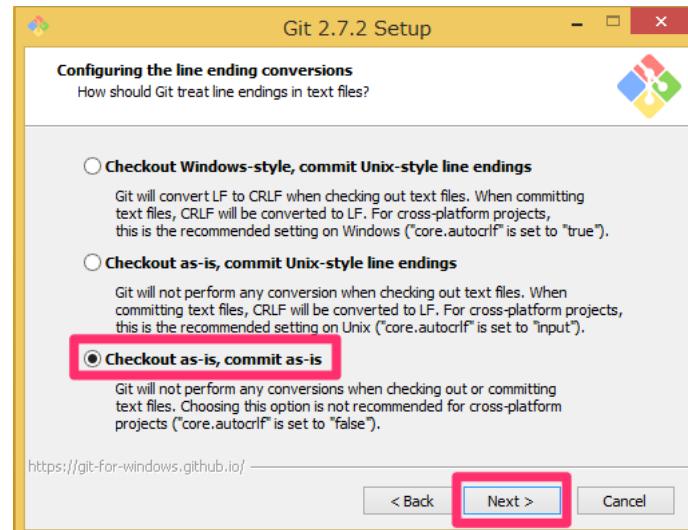


図9 改行コードの設定。「Checkout as-is, commit as-is」にチェックを入れ、「Next」をクリック（Windows）

ターミナルの設定について聞かれます。デフォルトの設定でインストールします。そのまま「Next」をクリックします。

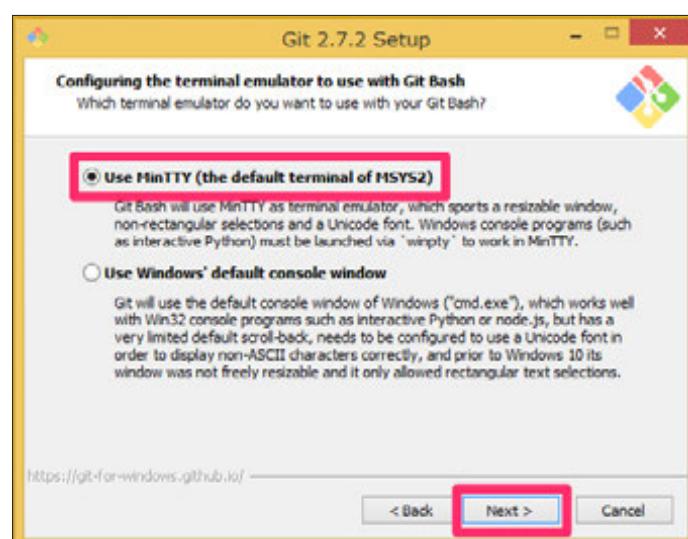


図10 ターミナルの設定。そのまま「Next」をクリック（Windows）

キャッシュの設定について聞かれます。デフォルトの設定でインストールします。そのまま「Next」をクリックします。

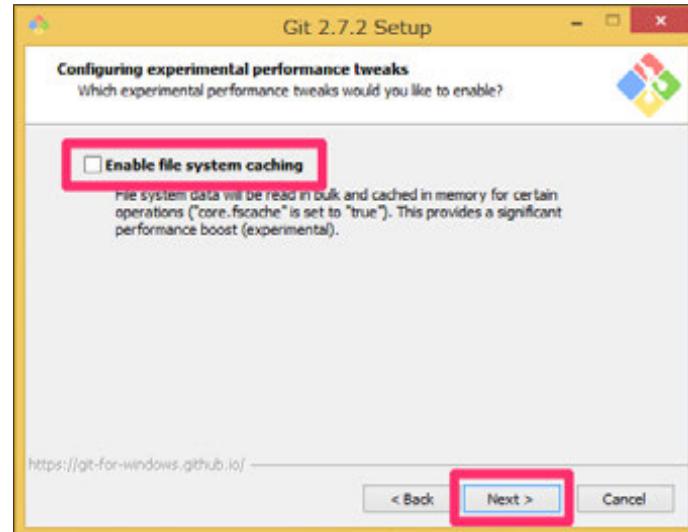


図 11 キャッシュの設定。そのまま「Next」をクリック（Windows）

インストールが始まります。完了するまで待ちます。

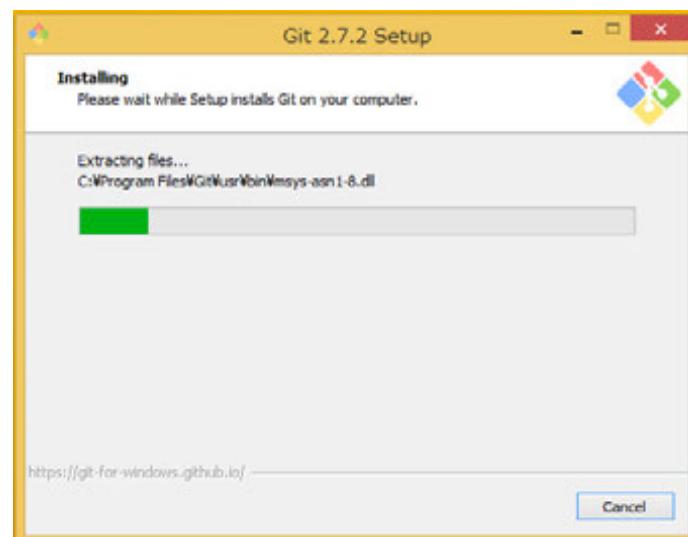


図 12 インストール処理（Windows）

「Finish」ボタンをクリックします。



図 13 インストール完了（Windows）

Windows 環境での Git の初期設定

初期設定を行いましょう。

Git のバージョンを確認するには

スタートメニューのアプリ一覧を開くと「Git」項目が追加されています。「Git Bash」をクリックします。

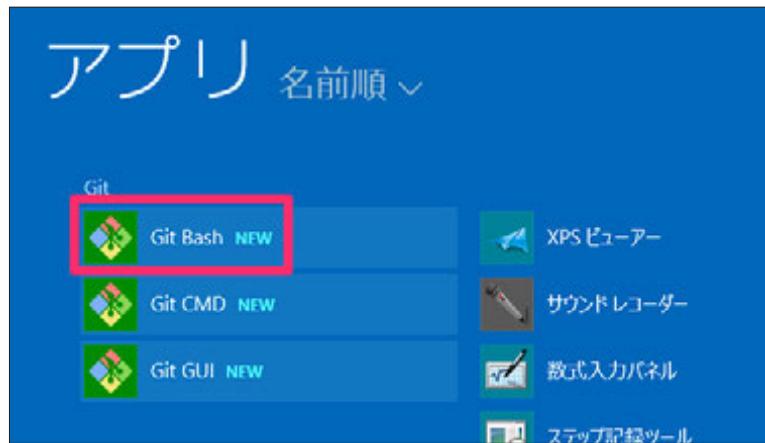


図 14 アプリ一覧。「Git Bash」をクリック (Windows)

Git 専用の bash コンソールが表示されます。



図 15 Git Bash (Windows)

Git が導入されたかを確認するために、Git のバージョンを表示してみましょう。

「git --version」と入力し、「Enter」キーを押します。以下の画像のように Git のバージョンが表示されれば OK です。

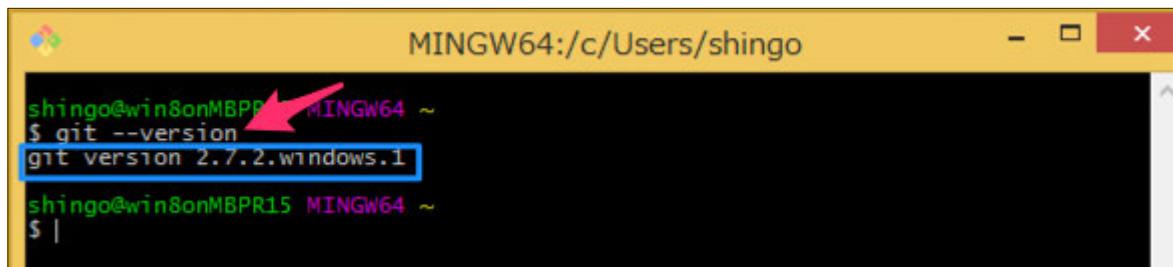


図 16 「git --version」と入力して Git のバージョンを表示 (Windows)

ユーザー名とメールアドレスを設定する

続いて、必要最低限の設定を行います。Git で使用する「ユーザー名」と「メールアドレス」を設定します。

「git config --global user.name " {ユーザー名} "」と入力し、「Enter」キーを押します。



```
shingo@win8onMBPR15 MINGW64 ~
$ git --version
git version 2.7.2.windows.1

shingo@win8onMBPR15 MINGW64 ~
$ git config --global user.name "Shingo Hiraya" ←

shingo@win8onMBPR15 MINGW64 ~
$ |
```

図 17 「git config --global user.name " {ユーザー名} "」と入力し、ユーザー名を設定 (Windows)

続いて、「git config --global user.email {メールアドレス}」と入力し、「Enter」キーを押します。



```
shingo@win8onMBPR15 MINGW64 ~
$ git --version
git version 2.7.2.windows.1

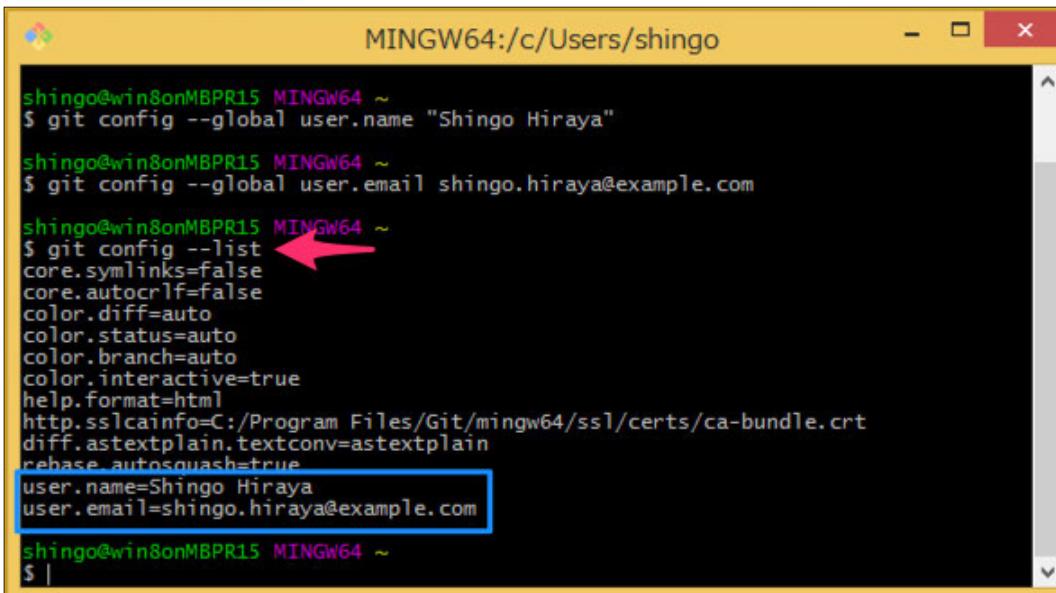
shingo@win8onMBPR15 MINGW64 ~
$ git config --global user.name "Shingo Hiraya"

shingo@win8onMBPR15 MINGW64 ~
$ git config --global user.email shingo.hiraya@example.com ←

shingo@win8onMBPR15 MINGW64 ~
$ |
```

図 18 「git config --global user.email {メールアドレス}」と入力し、メールアドレスを設定 (Windows)

設定した値を確認するには、「git config --list」と入力し、「Enter」キーを押します。



```
shingo@win8onMBPR15 MINGW64 ~
$ git config --global user.name "Shingo Hiraya"

shingo@win8onMBPR15 MINGW64 ~
$ git config --global user.email shingo.hiraya@example.com

shingo@win8onMBPR15 MINGW64 ~
$ git config --list ←
core.symlinks=false
core.autocrlf=false
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
diff.astextplain.textconv=astextplain
rebase.autosquash=true
user.name=Shingo Hiraya
user.email=shingo.hiraya@example.com

shingo@win8onMBPR15 MINGW64 ~
$ |
```

図 19 「git config --list」と入力し、設定した値を確認 (Windows)

「user.name」と「user.email」の行に設定した値が適用されていれば OK です。

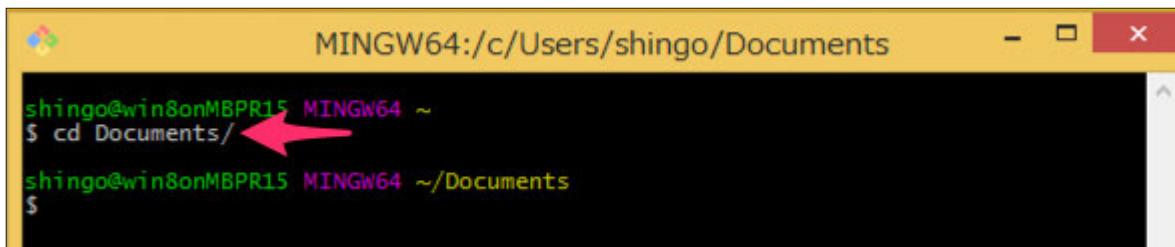
動作確認の準備（Windows）

Git を使う準備が整いましたので、基本的なコマンドを試してみましょう。

「Documents」フォルダ内に「hello-git」フォルダを作り、このフォルダ内のファイルを Git で管理してみます。

「hello-git」フォルダを作成

「cd Documents/」と入力し、「Enter」キーを押して「Documents」フォルダに移動します。



```
MINGW64:/c/Users/shingo/Documents
shingo@win8onMBPR15 MINGW64 ~
$ cd Documents/
shingo@win8onMBPR15 MINGW64 ~/Documents
```

図 20 「cd Documents/」と入力し、「Documents」フォルダに移動（Windows）

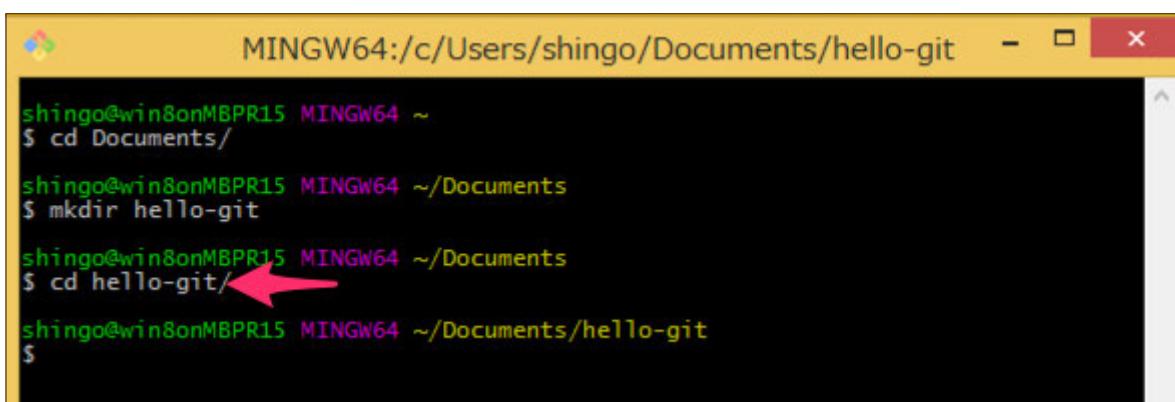
「mkdir hello-git」と入力し、「Enter」キーを押して「hello-git」フォルダを作成します。



```
MINGW64:/c/Users/shingo/Documents
shingo@win8onMBPR15 MINGW64 ~
$ cd Documents/
shingo@win8onMBPR15 MINGW64 ~/Documents
$ mkdir hello-git
shingo@win8onMBPR15 MINGW64 ~/Documents
```

図 21 「mkdir hello-git」と入力し、「hello-git」フォルダを作成（Windows）

「cd hello-git/」と入力し、「Enter」キーを押して「hello-git」フォルダに移動します。



```
MINGW64:/c/Users/shingo/Documents/hello-git
shingo@win8onMBPR15 MINGW64 ~
$ cd Documents/
shingo@win8onMBPR15 MINGW64 ~/Documents
$ mkdir hello-git
shingo@win8onMBPR15 MINGW64 ~/Documents
$ cd hello-git/
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git
```

図 22 「cd hello-git/」と入力し、「hello-git」フォルダに移動（Windows）

「hello.txt」ファイルを作成

「touch hello.txt」と入力し、「Enter」キーを押して「hello.txt」ファイルを作成します。



```
MINGW64:/c/Users/shingo/Documents/hello-git
$ touch hello.txt
```

A screenshot of a Windows terminal window titled "MINGW64:/c/Users/shingo/Documents/hello-git". The command "\$ touch hello.txt" is highlighted with a red arrow pointing to the "touch" command.

図 23 「touch hello.txt」と入力し、「hello.txt」ファイルを作成（Windows）

「ls」と入力し、「Enter」キーを押して「hello.txt」ファイルが作成されていることを確認します。



```
MINGW64:/c/Users/shingo/Documents/hello-git
$ ls
```

A screenshot of a Windows terminal window titled "MINGW64:/c/Users/shingo/Documents/hello-git". The command "\$ ls" is highlighted with a red arrow pointing to the "ls" command. The output shows a single file named "hello.txt" in blue.

図 24 「ls」と入力し、ファイル一覧を表示（Windows）

以上で準備完了です。

幾つかの基本コマンドを使い、動作確認（Windows）

幾つかの基本コマンドを使い、動作確認をしてみましょう。「リポジトリ」「コミット」という単語が出てきますが、Git を使う上での“基本”用語です。知らない方は、記事「[ガチで 5 分で分かる分散型バージョン管理システム Git](#)」を参照してください。

「git init」でGitの「リポジトリ」を初期化

「git init」コマンドを使用して、Gitの「リポジトリ」を初期化します。「git init」と入力し、「Enter」キーを押します。

The screenshot shows a terminal window titled 'MINGW64:/c/Users/shingo/Documents/hello-git'. The user has navigated to their 'Documents' folder and created a new directory named 'hello-git'. Inside this directory, they have created a file named 'hello.txt'. The user then runs the command '\$ git init', which initializes an empty Git repository in the current directory. A red arrow points to this command. The output of the command is visible below it.

```
shingo@win8onMBPR15 MINGW64 ~
$ cd Documents/
shingo@win8onMBPR15 MINGW64 ~/Documents
$ mkdir hello-git
shingo@win8onMBPR15 MINGW64 ~/Documents
$ cd hello-git/
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git
$ touch hello.txt
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git
$ ls
hello.txt
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git
$ git init
Initialized empty Git repository in C:/Users/shingo/Documents/hello-git/.git/
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ |
```

図 25 「git init」と入力し、Gitの「リポジトリ」を初期化（Windows）

「hello-git」フォルダ内に「リポジトリ」が作成されました。

「git status」でステータスを確認

ここで「git status」コマンドを使用して、ステータスを確認してみましょう。「git status」と入力し、「Enter」キーを押します。

The screenshot shows the same terminal window as the previous one, but now the user has run the command '\$ git status'. A red arrow points to this command. The output shows that there is an untracked file named 'hello.txt'. A blue box highlights this message. The terminal also indicates that there is nothing added to commit but untracked files present, and suggests using 'git add' to track them.

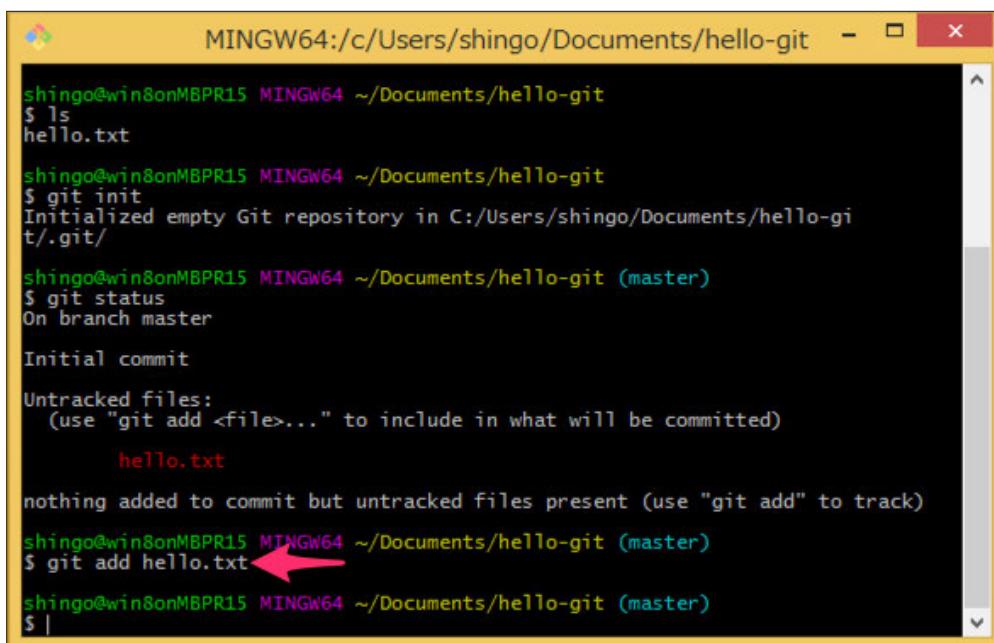
```
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git
$ touch hello.txt
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git
$ ls
hello.txt
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git
$ git init
Initialized empty Git repository in C:/Users/shingo/Documents/hello-git/.git/
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ git status
On branch master
Initial commit
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hello.txt
nothing added to commit but untracked files present (use "git add" to track)
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ |
```

図 26 「git status」と入力し、Gitの「ステータス」を確認（Windows）

「hello.txt ファイルは追跡されてないよ」という旨のメッセージが表示されます。

「git add」でファイルの追跡を開始

「git add」コマンドを使用して、hello.txt ファイルの追跡を開始します。「git add hello.txt」と入力し、「Enter」キーを押します。



```
MINGW64:/c/Users/shingo/Documents/hello-git
$ ls
hello.txt

shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git
$ git init
Initialized empty Git repository in C:/Users/shingo/Documents/hello-git/.git/
$ git status
On branch master

Initial commit

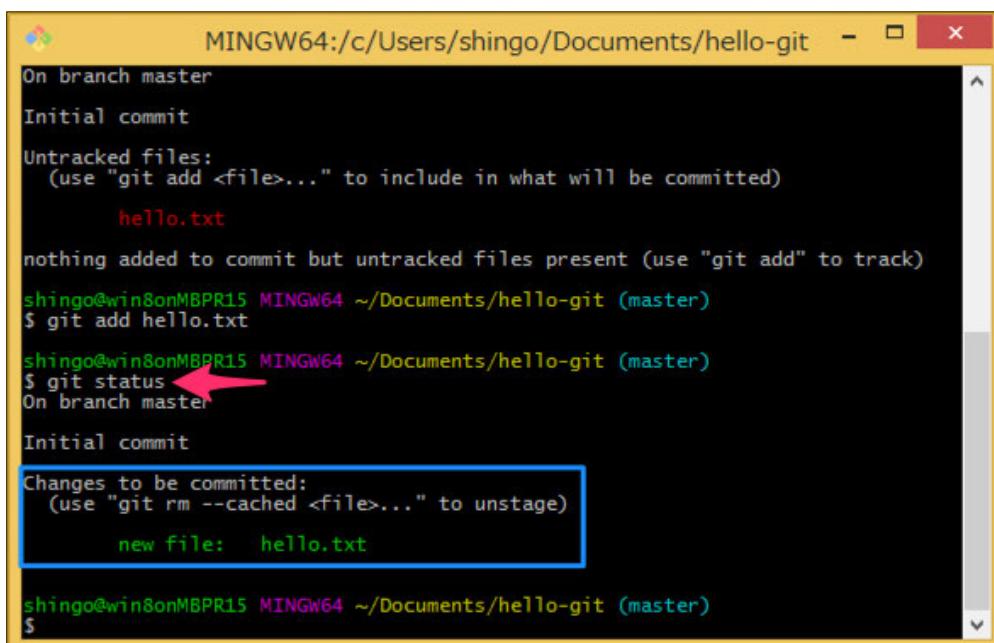
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    hello.txt

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

shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ git add hello.txt
$ git status
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ |
```

図 27 「git add hello.txt」と入力し、hello.txt ファイルの追跡を開始（Windows）

再び「git status」コマンドを使用して、ステータスを確認してみましょう。「git status」と入力し、「Enter」キーを押します。



```
MINGW64:/c/Users/shingo/Documents/hello-git
$ On branch master
$ Initial commit
$ Untracked files:
$   (use "git add <file>..." to include in what will be committed)
$     hello.txt
$ nothing added to commit but untracked files present (use "git add" to track)

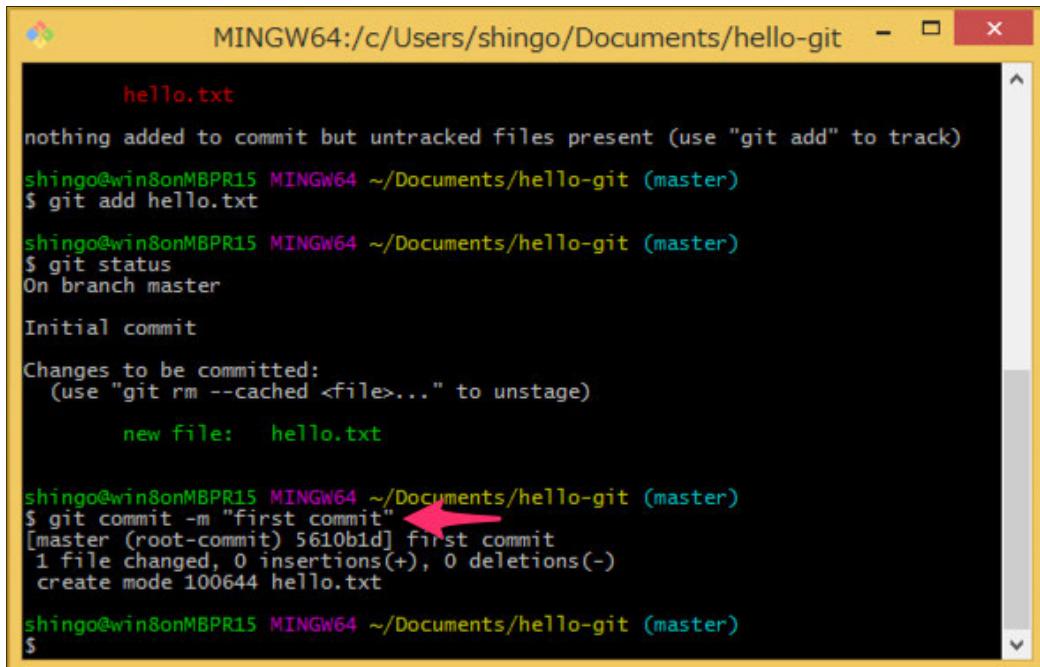
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ git add hello.txt
$ git status
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ |
```

図 28 「git status」と入力し、Git の「ステータス」を確認（Windows）

hello.txt ファイルがコミットできる状態になりました。

「git commit」で変更内容を「リポジトリ」にコミット

「git commit」コマンドを使用して、変更内容を「リポジトリ」にコミットします。「git commit -m "first commit"」と入力し、「Enter」キーを押します。



```

MINGW64:/c/Users/shingo/Documents/hello-git
$ hello.txt
nothing added to commit but untracked files present (use "git add" to track)
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ git add hello.txt

shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ git status
On branch master

Initial commit

Changes to be committed:
(use "git rm --cached <file>..." to unstage)

    new file:   hello.txt

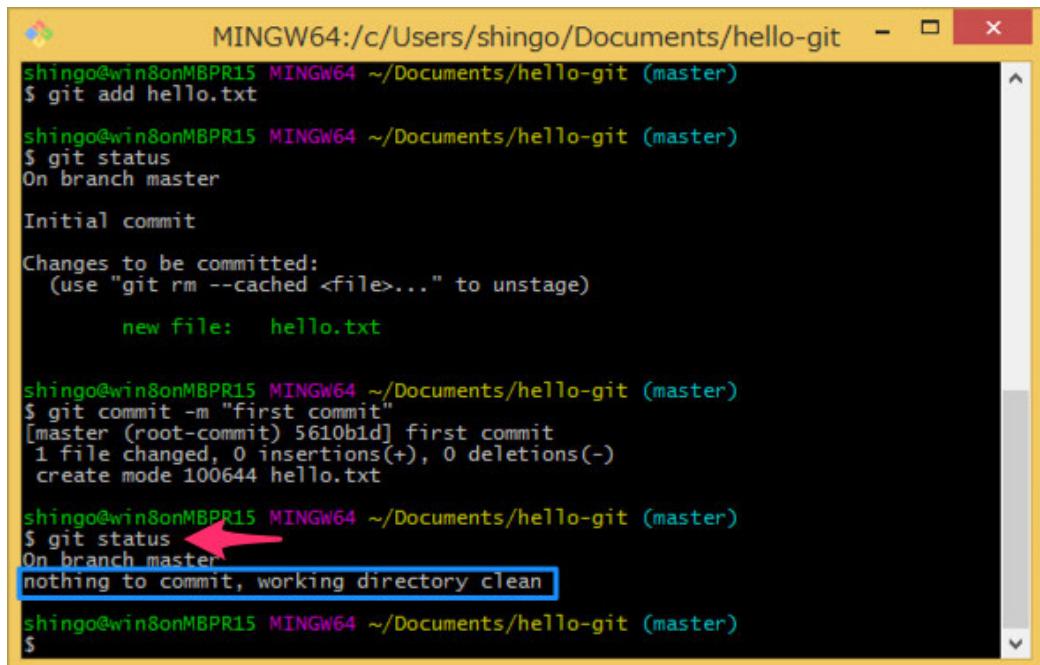
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ git commit -m "first commit"
[master (root-commit) 5610b1d] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 hello.txt

shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ 
```

図 29 「git commit -m "first commit"」と入力し、変更内容を「リポジトリ」にコミット（Windows）

hello.txt ファイルがコミットされました。

「git status」コマンドを使用して、ステータスを確認してみましょう。



```

MINGW64:/c/Users/shingo/Documents/hello-git
$ git add hello.txt

shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ git status
On branch master

Initial commit

Changes to be committed:
(use "git rm --cached <file>..." to unstage)

    new file:   hello.txt

shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ git commit -m "first commit"
[master (root-commit) 5610b1d] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 hello.txt

shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ git status
On branch master
nothing to commit, working directory clean

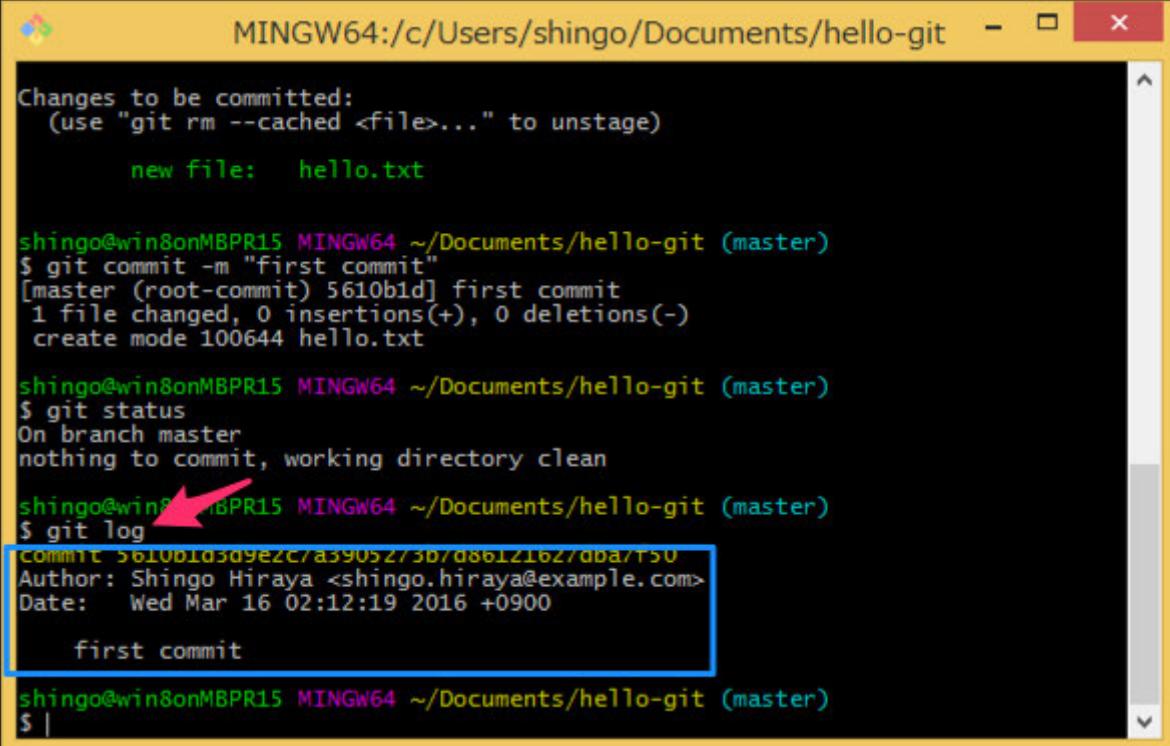
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)
$ 
```

図 30 「git status」と入力し、Git の「ステータス」を確認（Windows）

「作業スペースがクリーンになった」という旨のメッセージが表示されます。

「git log」でコミットの履歴を確認

最後に「git log」コマンドを使用して、コミットの履歴を確認してみましょう。「git log」と入力し、「Enter」キーを押します。



The screenshot shows a terminal window titled 'MINGW64:/c/Users/shingo/Documents/hello-git'. The command history is as follows:

```
Changes to be committed:  
(use "git rm --cached <file>..." to unstage)  
    new file:   hello.txt  
  
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)  
$ git commit -m "first commit"  
[master (root-commit) 5610b1d] first commit  
1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 hello.txt  
  
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)  
$ git status  
On branch master  
nothing to commit, working directory clean  
  
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)  
$ git log  
commit 5610b1d3d9e2c/a39052730/d86121627/0ba/150  
Author: Shingo Hiraya <shingo.hiraya@example.com>  
Date:   Wed Mar 16 02:12:19 2016 +0900  
  
    first commit  
  
shingo@win8onMBPR15 MINGW64 ~/Documents/hello-git (master)  
$ |
```

A red arrow points to the command 'git log' in the history.

図 31 「git log」と入力し、コミットの履歴を確認（Windows）

以下の3つの情報が表示されます。

- コミットしたユーザーの情報
- 日付
- コミットメッセージ

「コミットしたユーザーの情報」の名前とメールアドレスは、「初期設定」で設定した情報です。

Mac 環境での Git のインストール

Macへのインストール手順を解説していきます。筆者が実行した環境はOS X El Capitan、バージョンは10.11.3(15D21)です。

Git がインストールされているかどうかを確認する方法

OS X には Git が付属しているので、そのままでも使えるはずですが、インストールされていない場合は、インストールします。

Git をインストール済みかどうか分からぬ場合は、以下の操作を実行します。

まず、「アプリケーション」→「ユーティリティ」の中にある「ターミナル」アプリを起動します。

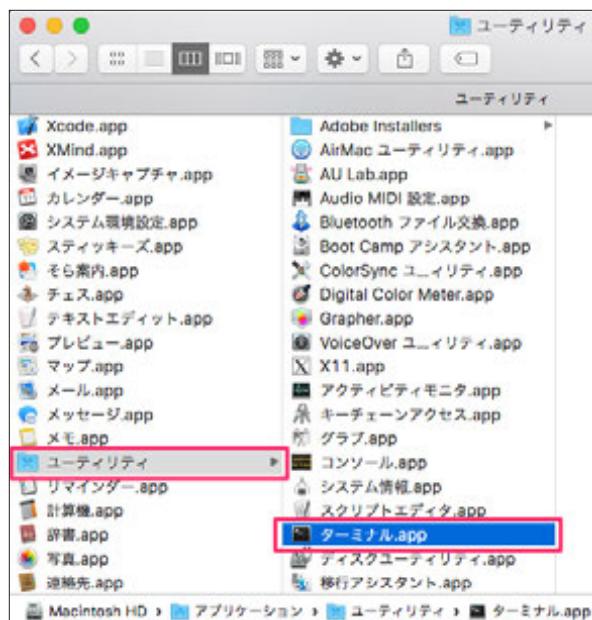


図 32 「アプリケーション」→「ユーティリティ」の中にある「ターミナル」アプリを起動 (Mac)

「git --version」と入力し、「Enter」キーを押します。

```
hirayashingo — -bash — 80x24
Last login: Wed Mar 16 23:32:11 on ttys001
MBPR15-2:~ hirayashingo$ git --version
```

図 33 「git --version」と入力し、Git のバージョンを表示 (Mac)

以下のように、Git のバージョン番号が表示された場合は、すぐに Git を使うことができます。「初期設定」の章へ進んでください。

```
hirayashingo — -bash — 80x24
Last login: Wed Mar 16 23:32:11 on ttys001
MBPR15-2:~ hirayashingo$ git --version
git version 2.5.4 (Apple Git-61)
MBPR15-2:~ hirayashingo$
```

図 34 Git のバージョンが表示された様子 (Mac)

ダイアログが表示された場合→ Command Line Tools のインストール

以下のように「"git" コマンドを実行するには、コマンドライン・デベロッパ・ツールが必要です。」と表示された場合は、Git をインストールする必要があります。

Git をインストールする方法は幾つかありますが、今回は「Command Line Tools」をインストールすることで Git をインストールする方法を紹介します。「"git" コマンドを実行するには ...」ダイアログの「インストール」をクリックします。



図 35 Command Line Tools のインストール確認ダイアログ。「インストール」をクリック (Mac)

使用許諾契約画面が表示されます。使用許諾契約を読んで問題なければ「同意する」をクリックします。



図 36 Command Line Tools の使用許諾契約。「同意する」をクリック (Mac)

ダウンロードとインストールが実行されます。

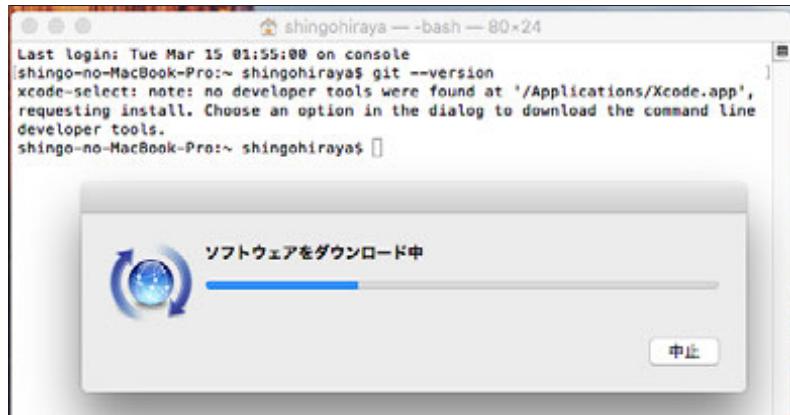


図 37 Command Line Tools のインストール処理 (Mac)

インストールが完了したら、「完了」をクリックします。

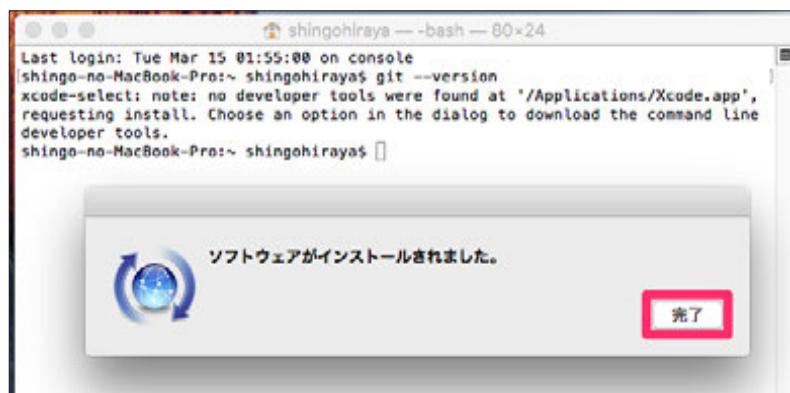


図 38 Command Line Tools のインストール完了。「完了」をクリック (Mac)

ターミナルで再度「git --version」と入力し、「Enter」キーを押してみましょう。バージョンが表示されるはずです。

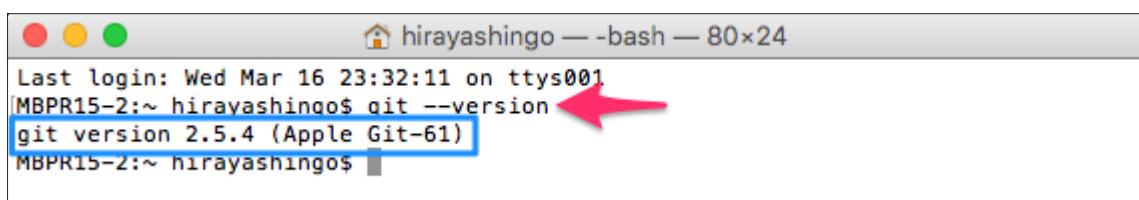


図 39 ターミナルで再度「git --version」と入力し、Git のバージョンを確認 (Mac)

2016 年 3 月時点最新バージョンの 2.7 系を使いたい場合は、以下のページを参考にアップデートしてみてください。

- Mac の HomeBrew で Git を 2.7.0 にアップデートしよう - Qiita

Mac 環境での Git の初期設定

初期設定を行いましょう。

ユーザー名とメールアドレスを設定する

Git で使用する「ユーザー名」と「メールアドレス」を設定します。

「git config --global user.name " {ユーザー名} "」と入力し、「Enter」キーを押します。続いて、「git config --global user.email {メールアドレス}」と入力し、「Enter」キーを押します。

```
Last login: Wed Mar 16 23:32:11 on ttys001
[MBPR15-2:~ hirayashingo$ git --version
git version 2.5.4 (Apple Git-61)
[MBPR15-2:~ hirayashingo$ git config --global user.name "Shingo Hiraya"
[MBPR15-2:~ hirayashingo$ git config --global user.email shingo.hiraya@example.co
m
[MBPR15-2:~ hirayashingo$
```

図 40 「git config --global user.name " {ユーザー名} "」「git config --global user.email {メールアドレス}」と入力し、ユーザー名とメールアドレスを設定 (Mac)

設定した値を確認するには、「git config --list」と入力し、「Enter」キーを押します。

```
Last login: Wed Mar 16 23:32:11 on ttys001
[MBPR15-2:~ hirayashingo$ git --version
git version 2.5.4 (Apple Git-61)
[MBPR15-2:~ hirayashingo$ git config --global user.name "Shingo Hiraya"
[MBPR15-2:~ hirayashingo$ git config --global user.email shingo.hiraya@example.co
m
[MBPR15-2:~ hirayashingo$ git config --list ←
user.name=Shingo Hiraya
user.email=shingo.hiraya@example.com
core.excludesfile=/Users/hirayashingo/.gitignore_global
[MBPR15-2:~ hirayashingo$
```

図 42 「git config --list」と入力し、設定した値を確認 (Mac)

「user.name」「user.email」の行に設定した値が適用されていれば OK です。

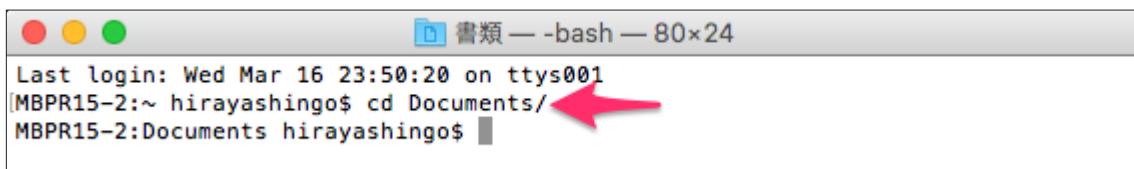
動作確認の準備 (Mac)

Git を使う準備が整いましたので、基本的なコマンドを試してみましょう。

「Documents」フォルダ内に「hello-git」フォルダを作り、このフォルダ内のファイルを Git で管理してみます。

「hello-git」フォルダを作成

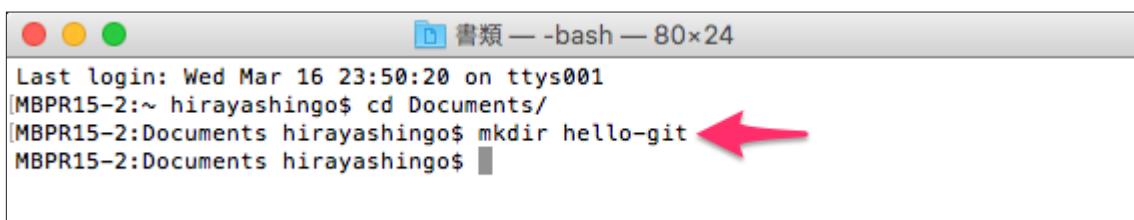
「cd Documents/」と入力し、「Enter」キーを押して「Documents」フォルダに移動します。



```
Last login: Wed Mar 16 23:50:20 on ttys001
[MBPR15-2:~ hirayashingo$ cd Documents/
MBPR15-2:Documents hirayashingo$ ]
```

図 42 「cd Documents/」と入力し、「Documents」フォルダに移動 (Mac)

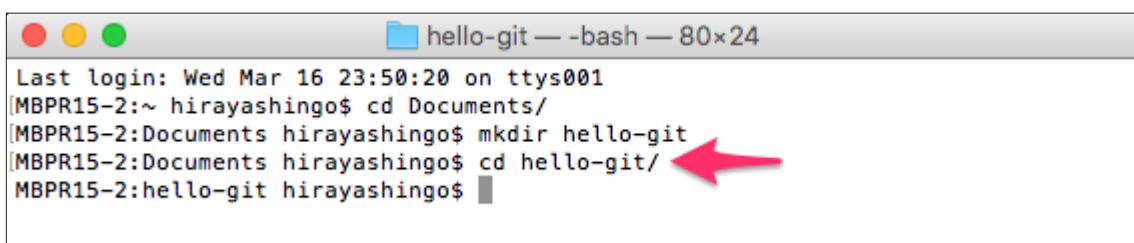
「mkdir hello-git」と入力し、「hello-git」フォルダを作成します。



```
Last login: Wed Mar 16 23:50:20 on ttys001
[MBPR15-2:~ hirayashingo$ cd Documents/
[MBPR15-2:Documents hirayashingo$ mkdir hello-git <-- This command is highlighted with a red arrow.
MBPR15-2:Documents hirayashingo$ ]
```

図 43 「mkdir hello-git」と入力し、「hello-git」フォルダを作成 (Mac)

「cd hello-git/」と入力し、「Enter」キーを押して「hello-git」フォルダに移動します。

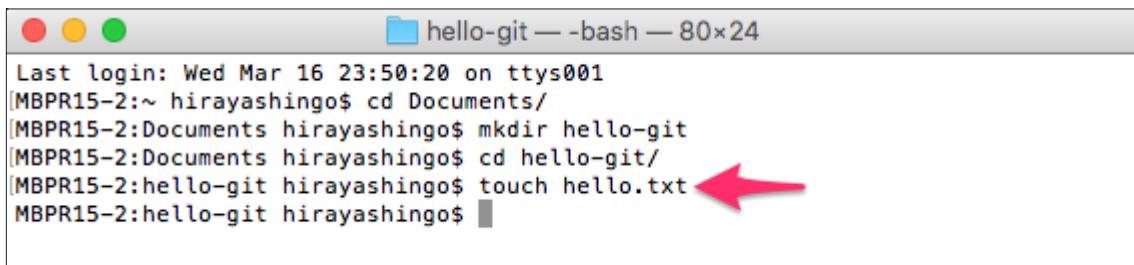


```
Last login: Wed Mar 16 23:50:20 on ttys001
[MBPR15-2:~ hirayashingo$ cd Documents/
[MBPR15-2:Documents hirayashingo$ mkdir hello-git
[MBPR15-2:Documents hirayashingo$ cd hello-git/ <-- This command is highlighted with a red arrow.
MBPR15-2:hello-git hirayashingo$ ]
```

図 44 「cd hello-git/」と入力し、「hello-git」フォルダに移動 (Mac)

「hello.txt」ファイルを作成

「touch hello.txt」と入力し、「Enter」キーを押して「hello.txt」ファイルを作成します。



```
Last login: Wed Mar 16 23:50:20 on ttys001
[MBPR15-2:~ hirayashingo$ cd Documents/
[MBPR15-2:Documents hirayashingo$ mkdir hello-git
[MBPR15-2:Documents hirayashingo$ cd hello-git/
[MBPR15-2:hello-git hirayashingo$ touch hello.txt <-- This command is highlighted with a red arrow.
MBPR15-2:hello-git hirayashingo$ ]
```

図 45 「touch hello.txt」と入力し、「hello.txt」ファイルを作成 (Mac)

「ls」と入力し、「Enter」キーを押して「hello.txt」ファイルが作成されていることを確認します。

```
Last login: Wed Mar 16 23:50:20 on ttys001
[MBPR15-2:~ hirayashingo$ cd Documents/
[MBPR15-2:Documents hirayashingo$ mkdir hello-git
[MBPR15-2:Documents hirayashingo$ cd hello-git/
[MBPR15-2:hello-git hirayashingo$ touch hello.txt
[MBPR15-2:hello-git hirayashingo$ ls
hello.txt
[MBPR15-2:hello-git hirayashingo$ ]
```

図 46 「ls」と入力し、ファイル一覧を表示（Mac）

以上で準備完了です。

幾つかの基本コマンドを使い、動作確認（Mac）

幾つかの基本コマンドを使い、動作確認をしてみましょう。「リポジトリ」「コミット」という単語が出てきますが、Git を使う上での“基本”用語です。知らない方は、記事「[ガチで 5 分で分かる分散型バージョン管理システム Git](#)」を参照してください。

「git init」で Git の「リポジトリ」を初期化

「git init」コマンドを使用して、Git の「リポジトリ」を初期化します。「git init」と入力し、「Enter」キーを押します。

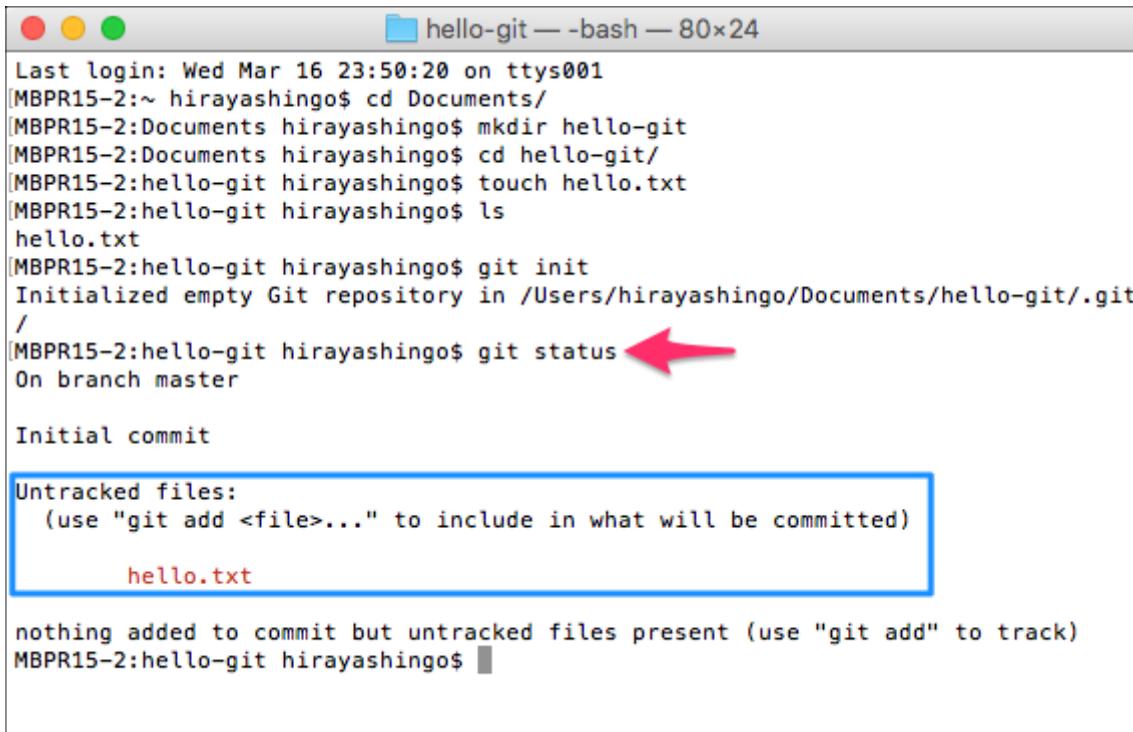
```
Last login: Wed Mar 16 23:50:20 on ttys001
[MBPR15-2:~ hirayashingo$ cd Documents/
[MBPR15-2:Documents hirayashingo$ mkdir hello-git
[MBPR15-2:Documents hirayashingo$ cd hello-git/
[MBPR15-2:hello-git hirayashingo$ touch hello.txt
[MBPR15-2:hello-git hirayashingo$ ls
hello.txt
[MBPR15-2:hello-git hirayashingo$ git init
Initialized empty Git repository in /Users/hirayashingo/Documents/hello-git/.git/
[MBPR15-2:hello-git hirayashingo$ ]
```

図 47 「git init」と入力し、Git の「リポジトリ」を初期化（Mac）

「hello-git」フォルダ内に「リポジトリ」が作成されました。

「git status」でステータスを確認

ここで「git status」コマンドを使用して、ステータスを確認してみましょう。「git status」と入力し、「Enter」キーを押します。



```
Last login: Wed Mar 16 23:50:20 on ttys001
[MBPR15-2:~ hirayashingo$ cd Documents/
[MBPR15-2:Documents hirayashingo$ mkdir hello-git
[MBPR15-2:Documents hirayashingo$ cd hello-git/
[MBPR15-2:hello-git hirayashingo$ touch hello.txt
[MBPR15-2:hello-git hirayashingo$ ls
hello.txt
[MBPR15-2:hello-git hirayashingo$ git init
Initialized empty Git repository in /Users/hirayashingo/Documents/hello-git/.git
/
[MBPR15-2:hello-git hirayashingo$ git status ←
On branch master

Initial commit

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

    hello.txt

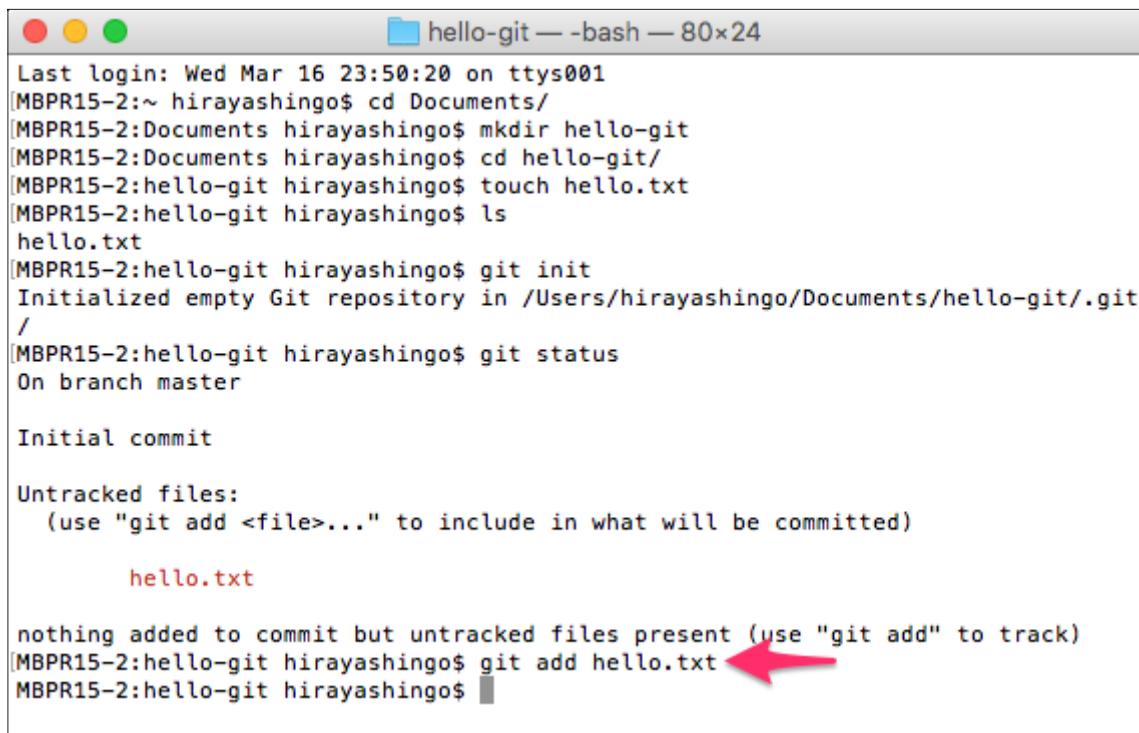
nothing added to commit but untracked files present (use "git add" to track)
MBPR15-2:hello-git hirayashingo$
```

図 48 「git status」と入力し、Git の「ステータス」を確認 (Mac)

「hello.txt ファイルは追跡されてないよ」という旨のメッセージが表示されます。

「git add」でファイルの追跡を開始

「git add」コマンドを使用して、hello.txt ファイルの追跡を開始します。「git add hello.txt」と入力し、「Enter」キーを押します。



```
Last login: Wed Mar 16 23:50:20 on ttys001
[MBPR15-2:~ hirayashingo$ cd Documents/
[MBPR15-2:Documents hirayashingo$ mkdir hello-git
[MBPR15-2:Documents hirayashingo$ cd hello-git/
[MBPR15-2:hello-git hirayashingo$ touch hello.txt
[MBPR15-2:hello-git hirayashingo$ ls
hello.txt
[MBPR15-2:hello-git hirayashingo$ git init
Initialized empty Git repository in /Users/hirayashingo/Documents/hello-git/.git
/
[MBPR15-2:hello-git hirayashingo$ git status
On branch master

Initial commit

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

    hello.txt

nothing added to commit but untracked files present (use "git add" to track)
[MBPR15-2:hello-git hirayashingo$ git add hello.txt ←
[MBPR15-2:hello-git hirayashingo$
```

図 49 「git add hello.txt」と入力し、hello.txt ファイルの追跡を開始 (Mac)

再び「git status」コマンドを使用して、ステータスを確認してみましょう。「git status」と入力し、「Enter」キーを押します。

hello.txt

nothing added to commit but untracked files present (use "git add" to track)
[MBPR15-2:hello-git hirayashingo\$ git add hello.txt
[MBPR15-2:hello-git hirayashingo\$ git status ←
On branch master

Initial commit

Changes to be committed:
(use "git rm --cached <file>..." to unstage)

new file: hello.txt

MBPR15-2:hello-git hirayashingo\$"/>

図 50 「git status」と入力し、Git の「ステータス」を確認（Mac）

hello.txt ファイルがコミットできる状態になりました。

「git commit」で変更内容を「リポジトリ」にコミット

「git commit」コマンドを使用して、変更内容を「リポジトリ」にコミットします。「git commit -m "first commit"」と入力し、「Enter」キーを押します。

Initial commit

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

hello.txt

nothing added to commit but untracked files present (use "git add" to track)
[MBPR15-2:hello-git hirayashingo\$ git add hello.txt
[MBPR15-2:hello-git hirayashingo\$ git status
On branch master

Initial commit

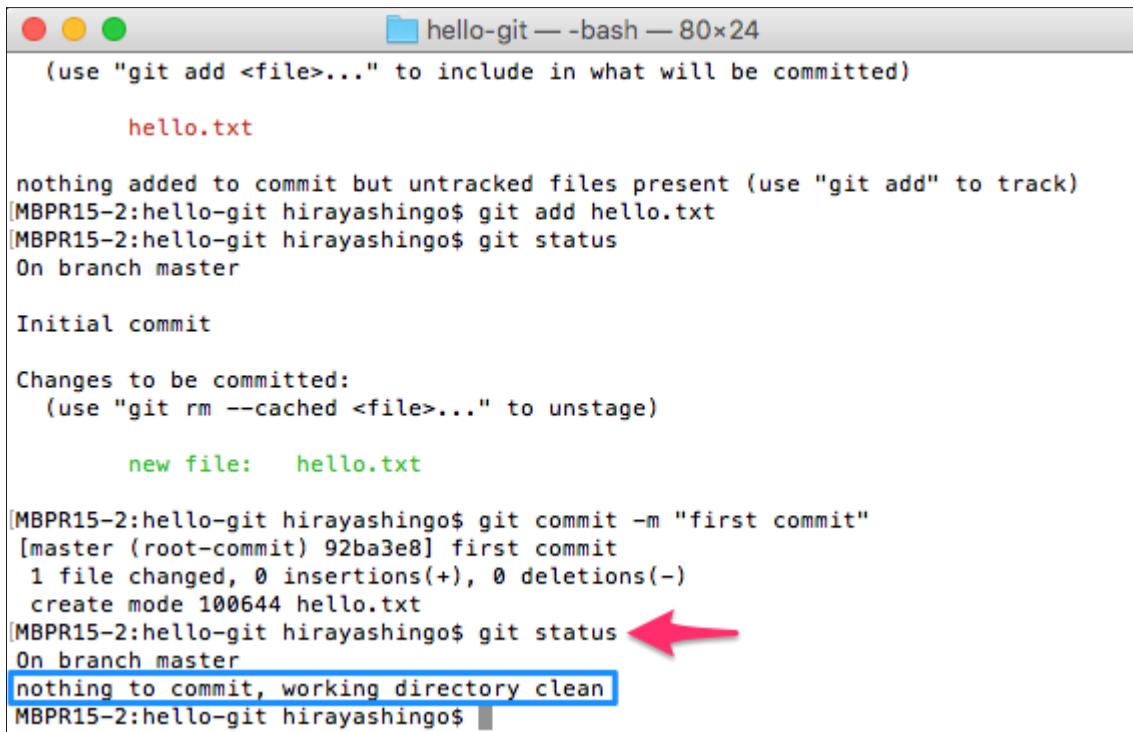
Changes to be committed:
(use "git rm --cached <file>..." to unstage)

new file: hello.txt

[MBPR15-2:hello-git hirayashingo\$ git commit -m "first commit" ←
[master (root-commit) 92ba3e8] first commit
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 hello.txt
MBPR15-2:hello-git hirayashingo\$"/>

図 51 「git commit -m "first commit"」と入力し、変更内容を「リポジトリ」にコミット（Mac）

hello.txt ファイルがコミットされました。「git status」コマンドを使用して、ステータスを確認してみましょう。



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

hello.txt

nothing added to commit but untracked files present (use "git add" to track)
[MBPR15-2:hello-git hirayashingo$ git add hello.txt
[MBPR15-2:hello-git hirayashingo$ git status
On branch master

Initial commit

Changes to be committed:
(use "git rm --cached <file>..." to unstage)

    new file:   hello.txt

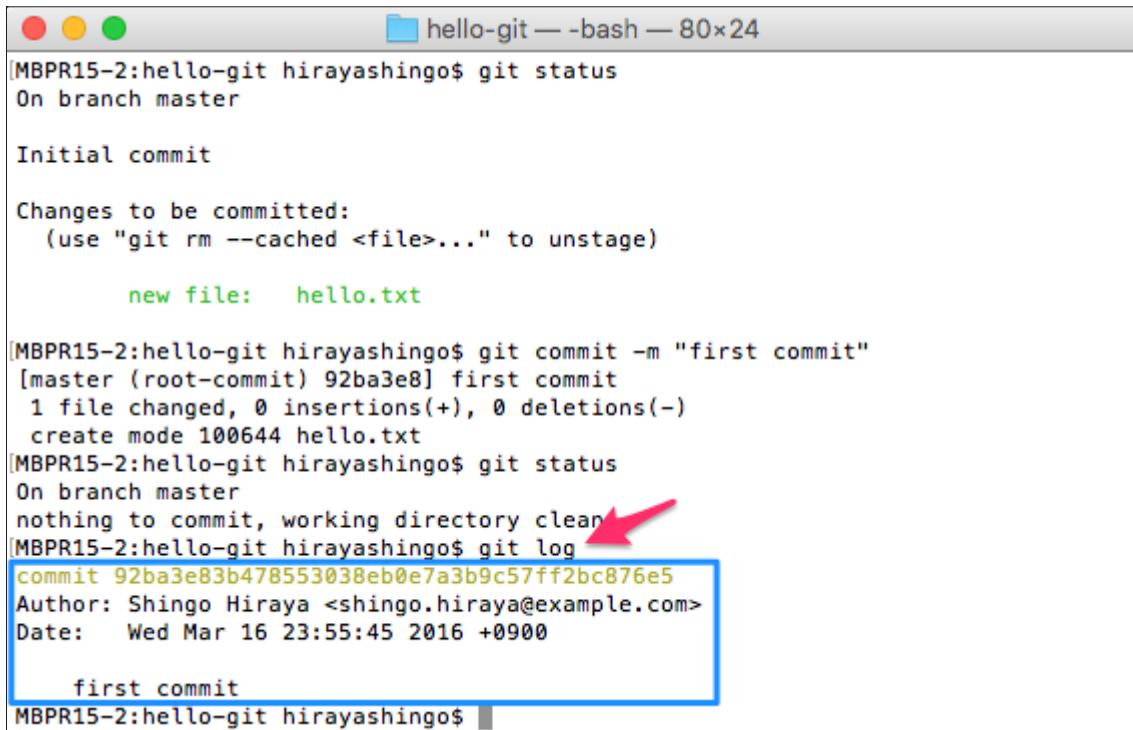
[MBPR15-2:hello-git hirayashingo$ git commit -m "first commit"
[master (root-commit) 92ba3e8] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 hello.txt
[MBPR15-2:hello-git hirayashingo$ git status ←
On branch master
nothing to commit, working directory clean
MBPR15-2:hello-git hirayashingo$ ]
```

図 52 「git status」と入力し、Git の「ステータス」を確認 (Mac)

「作業スペースがクリーンになった」という旨のメッセージが表示されます。

「git log」でコミットの履歴を確認

最後に「git log」コマンドを使用して、コミットの履歴を確認してみましょう。「git log」と入力し、「Enter」キーを押します。



```
[MBPR15-2:hello-git hirayashingo$ git status
On branch master

Initial commit

Changes to be committed:
(use "git rm --cached <file>..." to unstage)

    new file:   hello.txt

[MBPR15-2:hello-git hirayashingo$ git commit -m "first commit"
[master (root-commit) 92ba3e8] first commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 hello.txt
[MBPR15-2:hello-git hirayashingo$ git status ←
On branch master
nothing to commit, working directory clean
[MBPR15-2:hello-git hirayashingo$ git log ←
commit 92ba3e83b478553038eb0e7a3b9c57ff2bc876e5
Author: Shingo Hiraya <shingo.hiraya@example.com>
Date:   Wed Mar 16 23:55:45 2016 +0900

  first commit
MBPR15-2:hello-git hirayashingo$ ]
```

図 53 「git log」と入力し、コミットの履歴を確認 (Mac)

以下の 3 つの情報が表示されます。

- コミットしたユーザーの情報
- 日付
- コミットメッセージ

「コミットしたユーザーの情報」の名前とメールアドレスは、「初期設定」で設定した情報です。

次回からは、コマンドなど実践的な使い方を紹介

本記事では Git を使い始めるための環境構築について解説しました。Git のインストールと動作確認は無事に終わりましたでしょうか。

今回解説を省略した、コマンドなど実践的な使い方などは、次回以降で詳しく解説していきます。お楽しみに!

02.“はじめの Git” ——超基本的な作業フローと 5 つのコマンド

(2016 年 04 月 26 日)

本連載では、バージョン管理システム「Git」と Git のホスティングサービスの 1 つ「GitHub」を使うために必要な知識を基礎から解説していきます。今回は、Git の基本的な作業フローに登場する 3 つの場所について図を交えて解説し、フローごとにコマンドの使い方を紹介します。

まずは、とにかく手を動かして試してみないと

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」と Git のホスティングサービスの 1 つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終える頃には、Git や GitHub の基本的な操作が身に付いた状態になっていると思います。

連載第 2 回目の本稿のテーマは「Git の基本的な作業フローを学ぶ」です。

前回の「初心者でも Windows や Mac ができる、Git のインストールと基本的な使い方」でも「Git の基本的な作業フロー」を扱いましたが、かなり急ぎ足な解説となってしまいました。今回は、作業フローの各要素について、前回よりも詳しく解説していきます。

「Git のインストール」や「初期設定」については、前回の記事で解説したので、そちらを参考にしてください。また「バージョン管理システムそのもの」「Git と他のバージョン管理システムとの違い」などについては、記事「ガチで 5 分で分かる分散型バージョン管理システム Git」を参照してください。

Git の基本的な作業フロー

バージョン管理を利用したソフトウェア開発では「バグ修正の完了」「機能の追加完了」などの「ある特定のタイミング」ごとに各ソースコードの状態をリポジトリに保存します。Git を使用する場合、「ファイルの変更」から「ファイルの状態の保存」までの 1 サイクルの間に以下の 3 つの操作を行うことになります。

1. 「作業ディレクトリ」上のファイルを変更する（あるいはファイルを新たに追加する）
2. 変更済みのファイルをステージする
 - 「git add」コマンドを使用して「変更済みのファイル」を「ステージングエリア」に追加する
3. ステージ済みのファイルをコミットする

- 「git commit」コマンドを使用して、手順 2 でステージしたファイルの「スナップショット」を「Git リポジトリ」に保存する

これが「Git の基本的な作業フロー」です。

Git の基本的な作業フローに登場する 3 つの場所

上記「作業フロー」で登場した「場所」を整理すると、以下の通りになります。

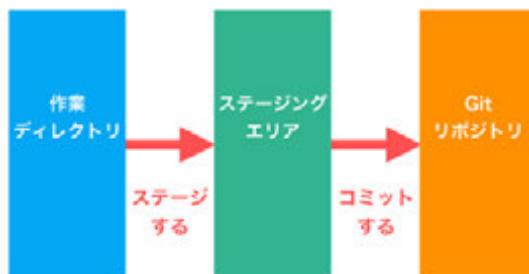


図 1 Git の基本的な作業フローに登場する 3 つの場所

1.Git リポジトリ (Git Repository)

- ・ファイルのスナップショットなどを保存する領域
- ・実態は「作業ディレクトリ」の中の「.git ディレクトリ」

2. 作業ディレクトリ (Working Directory)

- ・ファイルの編集作業を行うディレクトリ
- ・Git を使ったバージョン管理では、作業ディレクトリ内のファイルの変更履歴を保存していく

3. ステージングエリア (Staging Area)

- ・「作業ディレクトリ」と「Git リポジトリ」の中間に存在する領域。「インデックス」と呼ばれることがある
- ・コミットする準備ができたファイルはここに追加する

作業ディレクトリ内のファイルの状態

また、「作業ディレクトリ」内のファイルの状態を整理すると、以下の通りになります。

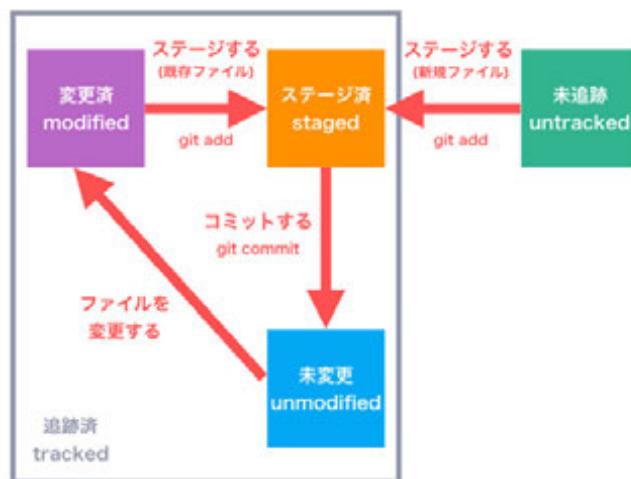


図 2 作業ディレクトリ内のファイルの状態

1. 追跡済み (tracked) : Git が変更を追跡しているファイル

1) 未変更 (unmodified)

- スナップショットがリポジトリに保存されているファイル

2) 変更済み (modified)

- Git が変更を追跡しているが、変更がリポジトリに保存されていないファイル

3) ステージ済み (staged)

- ステージングエリアに追加され、次回コミットの対象となっているファイル

2. 未追跡 (untracked) : Git が変更を追跡していないファイル

Git のコマンドラインツールを使う利点

本連載では、基本的にコマンドラインツールを使って Git の操作を解説していく予定です。コマンドラインツールを使う場合の利点には以下のようなものがあります。

- Git のコマンド群を全て実行できる
- 環境によらず、同じ手順に沿って操作できる

Git リポジトリを準備する——git init コマンド

Git リポジトリを新規作成するには、Git で管理したいディレクトリに移動し、「git init」コマンドを使用します。

コマンド実行

今回は空のディレクトリ「hello-git-2」に Git リポジトリを作成してみます。「hello-git-2」が「作業ディレクトリ」になります。

```
$ cd /Users/hirayashingo/Documents/hello-git-2
$ git init
Initialized empty Git repository in /Users/hirayashingo/Documents/hello-git-2/.git/
```

ファイルシステム上の確認

「ls -a」コマンドを使用すれば、「.git ディレクトリ」が作成されていることを確認できます。このディレクトリの中のファイルが Git リポジトリの実態です。ファイルのスナップショットなどが保存されます。

```
$ ls -a
.          ..        .git
```

以上の操作で作業ディレクトリ「hello-git-2」内のファイルの変更を Git に追跡してもらうための準備が整いました。

git clone コマンドについては、後の連載で

既に Git リポジトリが存在する場合は「git clone」コマンドを使用してリポジトリをクローンすることができます。このコマンドについては連載第 9 回記事の「リモートリポジトリを複製してローカルに取り込む——git clone コマンド」をご覧ください。

git status コマンドで確認

ここで「git status」コマンドを使用して、ステータスを確認してみましょう。

```
$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

1 行目の「On branch master」は現在「master」という名前の「ブランチ」にいるということを表しています。「ブランチ」については次回以降の記事で解説します。

2 行目のメッセージは、まだ 1 回もコミットしていないので表示されていますが、今のところ無視して構いません。

3 行目の「nothing to commit」については、「作業ディレクトリ」がクリーンな状態（コミットし忘れているファイルなどがない状態）なので、このメッセージが表示されます。「hello-git-2」ディレクトリにはファイルが 1 つもないのに当然ですね。

ファイルを追加する

1 つ前の「Git リポジトリを準備する」で Git リポジトリを作成しました。次に新しいファイルを追加してみます。

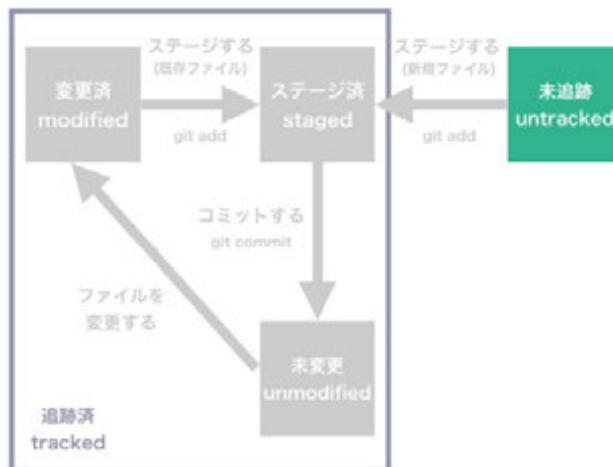


図 3 「ファイルを追加する」で行う操作

コマンド実行

今回は「echo」コマンドを使用して「hello.txt」ファイルを作成します。

```
$ echo Hello > hello.txt
$ ls
hello.txt
```

「cat」コマンドを使用すれば、ファイルの内容を確認できます。内容は「Hello」というテキスト1行です。

```
$ cat hello.txt
Hello
```

git status コマンドで確認

ここで再び Git のステータスを確認します。

```
$ git status
On branch master

Initial commit

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

    hello.txt

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

先ほどとは異なるメッセージが表示されると思います。

「Untracked files」欄には「未追跡」のファイルが表示されます。Git はまだ「hello.txt」ファイルの変更を追跡していないので、ここに表示されます。

ファイルをステージする——git add コマンド

「ファイルを追加する」で新規作成したファイルをステージングエリアに追加しましょう。

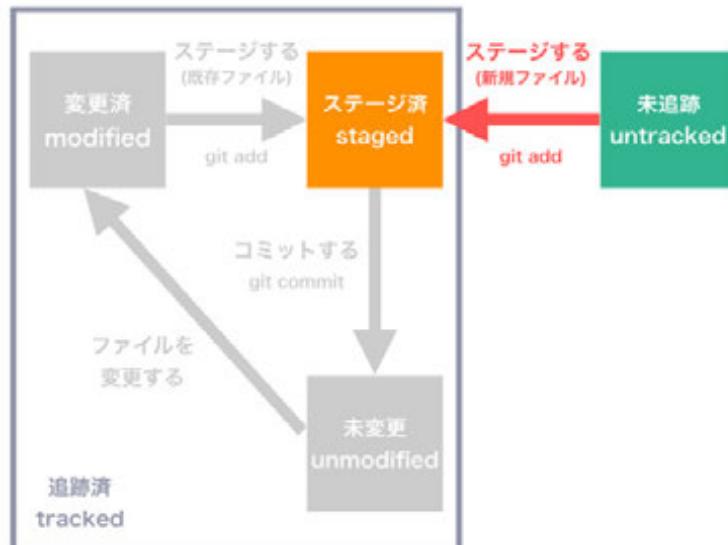


図 4 「ファイルをステージする」で行う操作

コマンド実行

「git add」コマンドを使用します。

```
$ git add hello.txt
```

git status コマンドで確認

Git のステータスを確認すると、以下のようなメッセージが表示されます。

```
$ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   hello.txt
```

「Changes to be committed」欄には「ステージ済み」のファイルが表示されます。

「hello.txt」ファイルは「git add」コマンドでステージングエリアに追加されました。コミットできる状態になっているので、ここに表示されます。また、「hello.txt」ファイルは新たに追加されたファイルなので、ファイル名の前に「new file」が表示されます。

ファイルをコミットする——git commit コマンド

「ファイルをステージする」で行った操作によって、「hello.txt」ファイルは Git リポジトリにコミットできる状態になりました。

「git commit」コマンドを使用して「hello.txt」ファイルをコミットしましょう。

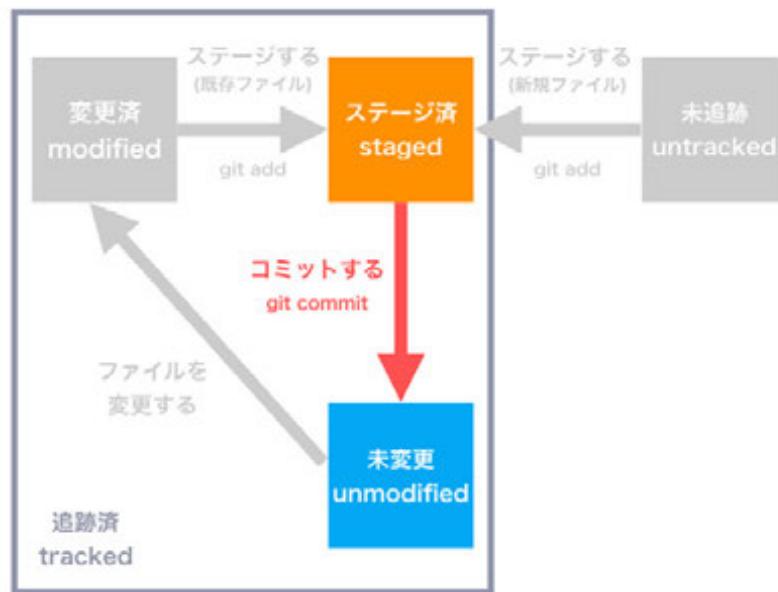


図 5 「ファイルをコミットする」で行う操作

「-m」オプションでコミットメッセージをインラインで記述

以下の例では、「-m」オプションを付けてコミットメッセージをインラインで記述しています。オプションなしの場合はエディタ（Vim や Emacs など）でコミットメッセージを入力できます。

```
$ git commit -m "first commit"
[master (root-commit) dc32bd2] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt
```

「SHA-1 チェックサム」とは

コミット後のメッセージには、以下のような情報が含まれています。

- コミットしたブランチ（master）
- SHA-1 チェックサム（dc32bd2）
- コミットメッセージ（first commit）
- 変更したファイルの数（1 file changed）
- 追加した行数（1 insertion）

「SHA-1 チェックサム」はコミットを識別するための文字列であり、コミットの歴史を書き換える場合などに使用します。

ファイルを変更する

ここまで操作で「Git の基本的な作業フロー」を 1 周分行いました。ここで、さらにファイルを変更して、作業フローの 1 周目と何が変わるのが見てみましょう。

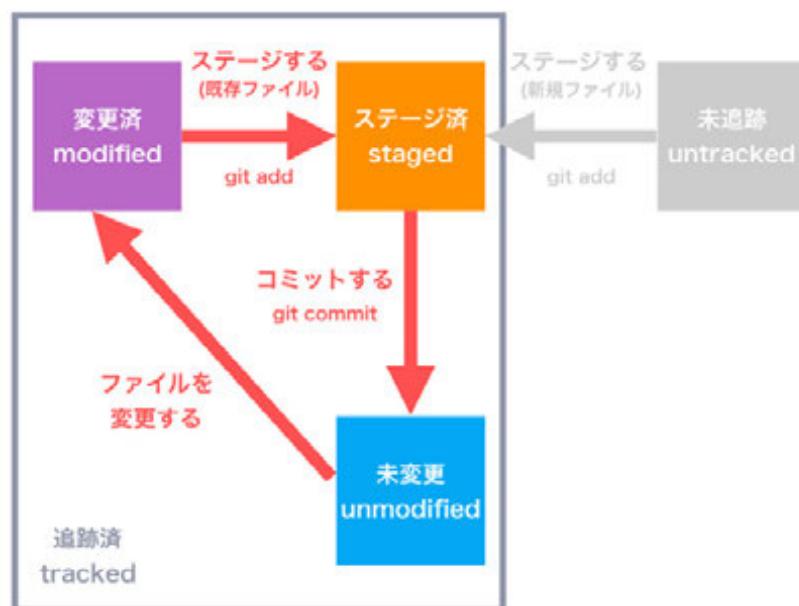


図 6 「ファイルを変更する」で行う操作

「ファイルを追加する」で作成した「hello.txt」ファイルに対し「echo」コマンドを使用して 2 行目を追加します。

```
$ echo goodbye >> hello.txt
$ cat hello.txt
Hello
goodbye
```

「Changes not staged for commit」が表すこと

Git のステータスを確認します。

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git checkout -- <file>..." to discard changes in working directory)
```

```
modified:   hello.txt

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

「Changes not staged for commit」という欄が表示されます。この欄は「ファイルが変更されている」か
つ「変更がステージされていない」ファイルがある場合に表示されます。「hello.txt」ファイルは、今「変更済み」状態になっているので、この欄に表示されます。

変更をステージングエリアに追加

次に、変更をステージングエリアに追加しステータスを確認します。

```
$ git add hello.txt
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

modified:   hello.txt
```

これで「hello.txt」ファイルに対する変更をコミットする準備が整いました。

今回は、既に Git が変更を追跡しているファイルに対する変更をステージングエリアに追加しました。そのため、「Changes to be committed」欄のファイル名の前に「modified」（変更済み）と書かれています。

変更をコミット

最後にコミットします。

```
$ git commit -m "edit hello.txt"
[master 998e5a0] edit hello.txt
 1 file changed, 1 insertion(+)
```

コミット履歴を確認する——git log コマンド

本記事では、これまでにコミットを 2 回行いました。「git log」コマンドを使用してコミットの履歴を確認してみましょう。

```
$ git log
commit 998e5a0b8f7397f93ba18d7551d804812e957069
```

```
Author: Shingo Hiraya <shingo.hiraya@example.com>
Date: Mon Apr 18 06:56:39 2016 +0900
```

```
edit hello.txt
```

```
commit dc32bd20c33bcae780565ccf4c51745ddefd44dd
Author: Shingo Hiraya <shingo.hiraya@example.com>
Date: Mon Apr 18 01:27:58 2016 +0900
```

```
first commit
```

コミットごとに以下の情報が表示されます。

- SHA-1 チェックサム
- 作者の名前とメールアドレス
- コミット日時
- コミットメッセージ

「-p」オプションでコミットの変更点を表示

「git log」コマンドにも、たくさんオプションが存在します。例えば、「-p」オプションを使うと、そのコミットの変更点を表示することもできます。

```
$ git log -p
commit 998e5a0b8f7397f93ba18d7551d804812e957069
Author: Shingo Hiraya <shingo.hiraya@example.com>
Date: Mon Apr 18 06:56:39 2016 +0900
```

```
edit hello.txt
```

```
diff --git a/hello.txt b/hello.txt
index e965047..c86756d 100644
--- a/hello.txt
+++ b/hello.txt
@@ -1 +1,2 @@
Hello
+Goodbye
```

```
commit dc32bd20c33bcae780565ccf4c51745ddefd44dd
Author: Shingo Hiraya <shingo.hiraya@example.com>
Date: Mon Apr 18 01:27:58 2016 +0900
first commit
```

```
diff --git a/hello.txt b/hello.txt
new file mode 100644
index 0000000..e965047
--- /dev/null
+++ b/hello.txt
@@ -0,0 +1 @@
+Hello
```

1回目のコミットで文字列「Hello」を、2回目のコミットで文字列「Goodbye」を追加したことが分かります。

本稿で紹介した Git の基本的なコマンド一覧

本稿では「Git の基本的な作業フロー」について解説しました。Git を使用したバージョン管理のフローについてのイメージをつかめたでしょうか。

本稿で紹介した Git のコマンドは以下の通りです。特に「git status」「git add」「git commit」コマンドは頻繁に使うことになります。

- **git init** : カレントディレクトリに Git リポジトリを新規作成する
- **git status** : 作業ディレクトリとステージングエリア上のファイルの状態を表示する
- **git add <ファイル名>** : 変更済みのファイルをステージングエリアに追加する
- **git commit** : ステージ済みのファイルのスナップショットを「Git リポジトリ」に保存する。コミットメッセージはエディタで入力する
- **git commit -m "<メッセージ>"** : 動作は「git commit」と基本的に同じ。コミットメッセージをインラインで指定したい場合は、こちらを使う
- **git log** : コミットの履歴を表示する
- **git log -p** : 「git log」で表示可能な項目を表示しつつ、さらにコミットの変更点も表示する

次回の記事では今回解説を省略した「ブランチ」について解説する予定です。お楽しみに!

参考書籍

- 『Pro Git』 (written by Scott Chacon and Ben Straub and published by Apress)

03. ポインタ嫌いでも分かる Git ブランチの基本 ——作成、確認、切り替え、master にマージ、削除

(2016年06月10日)

本連載では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。今回は、ブランチの基本として作成、確認、切り替え、masterにマージ、削除という一連の作業をコマンドの使い方と図を交えて解説します。

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終える頃には、GitやGitHubの基本的な操作が身に付いた状態になっていると思います。

連載第3回目の本稿のテーマは「ブランチの基本」です。

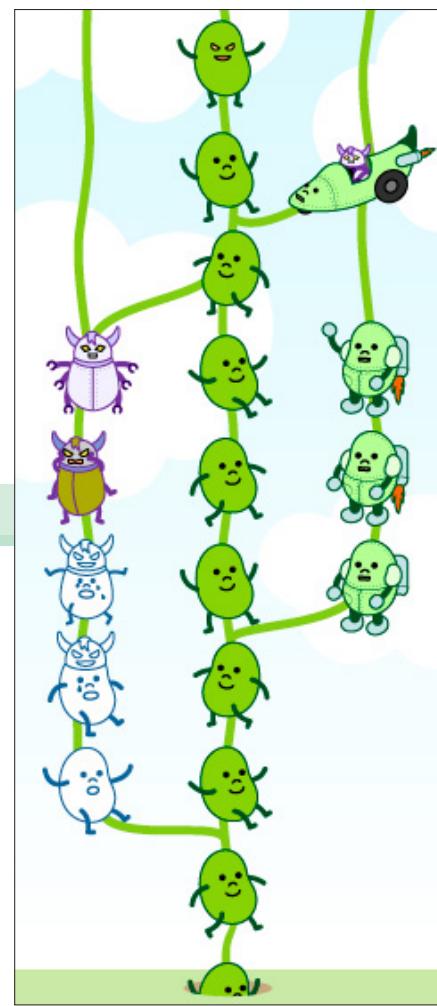
前回の「[はじめての Git](#)——超基本的な作業フローと5つのコマンド」では「Gitの基本的な作業フロー」を解説しましたが、「ブランチ」についてはほとんど触れませんでした。

今回は実際に作業を進めていくながら「ブランチ」の基本的な要素を解説していきます。

ブランチとは

英単語「branch」は「枝」「支流」などの意味を持ちます。Gitの用語としての「ブランチ」は、「コミット履歴の流れを枝分かれさせるための機能」のことを指します。開発の本流から枝分かれしたブランチは他のブランチから独立した環境を持つことができるので、開発作業を平行して進めていくことができます。

手を動かしながらの方が理解しやすいと思いますので、作業を開始しましょう。次章で準備を行って、その次の章から「ブランチ」に関する操作を試していきます。



ブランチ（枝分かれ）のイメージ（記事「[ガチで5分で分かる分散型バージョン管理システムGit](#)」より引用）

「ブランチ」を試すための環境を準備

まずは、連載第2回と同じ操作を行って、「ブランチ」を試すための環境を作ります。連載第2回に沿ってリポジトリの作成とコミット2回を行っている場合は、読み飛ばして「[カレントブランチを確認](#)」の章に進んで構いません。

Git リポジトリを準備

適当なディレクトリに移動し、[Git リポジトリを作成](#)します。

```
$ cd /Users/hirayashingo/Documents/hello-git-2
$ git init
Initialized empty Git repository in /Users/hirayashingo/Documents/hello-git-2/.git/
```

ファイルの追加とコミット

「hello.txt」[ファイルを追加し、ステージ](#)して、[コミット](#)します。

```
$ echo Hello > hello.txt
$ git add hello.txt
$ git commit -m "first commit"
[master (root-commit) dc32bd2] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 hello.txt
```

ファイルの変更とコミット

「hello.txt」[ファイルの内容を変更し、ステージ](#)して、[コミット](#)します。

```
$ echo goodbye >> hello.txt
$ git add hello.txt
$ git commit -m "edit hello.txt"
[master 998e5a0] edit hello.txt
 1 file changed, 1 insertion(+)
```

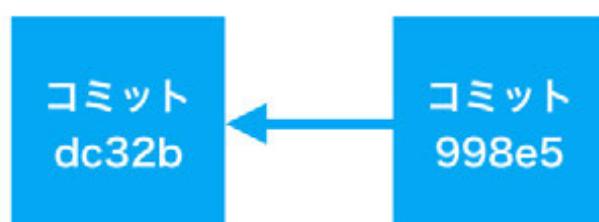


図 1 準備完了時点でのコミット履歴

カレントブランチを確認する

リポジトリがあるディレクトリに移動していない場合は、移動します。

```
$ cd /Users/hirayashingo/Documents/hello-git-2
```

git status コマンドで現在いるブランチを確認する

「現在、どのブランチにいるか」を確認する方法はたくさんあります。

まずは、「git status」コマンドを使用してカレントブランチ（現在いるブランチ）を確認してみましょう。

```
$ git status  
On branch master  
nothing to commit, working directory clean
```

「On branch master」と表示されます。現在「master」という名前の「ブランチ」にいることを表しています。

ブランチの一覧を表示する——git branch コマンド

「git branch」コマンドを使用して、カレントブランチを確認することもできます。「git branch」コマンドはブランチの一覧を表示するコマンドです。

```
$ git branch  
* master
```

現在「master」ブランチしかないので「master」だけ表示されます。また、カレントブランチの名前の左には「*」が表示されます。

ブランチの実体とは

「一連のコミット履歴」で構成される「枝」のことを「ブランチ」と呼びますが、「ブランチ」の実体は、「一連のコミット履歴」の最新のコミットを指すポインタです。図 2 でいうと、「master」と書かれた黄色の矢印や「HEAD」と書かれた赤色の矢印のことです。

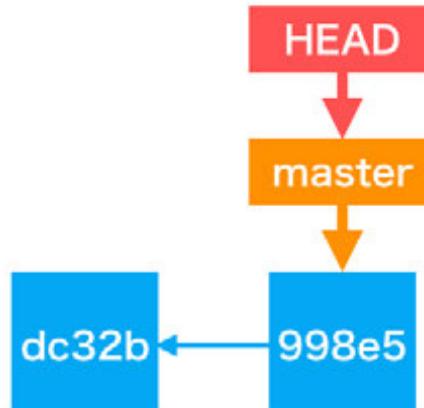


図 2 「ブランチ」の実体はコミットを指すポインタ

「git log」コマンドの「--oneline --decorate」オプションで「ポインタが、どのコミットを指しているか」を表示

「git log --oneline --decorate」コマンドを使用すれば、「ポインタが、どのコミットを指しているか」が分かります。これまでに2回コミットを行ってきましたが、現在、「master」は最新のコミット（2回目のコミット）を指しています。

```
$ git log --oneline --decorate
8a0afb3 (HEAD -> master) edit hello.txt
8d7430a first commit
```

カレントブランチを表す「HEAD」

上記メッセージに出てきた「HEAD」は基本的にカレントブランチを指します。

ブランチ作成と切り替え

ブランチを新規作成・一覧表示する——git branch コマンド

新しいブランチを作成しましょう。「git branch <新規作成するブランチ名>」コマンドを使用します。ここでは「second」という名前のブランチを新しく作ります。

```
$ git branch second
```

再び「git branch」コマンドを使用してブランチ一覧を表示します。「second」ブランチが増えましたが、まだ「master」ブランチにいます。

```
$ git branch
* master
  second
```

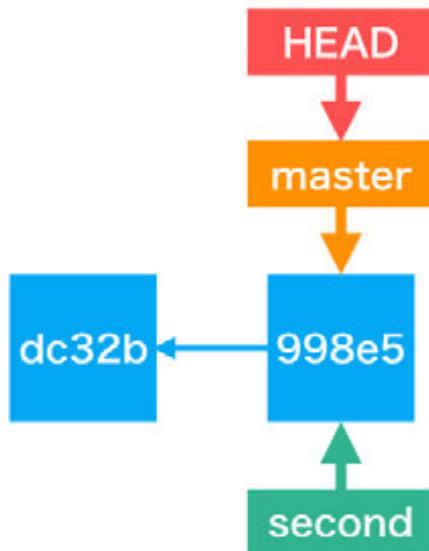


図 3 「second」の追加を行つただけの状態

ブランチの切り替えを行う——git checkout コマンド

ブランチの切り替えを行うには、「git checkout <ブランチ名>」コマンドを使用します。

```
$ git checkout second
Switched to branch 'second'
$ git branch
  master
* second
```

「second」ブランチに切り替えることができました。

ログを確認する

ここで再び、「git log --oneline --decorate」コマンドを実行してみます。「second」ブランチを作成して切り替えましたが、それ以降新しいコミットをしていません。そのため、現在「master」と「second」ブランチは同じコミットを指しています（2 行目）。

```
$ git log --oneline --decorate
8a0afb3 (HEAD -> second, master) edit hello.txt
8d7430a first commit
```

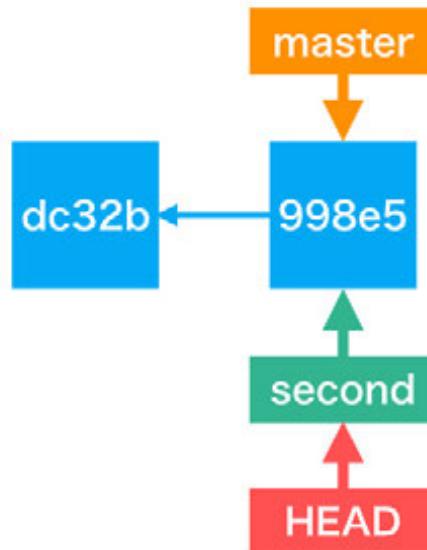


図 4 「HEAD」が「second」を指している状態

「second」 ブランチ上で 3 つ目のコミットを作る

「second」 ブランチを作成し「second」 ブランチに切り替えることができました。「second」 ブランチ上で作業をしていきましょう。

「hello.txt」 ファイルに 1 行追加し、コミットします。

```
$ echo konnichiwa >> hello.txt
$ git add hello.txt
$ git commit -m "add konnichiwa to hello.txt"
[second 6a2a9e2] add konnichiwa to hello.txt
 1 file changed, 1 insertion(+)
$ git status
On branch second
nothing to commit, working directory clean
```

「git log --oneline --decorate」 コマンドを実行します。

```
$ git log --oneline --decorate
d90401a (HEAD -> second) add konnichiwa to hello.txt
8a0afb3 (master) edit hello.txt
8d7430a first commit
```

「second」 ブランチは、図 5 では左から 3 つ目、コマンド実行結果では下から 3 つ目のコミットを指しています（2 行目）。「master」 は、そのままです。

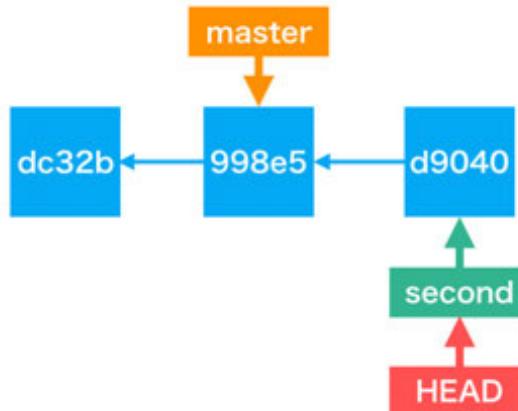


図 5 コミット後、「second」が左から 3 つ目のコミットを指している状態

「second」ブランチを「master」ブランチに「マージ」

「second」ブランチ上で行った変更を「master」ブランチに反映します。

あるブランチで行った変更を別のブランチに反映する操作のことを「マージ」と呼びます。「マージ」を行うには「git merge <マージ元のブランチ名>」コマンドを使用します。

「master」ブランチに切り替え

「git checkout <ブランチ名>」コマンドで、「master」ブランチに切り替えます。

```
$ git checkout master
Switched to branch 'master'
$ git log --oneline --decorate
8a0afb3 (HEAD -> master) edit hello.txt
8d7430a first commit
```

念のため、hello.txt の中身を確認してみます。「second」ブランチ上で追加した 1 行「konnichiwa」はありません。

```
$ cat hello.txt
Hello
goodbye
```

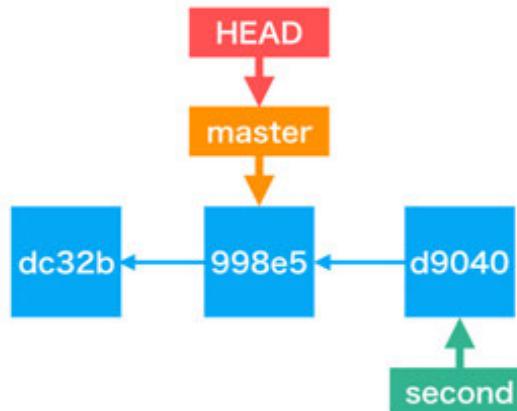


図 6 「HEAD」（カレントブランチ）が「master」を指している状態

ファイルのマージを実行する——git merge コマンド

「git merge <ブランチ名>」コマンドを実行します。

```
$ git merge second
Updating 8a0afb3..d90401a
Fast-forward
 hello.txt | 1 +
 1 file changed, 1 insertion(+)
```

「git log --oneline --decorate」コマンドを実行します。

```
$ git log --oneline --decorate
d90401a (HEAD -> master, second) add konnichiwa to hello.txt
8a0afb3 edit hello.txt
8d7430a first commit
```

「master」と「second」ブランチの両方が下から 3 つ目のコミットを指すようになりました（2 行目）。

Fast-forward (早送り) マージとは

「master」ブランチをマージ前後で比較すると、「master」ブランチが指すコミットが、図 7 では左から「2 つ目」から「3 つ目」に変わっただけです。

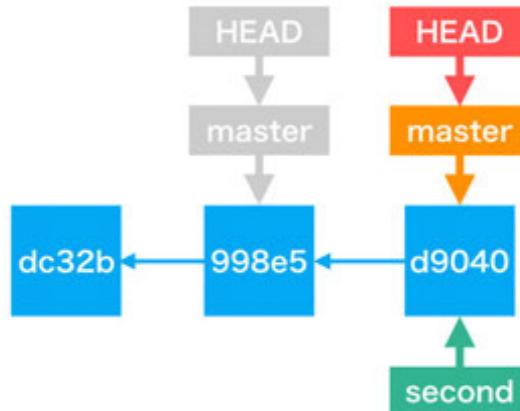


図 7 「master」が指すコミットが左から 2 つ目から 3 つ目に移動した状態

今回は「master」ブランチはそのまま、「second」ブランチだけ履歴が進んだだけなので、「master」ブランチが指すコミットが変わっただけで済みました。このようなマージのことを「Fast-forward（早送り）マージ」と呼びます。

今回「git checkout <ブランチ名>」コマンドを実行した結果表示されるメッセージに「Fast-forward」と表示されたのは、このためです。

ブランチを削除する——git branch コマンドの「-d」オプション

「second」ブランチ上で行った変更は「master」ブランチに反映したので、「second」ブランチは削除しても構いません。

不要になったブランチは「git branch -d <ブランチ名>」コマンドで削除できます。

```
$ git branch -d second
$ git branch
* master
```



図 8 「second」が削除された状態

本稿で紹介した Git の基本的なコマンド一覧

ここまでで「ブランチ作成」から「マージ」までの一連の操作を行いました。「ブランチ」についてのイメージをつかむことはできたでしょうか。

本記事で初めて登場した Git のコマンドは以下の通りです。

- `git branch` : ブランチの一覧を表示する
- `git log --oneline --decorate` : 「ブランチがどのコミットを指しているか」という情報と共にワンライントロゴを表示する
- `git checkout <ブランチ名>` : ブランチを切り替える
- `git merge <マージ元のブランチ名>` : 別のブランチをカレントブランチにマージする
- `git branch -d <ブランチ名>` : ブランチを削除する

今回行った一連の操作では片方の「second」ブランチだけで変更を進めたので、変更内容が複数ブランチ間でコンフリクト（衝突）することなく簡単にマージできました。しかし、実際の開発では変更内容がコンフリクトしてしまう場合があります。

次回は、意図的にコンフリクトが起こるような操作を行って、コンフリクトが起こったときの対処方法を説明します。

参考書籍

『Pro Git』(written by Scott Chacon and Ben Straub and published by Apress)

04.Git でコンフリクトしても慌てるな !! 解消に向けた 3 つの基本作業

(2016 年 07 月 14 日)

本連載では、バージョン管理システム「Git」と Git のホスティングサービスの 1 つ「GitHub」を使うために必要な知識を基礎から解説していきます。今回は、意図的にコンフリクトが起こるような操作を行って、コンフリクトが起こったときの対処方法を図を交えて説明します。

実際の開発では変更内容がコンフリクトしてしまう場合がある!

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」と Git のホスティングサービスの 1 つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終える頃には、Git や GitHub の基本的な操作が身に付いた状態になっていると思います。

連載第 4 回目の本稿のテーマは「コンフリクトが発生したときの対処方法」です。

前回の「[ピント嫌いでも分かる Git ブランチの基本——作成、確認、切り替え、master にマージ、削除](#)」までで「ブランチ作成」から「マージ」までの一連の操作を行いました。前回行った一連の操作では片方の「second」ブランチだけで変更を進めたので、変更内容が複数ブランチ間で「コンフリクト（衝突、競合）」することなく簡単にマージできました。しかし、実際の開発では変更内容がコンフリクトしてしまう場合があります。

今回は、意図的にコンフリクトが起こるような操作を行って、コンフリクトが起こったときの対処方法を説明します。

コンフリクトが発生したときに行う 3 ステップ

コンフリクトが発生したときに行う基本作業をまとめますと、以下のようになります。「1」の作業は特殊ですが、「2」と「3」は通常の Git の基本作業と同じです。

1. コンフリクトが発生している箇所を確認して対処する
2. 「git add」コマンドを実行する
3. 「git commit」コマンドを実行する

【準備 1】新たなブランチの作成とファイル変更

前回作業を終えた時点では、「second」ブランチを削除して「master」ブランチだけが存在しています。カレントブランチも「master」ブランチです。

```
$ git branch
* master
```



前回の作業を終えた状態（前回図 8 の再掲）

「git checkout -b」コマンドで「ブランチの作成」と「ブランチの切り替え」を同時に

この状態で新たにブランチを作成します。前回「second」ブランチまで作っていたので、新たなブランチは「third」とします。

「git checkout -b {新規作成するブランチ名}」コマンドを使用すれば、「ブランチの作成」と「ブランチの切り替え」を同時に行えます。

今回はブランチ作成と切り替えを同時に行ってみます。

```
$ git checkout -b third
Switched to a new branch 'third'
$ git branch
  master
* third
```

「third」ブランチが作成され、カレントブランチが「third」ブランチになりました。

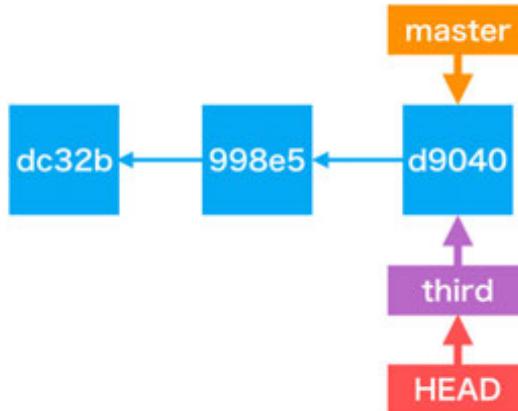


図1 「third」が作成され「HEAD」が「third」を指すようになった状態

「third」ブランチでのファイル変更

ここで、hello.txt の中身を確認してみます。3行表示されます。

```
$ cat hello.txt
Hello
goodbye
konnichiwa
```

「hello.txt」ファイルにさらに1行追加し、コミットします。

```
$ echo sayounara >> hello.txt
$ git add hello.txt
$ git commit -m "add sayounara to hello.txt"
[third b6d5577] add sayounara to hello.txt
 1 file changed, 1 insertion(+)
$ git status
On branch third
nothing to commit, working directory clean
```

4行目が追加されました。

```
$ cat hello.txt
Hello
goodbye
konnichiwa
sayounara
```

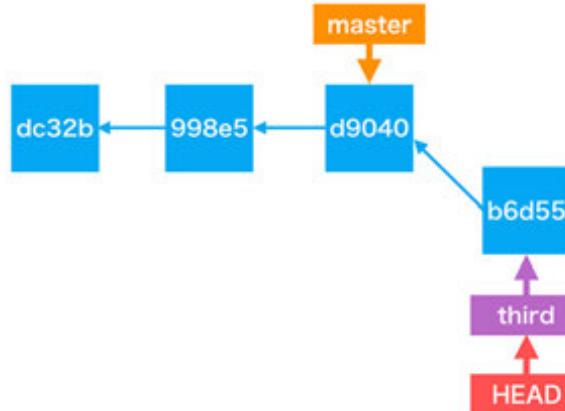


図 2 コミット後、「third」が 4 つ目のコミットを指している状態

【準備 2】「master」ブランチでのファイル変更

「master」ブランチへの切り替え

再び master ブランチに戻ります。

```
$ git checkout master
Switched to branch 'master'
$ git branch
* master
  third
```

ここで、hello.txt の中身を確認してみます。3 行のままでです。

```
$ cat hello.txt
Hello
goodbye
konnichiwa
```

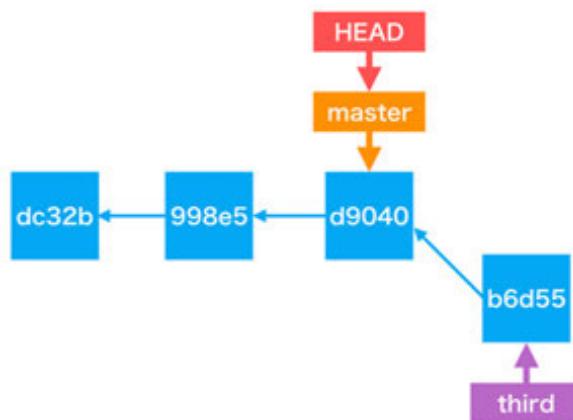


図 3 「HEAD」の向き先が「master」になった状態

「master」 ブランチでのファイル変更

「『third』 ブランチでのファイル変更」で追加した文字列とは別の文字列を追加し、コミットします。

```
$ echo konbanwa >> hello.txt
$ git add hello.txt
$ git commit -m "add konbanwa to hello.txt"
[master ad169d4] add konbanwa to hello.txt
 1 file changed, 1 insertion(+)
$ git status
On branch master
nothing to commit, working directory clean
```

4行目が追加されました。

```
$ cat hello.txt
Hello
goodbye
konnichiwa
konbanwa
```

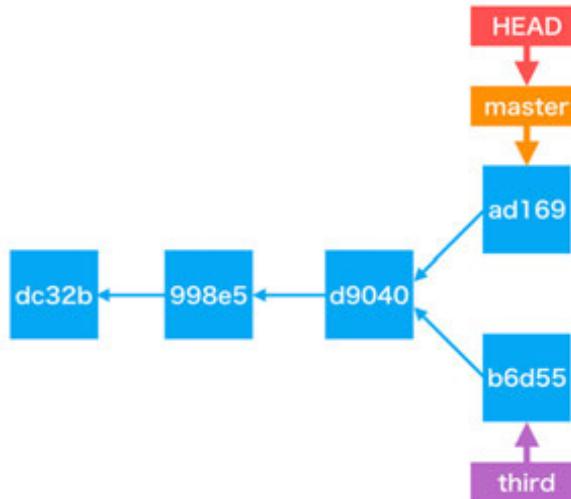


図 4 コミット履歴が 2 つに分岐した状態

【準備 3】今回作成したブランチを「master」ブランチにマージしてコンフリクトを起こす

今回新たに作成した「third」ブランチで行った変更を「master」ブランチにマージしましょう。「git merge」コマンドを使用します。

```
$ git merge third
Auto-merging hello.txt
CONFLICT (content): Merge conflict in hello.txt
Automatic merge failed; fix conflicts and then commit the result.
```

予想通り、hello.txt の変更がコンフリクトしました。「自動マージが失敗しました」というメッセージが表示されています。

コンフリクトを解消する 3 ステップの実践

【ステップ 1】コンフリクト発生箇所の確認

hello.txt ファイルを任意のエディタで開くと、以下のようになっています。

```
1. Hello
2. goodbye
3. konnichiwa
4. <<<<< HEAD
5. konbanwa
6. =====
7. sayounara
8. >>>>> third
```

4～6 行目の間が、マージした時のカレントブランチの内容です。また、6～8 行目の間が「third」ブランチ（マージ元のブランチ）の内容です。「どちらを残すのか」または「どちらも残すのか」を決めなければなりません。

ここでは、どちらも残すことにします。任意のエディタ上で 4、6、8 行目を削除し保存します。

```
1. Hello
2. goodbye
3. konnichiwa
4. konbanwa
5. sayounara
```

【ステップ 2】「git status」コマンドと「git add」コマンド

ここで「git status」コマンドを実行すると以下のようなメッセージが表示されます。「Unmerged paths」項目を見ると、「解決済みにするには git add コマンドを使用せよ」と書いてあります。

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

    both modified:  hello.txt

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

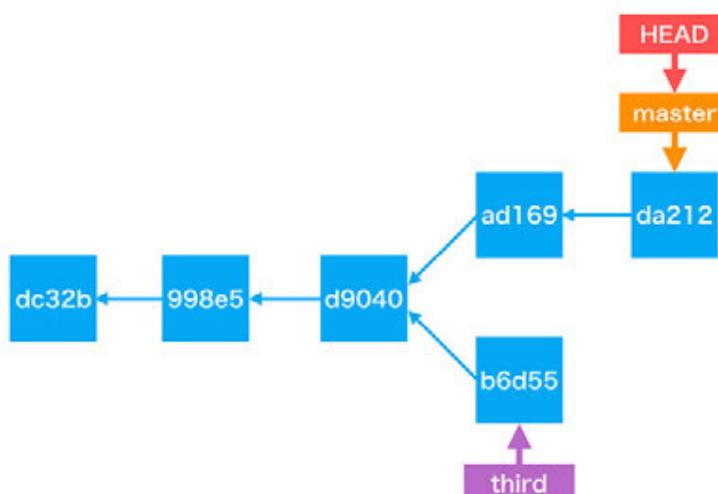
「git add」コマンドを実行します。

```
$ git add hello.txt
$ git status
On branch master
All conflicts fixed but you are still merging.
  (use "git commit" to conclude merge)

Changes to be committed:

  modified:  hello.txt
```

hello.txt ファイルをコミットできるようになりました。



【ステップ 3】「git commit」コマンド

「git commit」コマンドを実行します。

```
$ git commit  
[master da212e1] Merge branch 'third'  
$ git status  
On branch master  
nothing to commit, working directory clean
```

作業ディレクトリがクリーンな状態に戻りました。

コンフリクトが解消されたか確認

最後に「git log --oneline --decorate」コマンドを実行してみます。

```
$ git log --oneline --decorate  
da212e1 (HEAD -> master) Merge branch 'third'  
ad169d4 add konbanwa to hello.txt  
b6d5577 (third) add sayounara to hello.txt  
d90401a add konnichiwa to hello.txt  
8a0afb3 edit hello.txt  
8d7430a first commit
```

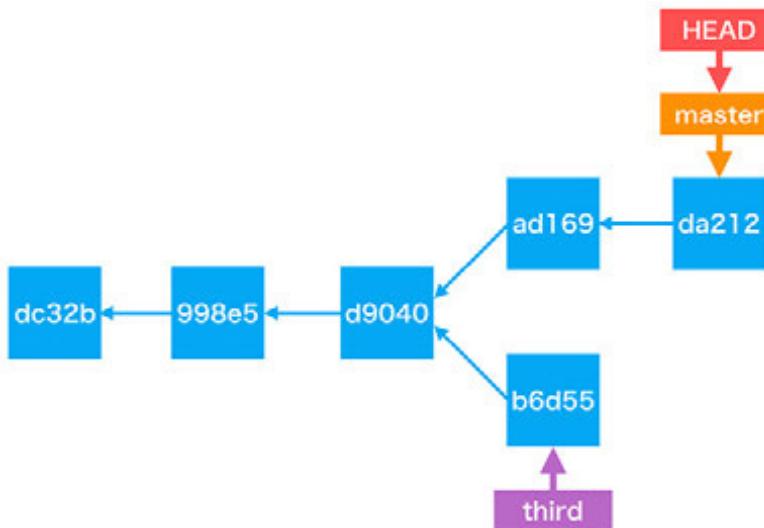


図 13 「third」と「master」で行った変更を統合した状態

「third」と「master」ブランチで行った変更を統合する操作は、以上で完了です。「third」ブランチは不要になったので削除できます。

本記事で初めて登場した Git のコマンド

本記事では「コンフリクトが発生したときの対処方法」について解説しました。

本記事で初めて登場した Git のコマンドは以下の通りです。

- **git checkout -b {新規作成するブランチ名}** : 「ブランチの作成」と「ブランチの切り替え」を同時に実行

次回は、前回と今回で紹介できなかった「ブランチ」の他の機能について解説する予定です。お楽しみに!

参考書籍

『Pro Git』 (written by Scott Chacon and Ben Straub and published by Apress)

05.Git コミット現場あるある ——やり直し、取り消し、変更したいときに使えるコマンド

(2016年08月29日)

本連載では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。今回は、コミットメッセージを間違えたり、追加すべきファイルをコミットに入れ忘れたりした場合の対処方法やファイルの変更の取り消し方法を説明。

開発現場ではコミットで間違えるなんて、よくあること

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終える頃には、GitやGitHubの基本的な操作が身に付いた状態になっていると思います。

連載第5回目の本稿では「コミットのやり直し」と「ファイルの変更の取り消し」を扱います。コミットメッセージを間違えたり、追加すべきファイルをコミットに入れ忘れたりした場合、そのコミットをやり直すことができます。

まずは、コミットをやり直す方法を解説していきます。実際に作業を進めていきながら、作業をやり直す方法を解説していきます。

Git リポジトリを作成しておく

任意のディレクトリに移動し、「コミットのやり直し」や「ファイルの変更の取り消し」を試すためのGitリポジトリを作成します。

今回は「ドキュメント」ディレクトリの中に「hello-git-5」ディレクトリを作成し、その中にGitリポジトリを作成しました。

```
$ cd /Users/hirayashingo/Documents/  
$ mkdir hello-git-5  
$ cd hello-git-5/  
$ git init  
Initialized empty Git repository in /Users/hirayashingo/Documents/hello-git-5/.git/
```

コミットメッセージを修正する

まずは「コミットメッセージを修正する」方法を解説していきます。

【準備】「コミットメッセージを間違えた」という状況を作る

「コミットメッセージを修正する」方法を解説するために、「コミットメッセージを間違えた」という状況を作ります。

まずはファイルを追加し、ステージします。ここでは「hello-5.txt」ファイルを追加しました。

```
$ echo Hello > hello-5.txt
$ git add hello-5.txt
```

そしてコミットします。ここではコミットメッセージをわざと間違えています。「first commit」ではなく「fist commit」というメッセージを指定しました。

```
$ git commit -m "fist commit"
[master (root-commit) 0512102] fist commit
 1 file changed, 1 insertion(+)
 create mode 100644 hello-5.txt
```

コミットログを確認してみても、コミットメッセージは「fist commit」になってしまっています。

```
$ git log
commit 051210244e102067c3291e4098b3acac242aa12c
Author: Shingo Hiraya <shingohiraya@example.com>
Date:   Thu Aug 11 20:35:20 2016 +0900

fist commit
```

「git commit --amend」コマンドで直前のコミットのコミットメッセージを修正する

「git commit --amend」コマンドを使用してコミットメッセージを変更します。

```
$ git commit --amend
```

上記コマンドを入力して「Enter」キーを押すと、エディタが起動します。

ここではデフォルトのエディタ「vi」でメッセージを編集する手順を解説します（Git の設定で明示的にエディタを指定していない場合は「vi」が起動します）。

図1 「vi」起動直後

「vi」では「モード」を切り替えながらテキストを編集していきます。起動直後は「コマンドモード」になっています。「コマンドモード」は文字通り各種コマンドを受け付けるモードです。テキストを編集するには「インサートモード」に切り替える必要があります。

「i」キーを押して「インサートモード」に切り替えてみましょう。「モード」が「インサートモード」になるとエディタ下部に「INSERT」と表示され、テキストを編集できるようになります。

2. git

fist commit

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date: Thu Aug 11 20:35:20 2016 +0900
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file: hello-5.txt
```

-- INSERT --

「INSERT」と表示される

図2 「インサートモード」に切り替え後

エディタ上に表示されているメッセージ「fist commit」を「first commit」に修正します。十字キーでカーソルを移動させて、文字「r」を挿入します。

2. git

first commit ← メッセージを修正

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date: Thu Aug 11 20:35:20 2016 +0900
#
# On branch master
#
# Initial commit
#
# Changes to be committed:
#       new file: hello-5.txt
```

-- INSERT --

図3 メッセージ編集後

修正が終わったら、「Esc」キーを押して、「コマンドモード」に切り替えます。エディタ下部に表示されていた「INSERT」が消えます。

図4 「コマンドモード」に切り替え後

「:wq」と入力し、「Enter」キーを押します。編集内容を保存してエディタを終了できます。

図5 「:wq」と入力

コミットメッセージを修正する手順は以上です。

修正結果

コミットログを表示すると、コミットメッセージが修正されたことを確認できます。

```
$ git log
commit a7e133b020e9359a944807f704e32e6b6f327a8d
Author: Shingo Hiraya <shingohiraya@example.com>
Date:   Thu Aug 11 20:35:20 2016 +0900

first commit
```

ステージし忘れていたファイルをコミットに追加する

次に「ステージし忘れていたファイルをコミットに追加する」方法を解説していきます。

【準備】「ファイルをステージし忘れた」という状況を作る

「ステージし忘れていたファイルをコミットに追加する」方法を解説するために、「ファイルをステージし忘れた」という状況を作っていきます。

2つのテキストファイルを追加します。ここでは「hello-5-2.txt」と「forgotten-file.txt」を追加しました。

```
$ echo hello-5-2 > hello-5-2.txt
$ echo forgotten > forgotten-file.txt
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    forgotten-file.txt
    hello-5-2.txt

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

次に、追加したファイルをステージします。ここでは片方のファイルだけステージします。もう片方のファイルは後でステージします。

```
$ git add hello-5-2.txt
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   hello-5-2.txt

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

    forgotten-file.txt
```

そして、コミットを行います。

「hello-5-2.txt」はコミットされました、「forgotten-file.txt」は未追跡のままの状態です。

```
$ git commit -m "second commit"
[master ceb3e81] second commit
 1 file changed, 1 insertion(+)
 create mode 100644 hello-5-2.txt
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    forgotten-file.txt

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

「git commit --amend --no-edit」コマンドで直前のコミットに「ステージし忘れていたファイル」を追加する

「git commit --amend」コマンドを使用して、「forgotten-file.txt」も先ほどのコミットに含めてしまいましょう。

まずは「forgotten-file.txt」をステージします。

```
$ git add forgotten-file.txt
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   forgotten-file.txt
```

そして、先ほどのコミットをやり直します。今回はコミットメッセージはそのまま良さうなので、「--no-edit」オプションを使用してみます。

以下のように「--no-edit」オプションを使用すれば、コミットメッセージは変更せずにコミットの内容だけを書き換えられます。

```
$ git commit --amend --no-edit
```

修正結果

コミットログを確認すると、2回目のコミットに「forgotten-file.txt」の追加が含まれていることを確認できます。

```
$ git log -p
commit 92a1228b0d82cff312d10698ca752902ef8e0382
Author: Shingo Hiraya <shingohiraya@example.com>
Date:   Thu Aug 11 23:23:00 2016 +0900

second commit

diff --git a/forgotten-file.txt b/forgotten-file.txt
new file mode 100644
index 0000000..295869f
--- /dev/null
+++ b/forgotten-file.txt
@@ -0,0 +1 @@
+forgotten
diff --git a/hello-5-2.txt b/hello-5-2.txt
new file mode 100644
index 0000000..2d6d8c6
--- /dev/null
+++ b/hello-5-2.txt
@@ -0,0 +1 @@
+hello-5-2
```

```
commit 1437f48aa1755c6a5814961e702df881530c3127
Author: Shingo Hiraya <shingohiraya@example.com>
Date: Thu Aug 11 22:56:18 2016 +0900

first commit

diff --git a/hello-5.txt b/hello-5.txt
new file mode 100644
index 0000000..e965047
--- /dev/null
+++ b/hello-5.txt
@@ -0,0 +1 @@
+Hello
```

ファイルの変更の取り消し

「実験的にファイルを変更したけど、元の状態に戻したい」という状況はよくあります。ファイルを Git で管理しているなら、変更を元に戻すのは簡単です。

このセクションでは、ステージされていないファイルの変更を取り消す方法を解説していきます。

【準備】任意のファイルを Git の管理下に置く

まずは、任意のファイルが Git によって管理されるようにします。ここでは以下のように「hello-5-3.txt」を追加し、ステージとコミットを行います。

```
$ echo hello > hello-5-3.txt
$ git add hello-5-3.txt
$ git commit -m "add hello-5-3.txt"
[master 1a7b6ad] add hello-5-3.txt
 1 file changed, 1 insertion(+)
 create mode 100644 hello-5-3.txt
$ git status
On branch master
nothing to commit, working directory clean
今、「hello-5-3.txt」の内容は以下のようになっています。

$ cat hello-5-3.txt
hello
```

ファイルを変更する

ここで「hello-5-3.txt」にテキスト「goodbye」を追加します。

```
$ echo goodbye >> hello-5-3.txt
```

1行追加されました。

```
$ cat hello-5-3.txt
hello
goodbye
```

ここで、「git status」コマンドを実行してみます。作業ディレクトリ上の「hello-5-3.txt」は変更されたけど、その変更内容はステージされていない状態になっていることが分かります。

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   hello-5-3.txt

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

「git checkout -- {ファイル名}」コマンドで変更を取り消す

ここで、「hello-5-3.txt」に対して行った変更を取り消してみましょう。使用するコマンドは、「git status」コマンドを実行した際に表示されたメッセージ内に書かれています。

```
- use "git checkout -- <file>..." to discard changes in working directory
```

以下のように「git checkout -- {ファイル名}」コマンドを使用すれば、「特定のファイルのステージされていない変更」を取り消すことができます。

```
$ git checkout -- hello-5-3.txt
$ git status
On branch master
nothing to commit, working directory clean
```

修正結果

ファイルの変更（2行目の追加）が取り消されていることを確認できます。

```
$ cat hello-5-3.txt
hello
```

ファイルの変更の取り消しにおける注意点

ファイルの変更の取り消しに関して、1つだけ注意点があります。

「git checkout -- {ファイル名}」コマンドを実行すると、ファイルに対して行った変更は完全に消えてしまいます。「ファイルの変更の取り消しの取り消し」はできないので、ファイルの変更が確実に不要だと分かっている場合のみ実行しましょう。

本稿で紹介した Git の基本的なコマンド一覧

本稿では「コミットのやり直し」と「ファイルの変更の取り消し」の手順を解説しました。

本記事で初めて登場した Git のコマンドは以下の通りです。

- **git commit --amend** : 直前のコミットをやり直す
- **git commit --amend --no-edit** : コミットメッセージを変更せずに直前のコミットをやり直す
- **git checkout -- {ファイル名}** : 特定のファイルのステージされていない変更を取り消す

次回の記事では「git reset」コマンドを使用して作業のやり直しを行う方法を解説する予定です。お楽しみに!

参考書籍

『Pro Git』(written by Scott Chacon and Ben Straub and published by Apress)

06. 「soft でも hard でも HEAD とブランチを付けたまま」 —git reset で作業の取り消し

(2016年10月27日)

本連載では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。今回は、「git reset」コマンドを使用して作業の「やり直し」や「取り消し」を行う方法を、「--soft」「--mixed」「--hard」オプションを使用して解説。

「mixed」もある

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終える頃には、GitやGitHubの基本的な操作が身に付いた状態になっていると思います。

前回の記事「Gitコミット現場あるある——やり直し、取り消し、変更したいときに使えるコマンド」では「git commit --amend」や「git checkout -- {ファイル名}」コマンドを使用して作業の「やり直し」や「取り消し」を行う方法を解説しました。

連載第6回目の本稿では「git reset」コマンドを使用して作業の「やり直し」や「取り消し」を行う方法を、「--soft」「--mixed」「--hard」オプションを使用して解説します。

「git reset」コマンドの概要

「git reset」コマンドの動作で鍵となるのは「HEAD」です。

本連載の第3回目の記事「ポイント嫌いでも分かるGitブランチの基本——作成、確認、切り替え、masterにマージ、削除」で説明した通り、HEADは現在のブランチを指すポインタです。そして、ブランチはそのブランチの最新のコミットを指すポインタです。

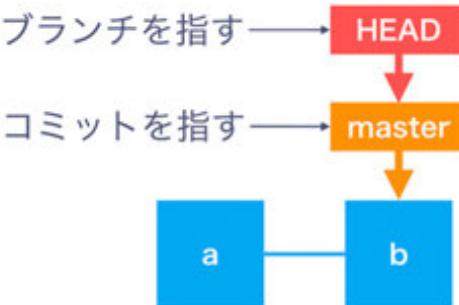


図 1 HEAD と ブランチ の 実体

コミットを重ねるごとにブランチが指すコミットと HEAD が指すブランチが変化していきます。

HEADが指すブランチ、 ブランチが指すコミットが変化する

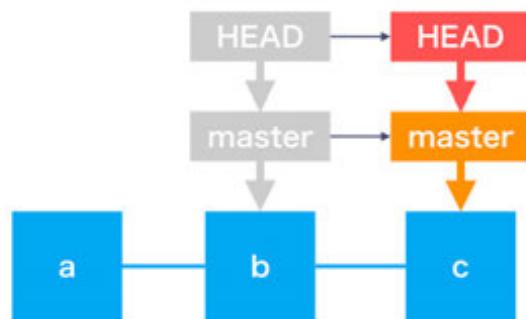


図 2 コミットによって HEAD と ブランチ が 移動する 様子

「git reset」コマンドを使えば、HEAD が指すブランチを移動させることができます。ブランチが指すコミットが変わることになり、これによって作業のやり直しを行えます。

HEADが指すブランチが移動

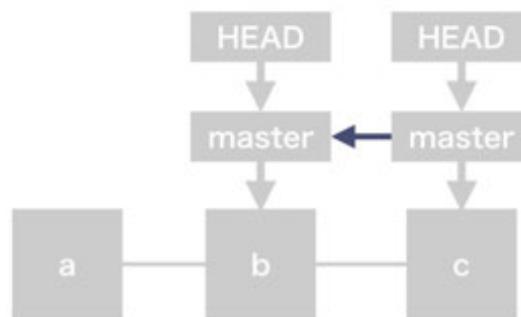


図 3 「git reset」コマンドによって HEAD が指すブランチが移動する 様子

例えば、HEAD が指すブランチを 1 つ前に移動させれば、最新のコミットを「コミット履歴」から消すことができます。

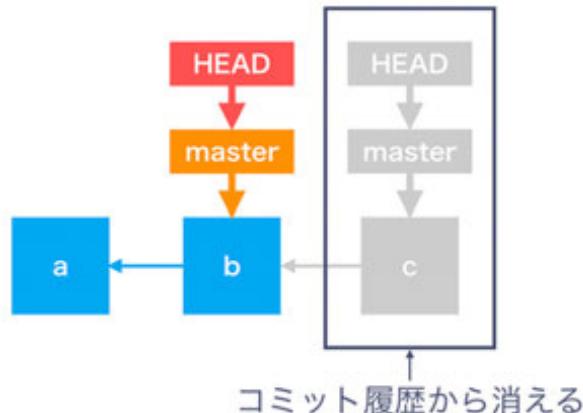


図4 「git reset」コマンドによってコミット履歴からコミットを破棄する様子

「git {任意のコマンド名} -h」でコマンドの使い方が分かる

ここで「git reset」コマンドの使い方を軽く見ておきましょう。「git {任意のコマンド名} -h」コマンドを使用すれば、指定したコマンドの使い方を表示できます。

「git reset」コマンドの使い方を表示する場合は以下のように「git reset -h」コマンドを使用します。

```
$ git reset -h
usage: git reset [--mixed | --soft | --hard | --merge | --keep] [-q] [<commit>]
  or: git reset [-q] <tree-ish> [--] <paths>...
  or: git reset --patch [<tree-ish>] [--] [<paths>...]
  -q, --quiet            be quiet, only report errors
  --mixed                reset HEAD and index
  --soft                 reset only HEAD
  --hard                 reset HEAD, index and working tree
  --merge                reset HEAD, index and working tree
  --keep                 reset HEAD but keep local changes
  -p, --patch             select hunks interactively
  -N, --intent-to-add    record only the fact that removed paths will be added later
```

基本型は「git reset {オプション} {コミット}」であり、「HEAD が指すブランチ」が指定した「コミット」を指すようになります。

また、指定するオプションによって動作が変わります。本稿では「--soft」「--mixed」「--hard」の3つのオプションの動作を試していきます。

「--soft」オプションを使用してコミットをまとめる

「git reset」コマンドを試していきましょう。まずは、「--soft」オプションを試してみます。

「--soft」オプションを指定した場合、HEAD が指すブランチの移動だけが行われます。ステージングエリアや作業ディレクトリは操作されません。この動作を利用して 2 つのコミットを 1 つのコミットにまとめる手順を解説していきます。

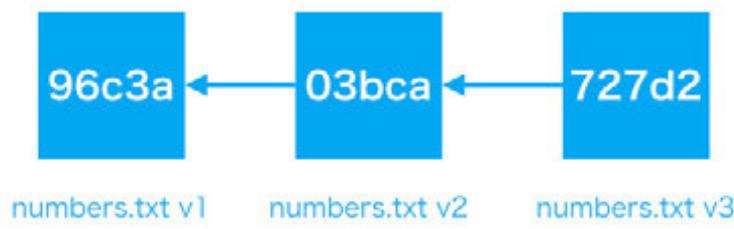


図 5 コミットをまとめる操作を行う前のコミット履歴

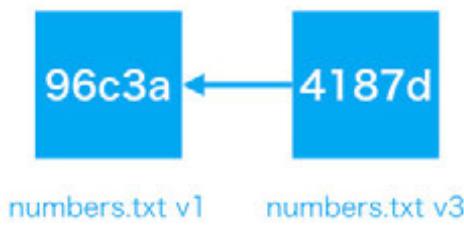


図 6 コミットをまとめる操作を行った後のコミット履歴

準備

「git reset」コマンドを試すための環境を作ります。

適当なディレクトリに移動し、Git リポジトリを作成します。そして、コミットを 3 回行います。

```
$ cd /Users/hirayashingo/Documents/hello-git-6
$ git init
Initialized empty Git repository in /Users/hirayashingo/Documents/hello-git-6/.git/

$ echo one > numbers.txt
$ git add numbers.txt
$ git commit -m "first commit"
[master (root-commit) 96c3a6a] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 numbers.txt

$ echo two >> numbers.txt
$ git add numbers.txt
$ git commit -m "second commit"
[master 03bcaed] second commit
```

```
1 file changed, 1 insertion(+)
```

```
$ echo three >> numbers.txt
$ git add numbers.txt
$ git commit -m "third commit"
[master 727d2f8] third commit
1 file changed, 1 insertion(+)
```

ここで、テキストファイルの内容とコミットログを確認してみます。コミットごとに1行ずつ文字列が追加されるコミット履歴が作られました。

```
$ cat numbers.txt
one
two
three

$ git log -p
commit 727d2f8ef84f396acdf38647b0063931d0d8dece
Author: Shingo Hiraya <shingohiraya@example.com>
Date:   Thu Oct 20 07:45:55 2016 +0900

    third commit
```

```
diff --git a/numbers.txt b/numbers.txt
index 814f4a4..4cb29ea 100644
--- a/numbers.txt
+++ b/numbers.txt
@@ -1,2 +1,3 @@
one
two
+three

commit 03bcaed4ebbba1c296f19efbc5ca0e39454dbc34
Author: Shingo Hiraya <shingohiraya@example.com>
Date:   Thu Oct 20 07:45:38 2016 +0900
```

```
    second commit

diff --git a/numbers.txt b/numbers.txt
index 5626abf..814f4a4 100644
--- a/numbers.txt
+++ b/numbers.txt
@@ -1 +1,2 @@

```

```
one
+two

commit 96c3a6adf606e2219e31a3911e73f8f65983113a
Author: Shingo Hiraya <shingohiraya@example.com>
Date:   Thu Oct 20 07:45:24 2016 +0900

first commit

diff --git a/numbers.txt b/numbers.txt
new file mode 100644
index 0000000..5626abf
--- /dev/null
+++ b/numbers.txt
@@ -0,0 +1 @@
+one
```

コミットをまとめる

「git reset --soft」コマンドを使用して、コミット「727d2f8」「03bcaed」を1つのコミットにまとめます。

現在、HEADはmasterを指し、masterはコミット「727d2f8」を指しています。

```
$ git log --oneline --decorate
727d2f8 (HEAD -> master) third commit
03bcaed second commit
96c3a6a first commit
```

「git reset --soft HEAD~2」コマンドを使用してmasterが指すコミットを2つ前のコミットに移動させます。
「HEAD~2」はHEADの2つ前のコミットを指します。今回は「96c3a6a」を指します。

```
$ git reset --soft HEAD~2
```

masterが2つ前のコミット(1番目のコミット)を指すようになりました。コミット「727d2f8」と「03bcaed」はコミット履歴から消えました。

```
$ git log --oneline --decorate
96c3a6a (HEAD -> master) first commit
```

ステータスを確認すると、numbers.txtに対する変更はステージングエリアに残っています。「準備」で行った変更(2~3行目の追加)がコミットされていない状態に戻っています。

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   numbers.txt
```

ここでコミットを実行すれば、もともと 2 つのコミットに分かれていた変更が 1 つのコミットにまとめます。

```
$ git commit -m "add two and three"
[master 4187dcf] add two and three
  1 file changed, 2 insertions(+)
```

作業ディレクトリがクリーンになり、コミット履歴が 2 件になりました。

```
$ git status
On branch master
nothing to commit, working directory clean

$ git log --oneline --decorate
4187dcf (HEAD -> master) add two and three
96c3a6a first commit
```

「git reset」コマンドと「--soft」オプションを使用してコミットをまとめる操作はこれで完了です。

「--mixed」オプションを使用してステージングエリアを巻き戻し、コミットをやり直す

次は「--mixed」オプションを試します。「『--soft』オプションを使用してコミットをまとめる」で使用したリポジトリに対して引き続き操作を行っていきます。

「--mixed」オプションを指定した場合、「--soft」オプションを指定した場合の動作に加え、「**ステージングエリアを指定コミット時の状態と一致させる**」処理が実行されます。作業ディレクトリの状態は変わらないので、HEAD とステージングエリアの状態だけを巻き戻したい場合に使えます。

この動作を利用してステージングエリアを 1 つ前のコミットの状態まで戻し、再度コミットを行う手順を解説していきます。



図 7 ステージングエリアの書き戻しとコミットのやり直しを行う前のコミット履歴
numbers.txt v1 numbers.txt v3 numbers.txt v4



図 8 ステージングエリアの書き戻しとコミットのやり直しを行った後のコミット履歴
numbers.txt v1 numbers.txt v3 numbers.txt v5

準備

「--mixed」オプションを試すために numbers.txt に少し変更を加えます。4 行目「four」を追加し、ステージとコミットを行います。

```
$ echo four >> numbers.txt
$ git add numbers.txt
$ git commit -m "add four"
[master d25d95f] add four
 1 file changed, 1 insertion(+)
```

ここで、ステータスとコミットログを確認してみます。3 つ目のコミットが追加されています。

```
$ git status
On branch master
nothing to commit, working directory clean

$ git log --oneline --decorate
d25d95f (HEAD -> master) add four
4187dcf add two and three
96c3a6a first commit
```

ステージングエリアを書き戻す

準備ができましたので、「--mixed」オプションを試してみます。

「準備」で 4 行目「four」を追加しましたが、ここで気が変わって最新のコミットに「5 行目の文字列追加」という変更も含めたくなったりします。「--mixed」オプションとともに「git reset」コマンドを使用して「HEAD の移動」と「ステージングエリアのリセット」を実行し、5 行目の文字列追加とコミットを行います。

「--mixed」オプションはデフォルトのオプションなので明示的に指定する必要はありません。実際は以下のように「git reset {コミット}」コマンドを使用します。また、「HEAD~」は1つ前のコミットを指します。

```
$ git reset HEAD~  
Unstaged changes after reset:  
M numbers.txt
```

「リセットしたので、numbers.txtへの変更がステージングエリアから取り除かれた」旨のメッセージが表示されました。

ステータスとコミットログを表示してみると、master がコミット「4187dcf」を指すようになり、numbers.txtへの変更がステージされていない状態になったことを確認できます。

```
$ git status  
On branch master  
Changes not staged for commit:  
(use "git add <file>..." to update what will be committed)  
(use "git checkout -- <file>..." to discard changes in working directory)  
  
modified:   numbers.txt  
  
no changes added to commit (use "git add" and/or "git commit -a")  
  
$ git log --oneline --decorate  
4187dcf (HEAD -> master) add two and three  
96c3a6a first commit
```

5行目「five」を追加し、ステージとコミットを行います。

```
$ echo five >> numbers.txt  
$ git add numbers.txt  
$ git commit -m "add four and five"  
[master 4896491] add four and five  
1 file changed, 2 insertions(+)
```

作業ディレクトリがクリーンになり、コミット履歴が3件になりました。

```
$ git status  
On branch master  
nothing to commit, working directory clean
```

```
$ git log --oneline --decorate
4896491 (HEAD -> master) add four and five
4187dcf add two and three
96c3a6a first commit
```

「--mixed」オプションを使用してステージングエリアを巻き戻し、コミットをやり直す操作はこれで完了です。

「--hard」オプションを使用して直前のコミットを破棄する

最後に「--hard」オプションを試します。「『--mixed』オプションを使用してステージングエリアを巻き戻し、コミットをやり直す」で使用したリポジトリに対して引き続き操作を行っていきます。

「--hard」オプションを指定した場合、「--mixed」オプションを指定した場合の動作に加え、「**作業ディレクトリを指定コミット時の状態と一致させる**」処理が実行されます。

例えば「実験的な変更をコミットしたけどその変更が不要になった場合」などに使用できます。1度もコミットしていない変更は取り戻すことができないので、「--hard」オプションは取り扱いを注意する必要があります。

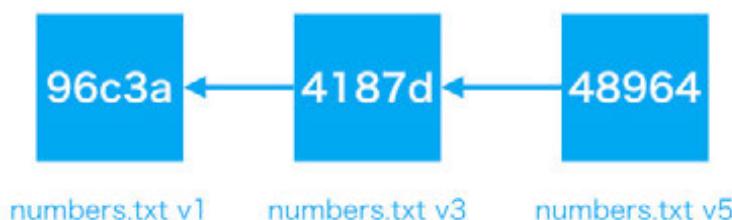


図9 直前のコミットを破棄する操作を行う前のコミット履歴

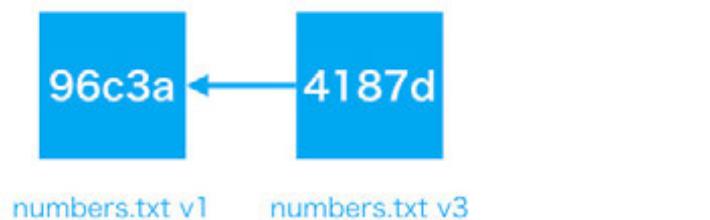


図10 直前のコミットを破棄する操作を行った後のコミット履歴

それでは「--hard」オプションを使用して、直前のコミットを破棄する操作を行ってみます。現在のコミット履歴は以下のようになっています。

```
$ git log --oneline --decorate
4896491 (HEAD -> master) add four and five
4187dcf add two and three
96c3a6a first commit
```

「git reset --hard HEAD~」コマンドを使用して、最新のコミット「4896491」を破棄します。master が 1 つ前のコミット「4187dcf」を指すようになり、ステージングエリアと作業ディレクトリがコミット「4187dcf」を記録したときの状態になります。

```
$ git reset --hard HEAD~  
HEAD is now at 4187dcf add two and three
```

コミット履歴を表示すると、コミット「4896491」がコミット履歴から消え、master がコミット「4187dcf」を指すようになったことを確認できます。

```
$ git log --oneline --decorate  
4187dcf (HEAD -> master) add two and three  
96c3a6a first commit
```

また、ステージングエリアと作業ディレクトリの両方がコミット「4187dcf」を記録したときの状態にリセットされています。

```
$ git status  
On branch master  
nothing to commit, working directory clean  
  
$ cat numbers.txt  
one  
two  
three
```

「git reset」コマンドと「--hard」オプションを使用して直前のコミットを破棄する操作はこれで完了です。

本稿で紹介した Git のコマンド一覧

本稿では「git reset」コマンドを使用して作業の「やり直し」や「取り消し」を行う方法を解説しました。

本記事で初めて登場した Git のコマンドは以下の通りです。

- **git reset --soft {コミット}** : 指定したコミットまで HEAD を移動する
- **git reset (--mixed) {コミット}** : 「git reset --soft」を行ったときの動作に加え、ステージングエリアを指定コミット時の状態と一致させる

- **git reset --hard {コミット}:** 「git reset (--mixed)」を行ったときの動作に加え、作業ディレクトリを指定コミット時の状態と一致させる
- **git {任意のコマンド名} -h:** 指定したコマンドの使い方を表示する

本稿で紹介した「git reset」コマンドの使い方は一部に過ぎません。ぜひ、機会がありましたら他の使い方も試してみてください。

次回は GitHub を使い始めるための環境構築の手順を解説する予定です。お楽しみに!

参考書籍

『Pro Git』(written by Scott Chacon and Ben Straub and published by Apress)

07. はじまりはいつもプルリク? Git リポジトリホスティングサービス GitHub とは

(2016年12月05日)

本連載では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。今回から、GitHubの使い方を紹介。まずは、GitHubの概要と使い始めるための準備、SSH設定についてです。

Git と GitHub の使い方を解説する連載。今回から GitHub 編開始!

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終える頃には、GitやGitHubの基本的な操作が身に付いた状態になっていると思います。

前回の『「soft でも hard でも HEAD とブランチを付けたまま」——git reset で作業の取り消し』までは、Gitのインストールから実践的なコマンドの使い方を解説してきました。連載第7回目の本稿では「GitHub」を使い始めるための準備を解説します。この機会にGitHubのアカウント作成や初期設定にチャレンジしてみてはいかがでしょうか。

GitHub とは

GitHubはGitリポジトリのホスティングサービスです。本連載では、これまでにPC上に作成したローカルリポジトリに対する操作を解説してきました。GitHubのアカウントを作成すれば、GitHub上にリポジトリ（リモートリポジトリ）を作成できます。これによって、自分で作ったソースコードを公開したり、自分以外の誰かとソースコードを共有したりすることができるようになります。

GitHubはソースコードを保管するだけのサービスではなく、ソフトウェアの開発者同士が連携するためのさまざまな機能が提供されています。修正したソースコードのチェックを行うための「Pull requests」（プルリク）や開発まわりのタスクなどを管理するための「Issues」などの機能を使用することによって、世界のどこにいてもチームで開発を進めていくことができます。

特に、GitHubを使うユーザーの間では、プルリク（コードレビュー依頼）を使った開発フローは「プルリク駆動開発」と呼ばれ、人気があります。

なお、本連載で扱う「GitHub」は主に「GitHub.com」という GitHub 社が提供するインフラ上のサービスです。GitHub でコードをレビューしたり、管理したりする範囲を企業内にとどめたい、アクセス制御をしたいなどのエンタープライズのニーズに答えるために、GitHub をオンプレミスや AWS など好みのインフラ環境にインストールして使う GitHub Enterprise もあります。

GitHub のサービスそのものについては、下記もご参考ください。

- 開発者からの強い支持、5年弱で300万ユーザーを突破:共同創業者に聞いた、GitHubは何が違ったのか?
- GitHub 創設者が語る“軌跡”: GitHubが実践するオープンソース式マネジメント法
- GitHubはリアルRPG?そして、ソーシャルコーディングへ

アカウント作成

GitHub を使い始めるための準備を整えていきましょう。まずは GitHub のアカウントを作成します。

GitHub のトップページをブラウザで開き、「Sign up」をクリックします。



図 1 GitHub トップページ

アカウント情報の入力

アカウント情報の入力フォームが表示されます。ユーザー名、メールアドレス、パスワードを入力し「Create an account」ボタンをクリックします。

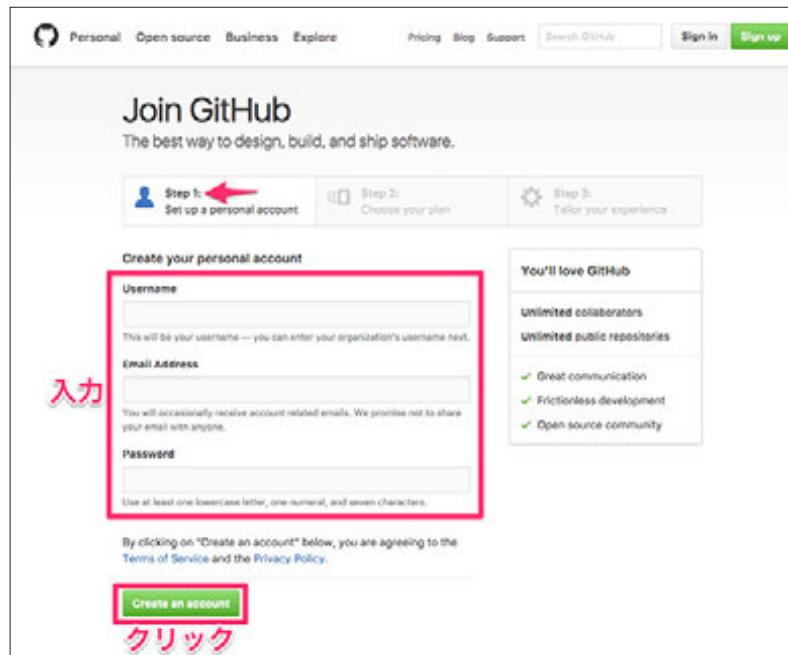


図 2 アカウント情報入力ページ

プランを選択

アカウントの作成が完了すると「Step 2」ページが表示されます。

このページではプランを選択できます。プランは後からでも変更できるので、デフォルトで選択されているプラン（無料プラン）を選択したまま「Continue」ボタンをクリックします。

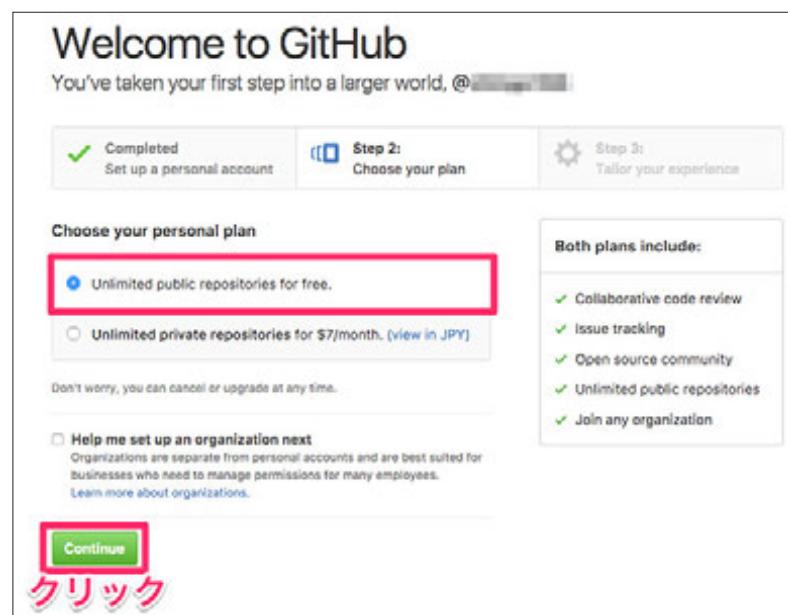


図 3 プラン選択ページ

「Step 3」ページが表示されます。「skip this step」ボタンをクリックします。

Welcome to GitHub

You'll find endless opportunities to learn, code, and create.

<input checked="" type="checkbox"/> Completed Set up a personal account	<input type="checkbox"/> Step 2: Choose your plan	<input type="checkbox"/> Step 3:
--	--	----------------------------------

How would you describe your level of programming experience?

Totally new to programming Somewhat experienced Very experienced

What do you plan to use GitHub for? (check all that apply)

Design Project Management Research
 Development School projects Other (please specify)

Which is closest to how you would describe yourself?

I'm a professional I'm a hobbyist I'm a student
 Other (please specify)

What are you interested in?

e.g. tutorials, android, ruby, web-development, machine-learning, open-source

Submit **skip this step**

図 4 アンケートページ

メールアドレスの確認

次のページが表示されます。「Start a project」ボタンをクリックします。



図 5 アカウント追加直後のダッシュボードページ

次のページでは「メールアドレスの確認を行ってください」という旨のメッセージが表示されます。アカウント作成時に入力したメールアドレスに確認メールが届いているはずなのでメールボックスを確認しましょう。

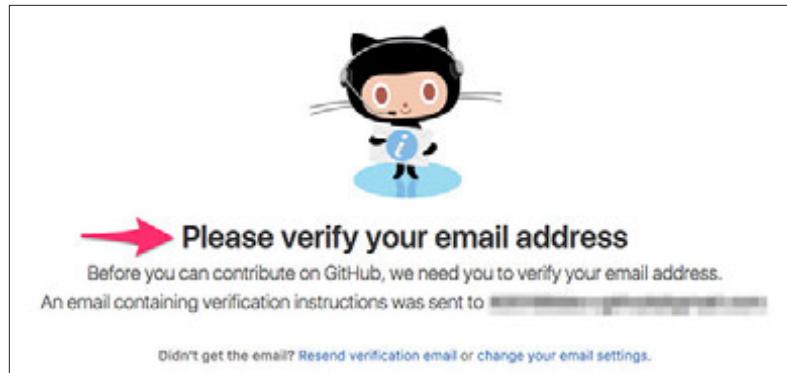


図 6 メールアドレス確認要求メッセージ

GitHub からのメールを表示し、本文内のリンク「Verify email address」をクリックします。

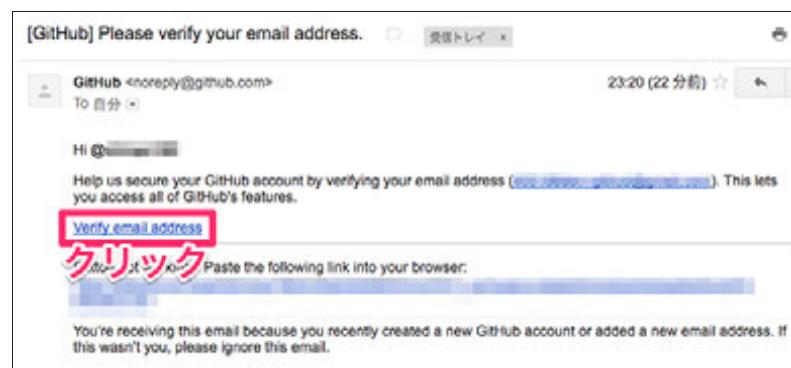


図 7 メールアドレス確認メール

「メールアドレスの確認ができた」という旨のメッセージが表示されます。アカウント作成はこれで完了です。

「Start a project」ボタンを再びクリックしてみます。



図 8 メールアドレス確認完了メッセージ

リポジトリ作成ページ

今度はリポジトリ作成ページへ遷移できました。GitHub 上に Git リポジトリを作成する準備が整いました。GitHub 上に Git リポジトリを作成する手順は次回以降の記事で解説します。

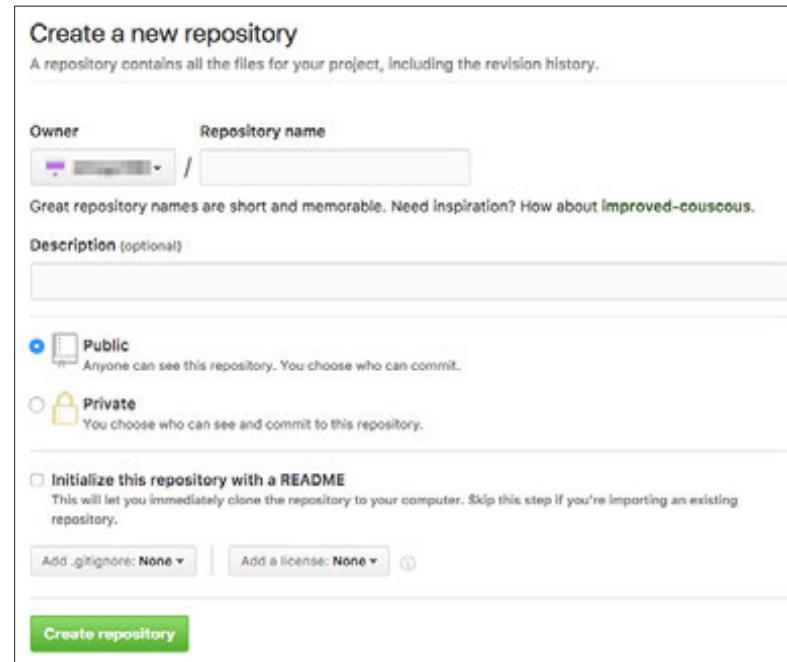


図 9 リポジトリ作成ページ

アカウントの設定

次にアカウントの初期設定を行います。任意のページのヘッダー部分右端のアイコンをクリックし、「Settings」を選択します。

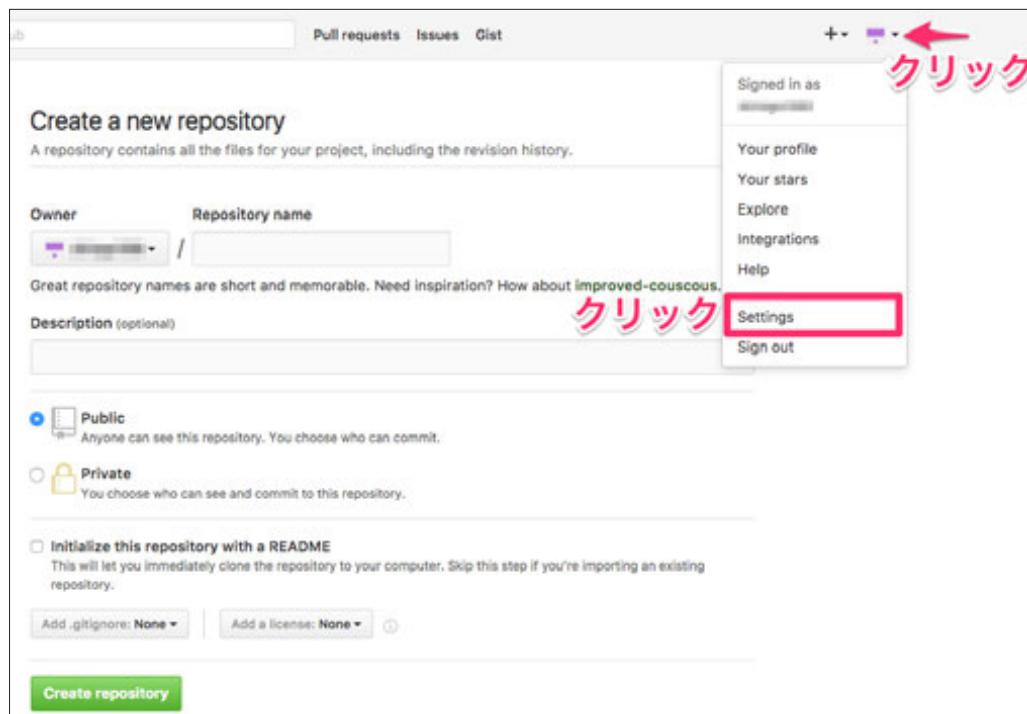


図 10 メニュー

アカウント設定ページが表示されました。このページではプロフィール、アカウント情報、通知、SSHなどの設定を行えます。まずは、デフォルトで表示されるプロフィール設定を行ってみます。

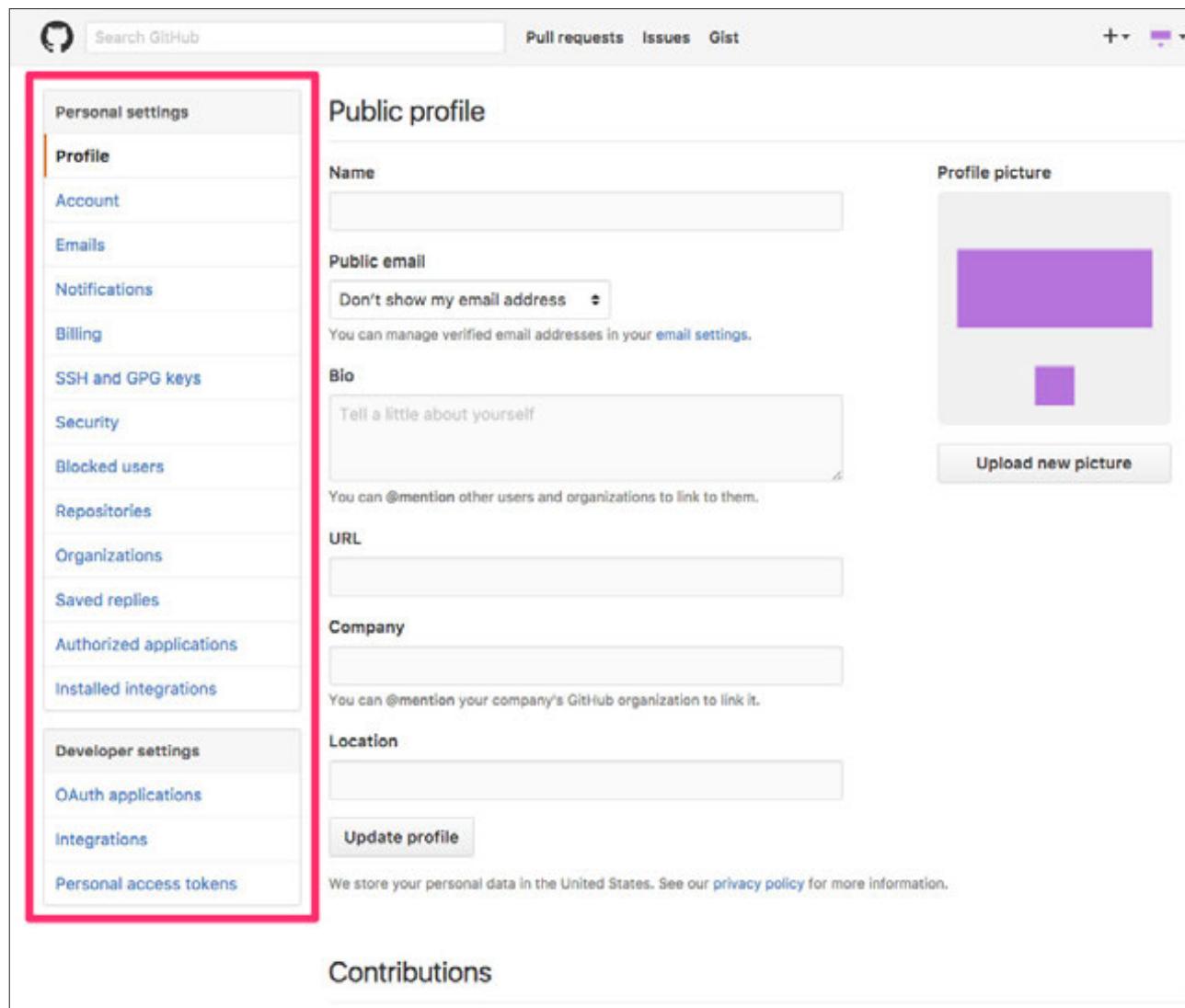


図 11 アカウント設定ページ

プロフィール設定

GitHub には Twitter や Facebook などのようにソーシャル機能があります。例えば、特定のユーザーを「フォロー」するとその人の GitHub 上での活動が「ダッシュボード」に流れています。

アカウント設定ページの「Public profile」セクション内の「Name」「Bio」「URL」などの情報は**他のユーザーに開示する情報**になります。

プロフィール情報の入力は**任意**です。他のユーザーに知ってほしい情報があれば入力しましょう。

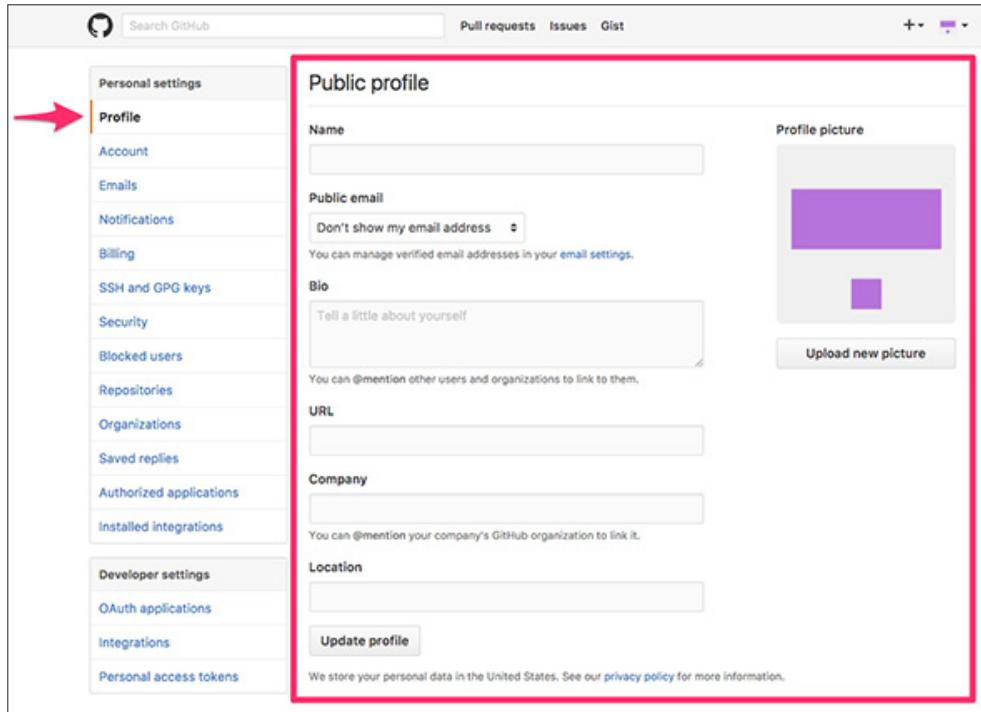


図 12 プロフィール情報入力

プロフィール情報を更新するには、各テキストフィールドに値を変更し「Update profile」をクリックします。

今回は「Name」「Location」を入力し「Update profile」をクリックしてみました。

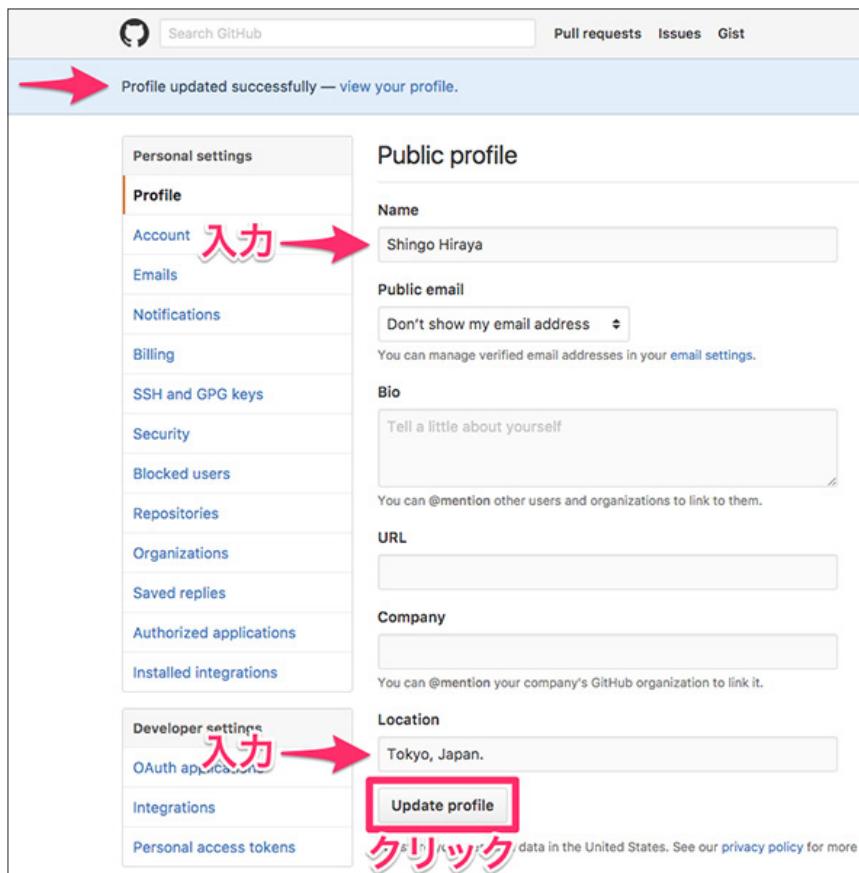


図 13 プロフィール情報入力完了

アイコンの設定

Twitter や Facebook などのようにユーザーを表すアイコンを設定でき、このアイコンは GitHub 上のさまざまなページで表示されます。GitHub アカウント作成すると、アイコンの初期設定はランダム生成される画像になりますが、アカウント設定ページで好きなアイコンに変更できます。

アイコンを変更するにはページ右側の「Upload new picture」をクリックし、表示される指示に従って画像アップロードを行います。

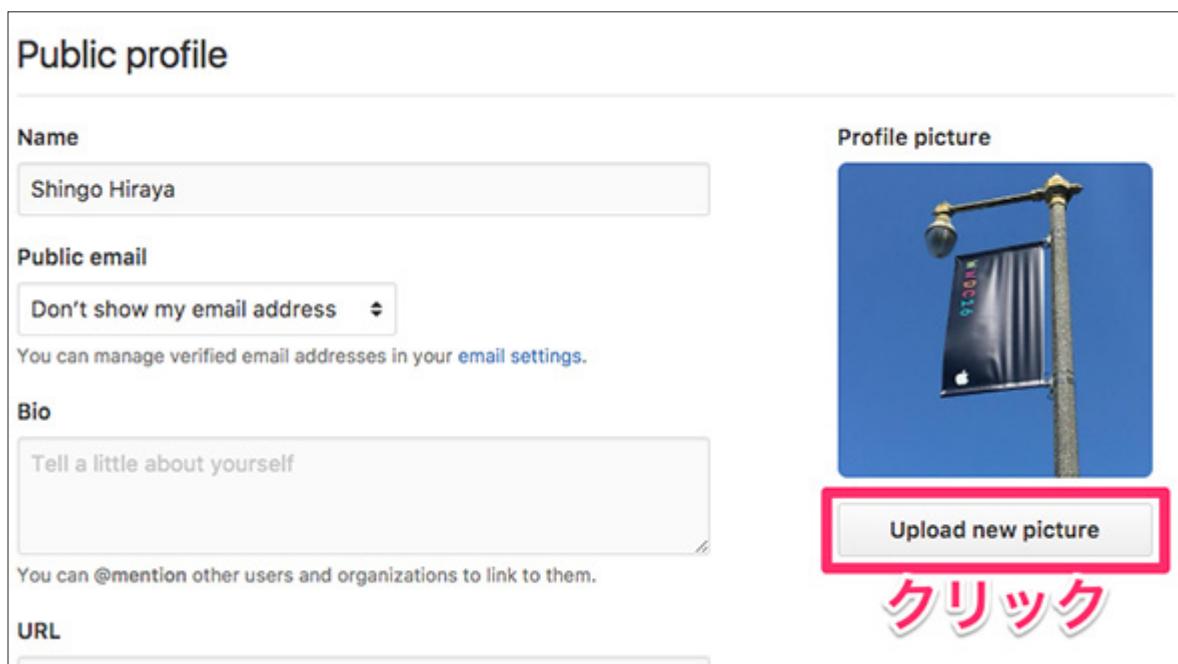


図 14 アイコン変更完了

今回はプロフィール情報とアイコンの設定を変更しました。

自分のプロフィールページを表示する

ここで自分のプロフィールページを表示して、他のユーザーからどのように見えるかを確認してみます。

自分のプロフィールページを表示するには、ヘッダー部分右端のアイコンをクリックし「Your profile」を選択します。

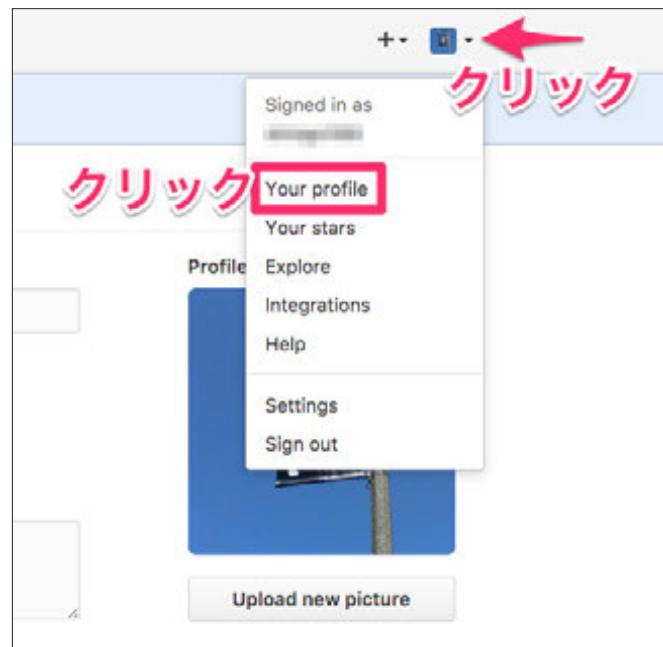


図 15 「Your profile」を選択

プロフィールページが表示されました。先ほど設定した「Name」「Location」「アイコン」が適用されています。

A screenshot of the GitHub profile page for 'Shingo Hiraya'. The profile picture, which is a black Apple laptop screen mounted on a street lamp, is highlighted with a red box. Below the picture, the name 'Shingo Hiraya' is displayed. At the bottom of the profile section, there are buttons for 'Add a bio' and location information ('Tokyo, Japan.') which is also highlighted with a red box. The bio field is currently empty. Below this, it says 'Joined on Nov 16, 2016'. To the right of the profile section, there are tabs for 'Overview', 'Repositories 0', and 'Stars'. Under 'Popular repositories', it says 'You don't have any'. Below that, it shows '1 contribution in the last year' with a calendar heatmap indicating activity. A note at the bottom says 'Learn how we count contributions.'

図 16 プロフィールページ

SSH 設定

最後に、「SSH 公開鍵の作成」と「GitHub への公開鍵登録」を行う手順を解説します。これは、GitHub 上の Git リポジトリに SSH を使って接続するために必要な作業です。

SSH 公開鍵の作成

ターミナル (Windows の場合は Git Bash) を開き、次のコマンドを入力します。「your-email@example.com」は自分のメールアドレスに置き換えてください

```
$ ssh-keygen -t rsa -C "your-email@example.com"
```

公開鍵／秘密鍵が生成されます。

```
Generating public/private rsa key pair.
```

以下のような「ファイルの保存場所を入力してください」という旨のメッセージが表示されたら、何も入力せずに「Enter」キーを押します。

```
Enter a file in which to save the key (/Users/you/.ssh/id_rsa): // Enter キーを押す
```

次に「パスフレーズを入力してください」という旨のメッセージが表示されます。「パスフレーズ」は PC 上の秘密鍵にアクセスするためのパスワードのようなものです。

パスフレーズを設定する場合は指示に従って 2 回パスフレーズを入力します。設定しない場合は 2 回とも何も入力せずに Enter キーを押します。

```
Enter passphrase (empty for no passphrase): // パスフレーズを入力し（または何も入力せずに）Enter キーを押す  
Enter same passphrase again: // 再度パスフレーズを入力し（または何も入力せずに）Enter キーを押す
```

SSH 公開鍵の作成作業は以上で完了です。

GitHub への公開鍵登録

作業「アカウントの設定」の冒頭部分と同じ操作を行ってアカウント設定ページが表示し、「SSH and GPG keys」をクリックします。

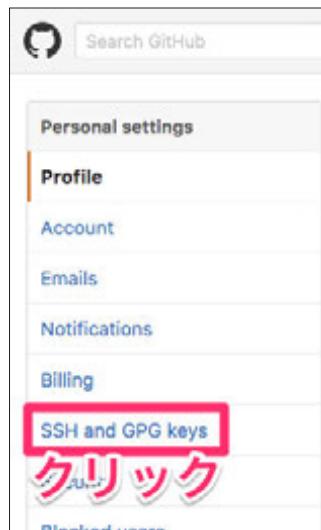


図 17 アカウント設定ページ

次に「New SSH key」をクリックします。

図 18 SSH 設定ページ

「Title」と「Key」を入力する欄が表示されます。「Title」には適当な名前を入力します。

図 19 SSH 公開鍵のタイトル入力

「Key」に入力する値は以下のコマンドを使用して取得できます。コマンド実行後に表示された文字列をクリップボードにコピーしてください。

```
$ cat ~/.ssh/id_rsa.pub
ssh-rsa { 公開鍵の内容 } { メールアドレス } // これをコピー
```

「Key」欄に張り付けて「Add SSH key」をクリックします。

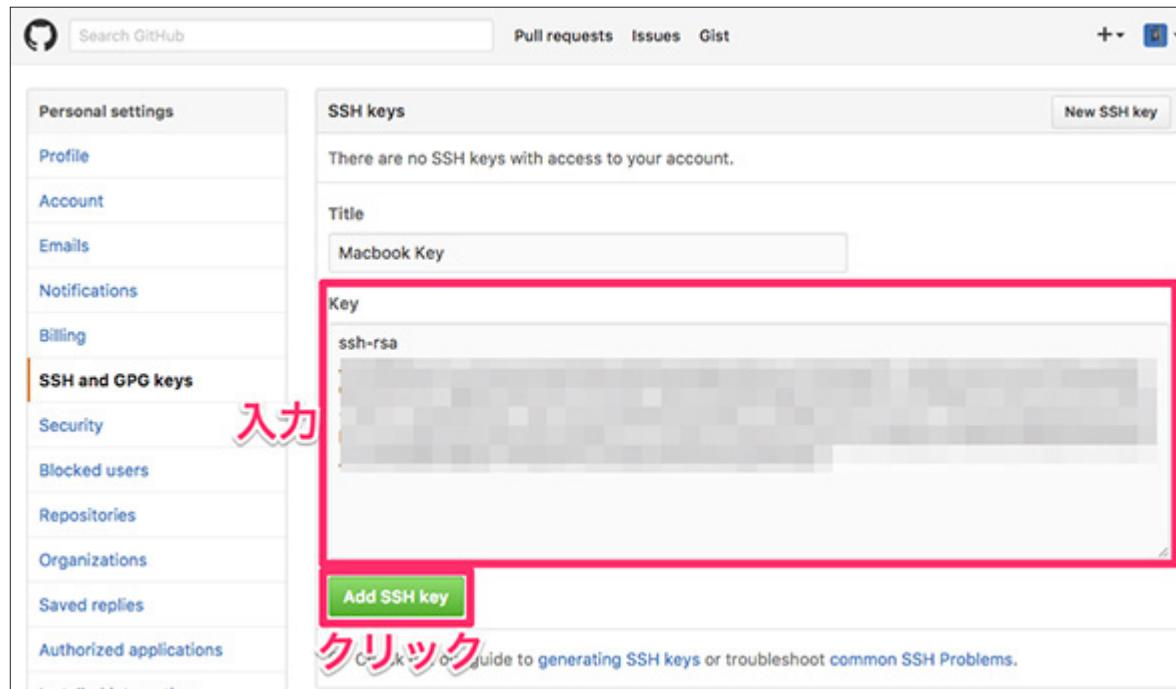


図 20 SSH 公開鍵入力

公開鍵の登録が完了しました。

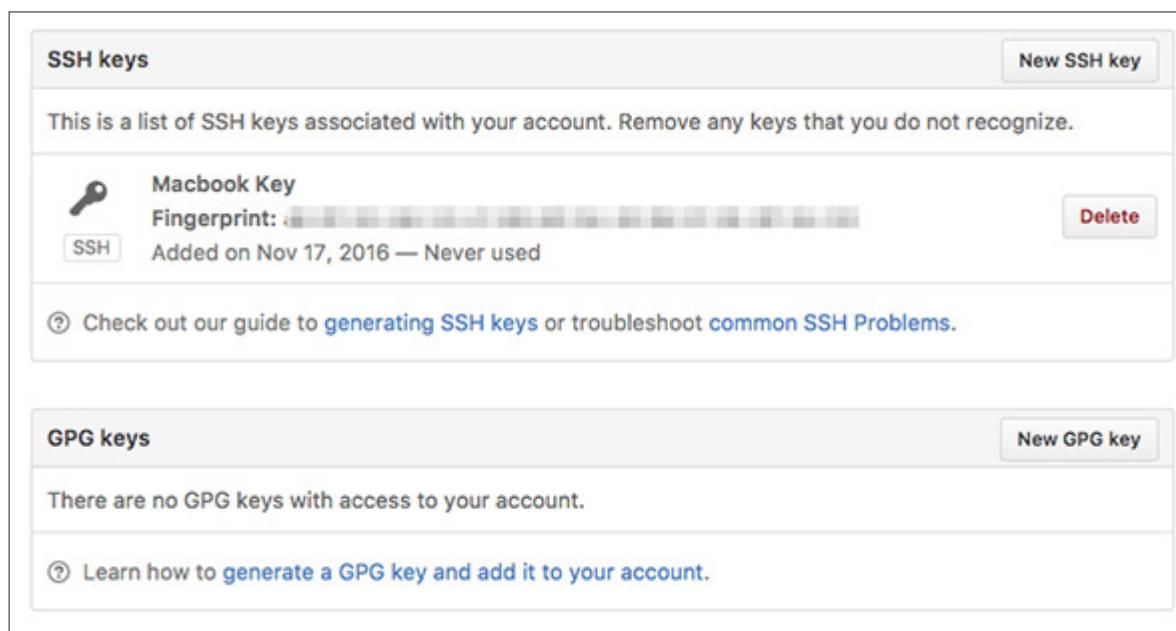


図 21 SSH 公開鍵設定完了

GitHub に公開鍵を登録する作業は、これで完了です。

次回からは GitHub の各機能の概要を解説

本稿では「GitHub」を使い始めるための準備を解説しました。アカウント作成や初期設定は無事に終わりましたでしょうか。

次回からは GitHub の各機能の概要を解説する予定です。お楽しみに!

参考書籍

『Pro Git』(written by Scott Chacon and Ben Straub and published by Apress)

08.2017年、GitHubを始めるために 最低限知っておきたい各機能

(2017年01月05日)

本連載では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。前回から、GitHubの使い方を紹介している。今回は、検索、Pull requests、Issues、Gist、Projects、Pulse、Graphs、ダッシュボードなど、GitHubの各機能の概要を解説します。

GitHubを使い始めてみよう

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終える頃には、GitやGitHubの基本的な操作が身に付いた状態になっていると思います。

前回の記事「はじまりはいつもプルリク? Gitリポジトリホスティングサービス GitHubとは」では「GitHub」を使い始めるための準備を解説しました。

連載第8回目の本稿では、検索、Pull requests、Issues、Gist、Projects、Pulse、Graphs、ダッシュボードなど、GitHubの各機能の概要を解説していきます。

トップページ

まずは、GitHubにログインした直後に表示される「トップページ」から使用できる各機能を解説していきます。

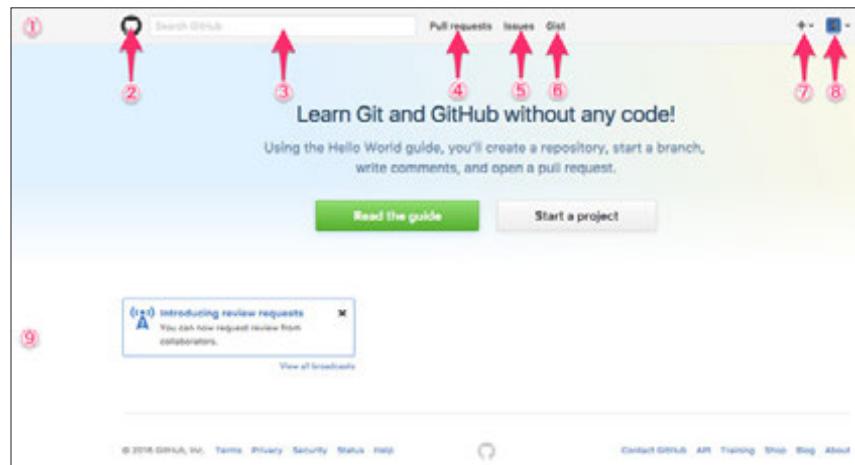


図1 GitHubトップページ

1. ツールバー
2. ロゴ
3. 検索バー
4. Pull requests
5. Issues
6. Gist
7. Create a new ...
8. ユーザーアイコン
9. ダッシュボード

トップページはツールバー（図 1 中の【1】）とダッシュボード（図 1 中の【9】）で構成されています。

【1】ツールバー

「github.com」内のどのページでも常に上部に表示されるコンポーネントです。頻繁に使う機能にアクセスすることができます。

【2】ロゴ

一般的な Web サイトと同様に、クリックするとトップページへ移動できます。

【3】検索バー

GitHub 内を検索する機能を提供します。

キーワードを入力して「Enter」キーを押すとリポジトリやコード、ユーザーなどを検索できます。

Search Search

We've found 63,823 repository results Sort: Best match

	Repositories	Issues
	63,823	
	999,936	
	103,794	
	9,189	
	3,368	

Languages

Language	Count
Swift	46,547
Objective-C	3,410
Python	639
Shell	529
JavaScript	488
Java	422
PHP	416
HTML	410
C	398
Ruby	387

Advanced search Cheat sheet

apple/swift
The Swift Programming Language
C++ ★ 38,632 5,186 Updated 31 minutes ago

carlosencoding/Swift
Reusable apps code. Written in Swift
Swift ★ 2,816 704 Updated 14 days ago

openstack/swift
OpenStack Object Storage (Swift)
Python ★ 1,434 787 Updated 2 days ago

facebook/swift
An annotation-based Java library for creating Thrift serializable types and services.
Java ★ 619 242 Updated on Nov 17

HunkSmile/Swift
Write The Code, Change The World

図 2 キーワード「swift」で検索した場合の結果

[4] Pull requests

クリックすると、自分が関わっているプルリクエストを表示するページへ移動できます。

プルリクエストについては、後ほど解説します。

Search GitHub Pull requests Issues Gist

Created Assigned Mentioned

Visibility Organization Sort

Open	Closed
0	0

No results matched your search.

Use the links above to find what you're looking for, or try a new search query. The Filters menu is also super helpful for quickly finding issues most relevant to you.

ProTip! Type on any issue or pull request to go back to the pull request listing page.

図 3 Pull requests

【5】Issues

クリックすると、自分が関わっているイシューを表示するページへ移動できます。

イシューについても、後ほど解説します。

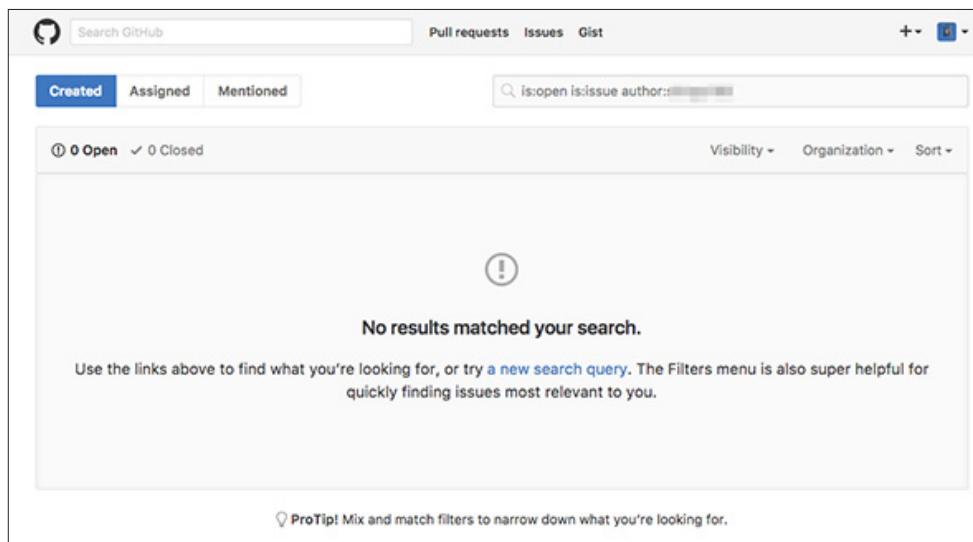


図4 Issues

【6】Gist

クリックすると、Gist のページへ移動できます。

Gist は単一のソースコードやコードの断片などを保存したり、シェアしたりするための機能です。リポジトリを作るほどでもない規模のコードをシェアする場合などに役に立ちます。埋め込み用の HTML を生成できるので、Gist に保存したコードを外部のブログ記事などに埋め込むこともできます。

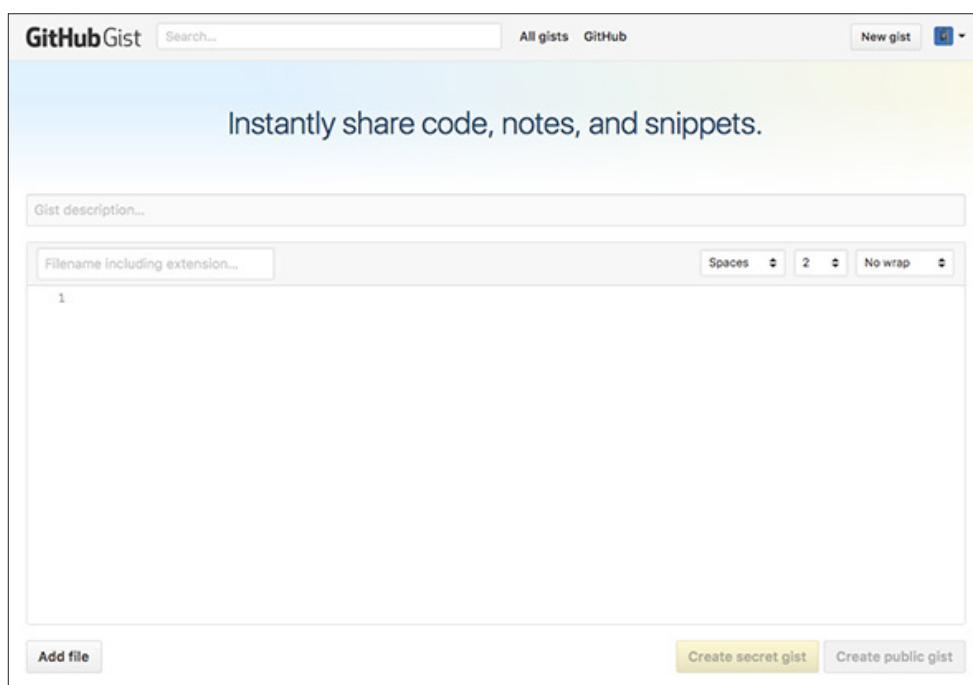


図5 Gist

[7] Create a new ...

クリックすると、リポジトリや Gist を新規作成するためのメニューを表示できます。

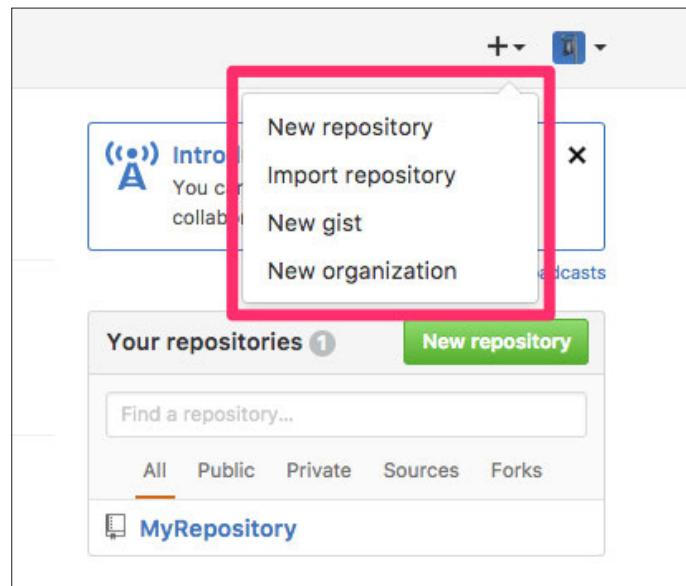


図6 「Create a new ...」をクリックすると表示されるメニュー

[8] ユーザーアイコン

クリックすると、プロフィールページや設定ページへ遷移するためのメニューを表示できます。

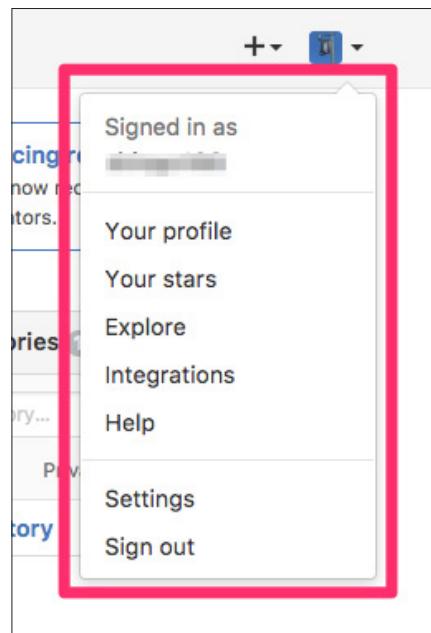


図7 ユーザーアイコンをクリックすると表示されるメニュー

[9] ダッシュボード

自分に関する情報が表示されます。

アカウント作成直後の場合、ユーザーガイドやリポジトリ作成ページへ遷移するためのボタンが表示されます。他のユーザーをフォローしたり、リポジトリを作成したりすると、ここに表示される情報が増えます。

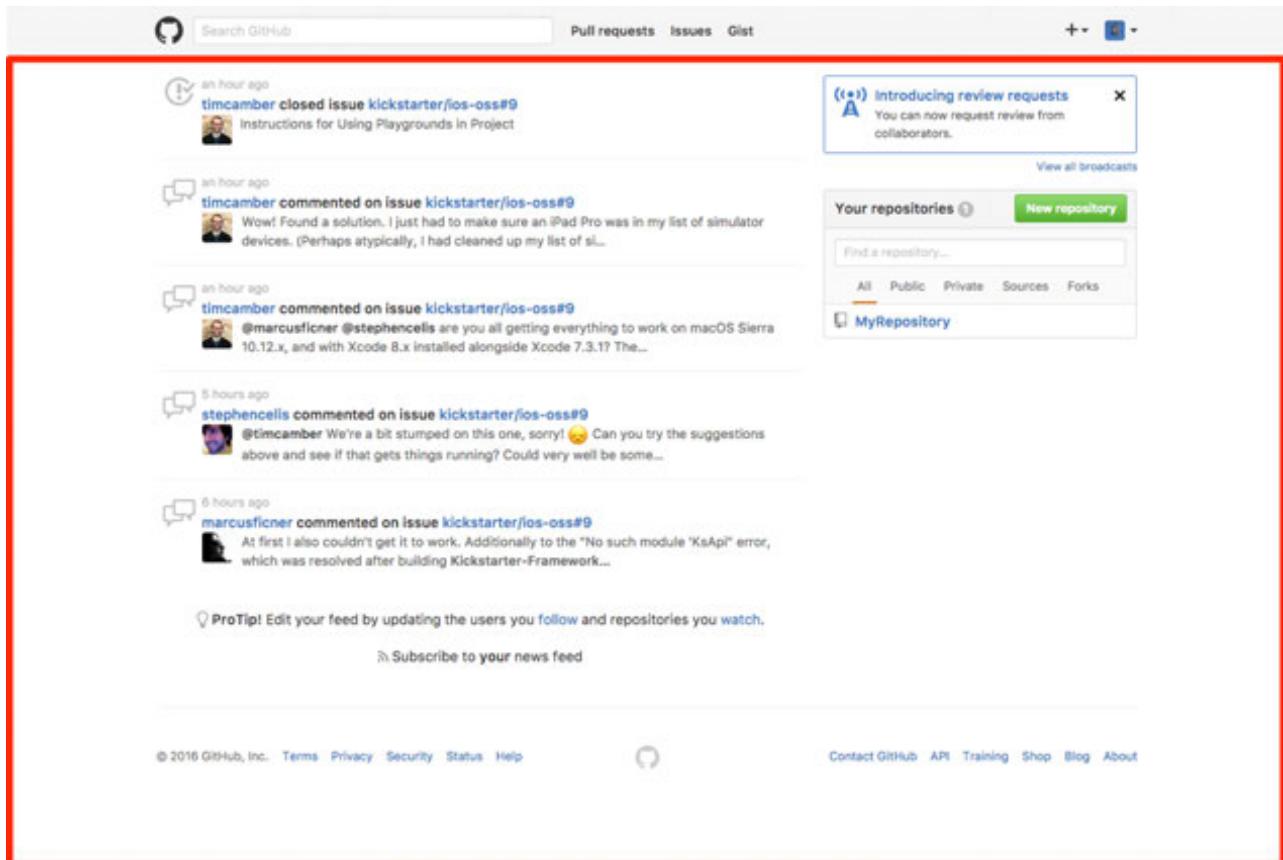


図 8 他ユーザーのフォロー、リポジトリ作成などを行った後のダッシュボード

トップページから使用できる各機能の解説は以上です。

リポジトリページ

次に、GitHub 上のリポジトリごとに作成される「リポジトリページ」から使用できる各機能を解説していきます。

今回は CSS フレームワーク 「Bootstrap」 のリポジトリページを例に解説していきます。

「トップページ」の解説で扱った「検索バー」を使用してキーワード「Bootstrap」で検索し、検索結果の 1 件目 (twbs/bootstrap) をクリックします。

The screenshot shows the GitHub search interface with the query 'Bootstrap' entered in the search bar. A pink box highlights the search term. Below the search bar, a summary states 'We've found 77,207 repository results'. The results list includes a top result for 'twbs/bootstrap' with a pink arrow pointing to its name. Other results include 'angular-ui/bootstrap' and 'jasny/bootstrap'. On the left, there's a sidebar for 'Languages' showing counts for JavaScript, HTML, CSS, PHP, Ruby, and Java.

図 9 キーワード「Bootstrap」で検索した場合の結果

Bootstrap のリポジトリページを表示できました。リポジトリページの構成は以下のようになっています。

The screenshot shows the GitHub repository page for 'twbs/bootstrap'. The page has several sections highlighted with numbered arrows: (1) Repository name 'twbs / bootstrap', (2) Watch count '6,649', Star count '104,586', Fork count '47,483', and a 'Watch' button. (3) Navigation tabs for 'Code', 'Issues 249', 'Pull requests 62', 'Projects 1', 'Pulse', and 'Graphs'. (4) Commit count '15,334 commits', (5) Branch count '20 branches', (6) Release count '40 releases', (7) Contributor count '812 contributors', and (8) License 'MIT'. (9) A summary text: 'The most popular HTML, CSS, and JavaScript framework for developing responsive, mobile first projects on the web.' followed by a link 'http://getbootstrap.com'. The main area displays a list of recent commits.

図 10 Bootstrap のリポジトリページの構成

1. リポジトリ名
2. Watch、Star、Fork
3. タブ
4. Code
5. Issues
6. Pull requests
7. Projects
8. Pulse
9. Graphs

【1】リポジトリ名

スラッシュの左に組織名（またはユーザー名）、右にリポジトリ名が表示されます。

組織名をクリックすると、その組織のページに遷移でき、プロフィールやリポジトリ一覧などを確認できます。

【2】Watch、Star、Fork

リポジトリに対して行える操作「Watch」「Star」「Fork」が行われた数が表示されます。

「Watch」はTwitterの「フォロー」のようなもので、「Star」はFacebookの「いいね！」のようなものです。「Fork」は、他の組織（またはユーザー）のリポジトリを自分のアカウント下に複製する操作です。それぞれの数が多いほどたくさんの人々に注目されることになります。

【3】タブ

特定のリポジトリのページを開いている時に常に上部に表示されます。項目を選択すると、タブの下に表示するコンテンツを切り替えることができます。

【4】Code

リポジトリのページを開くと、デフォルトで Code タブが選択されます。このタブを選択すると、リポジトリの概要、コミット数、リポジトリ内のファイルなどを確認できます。

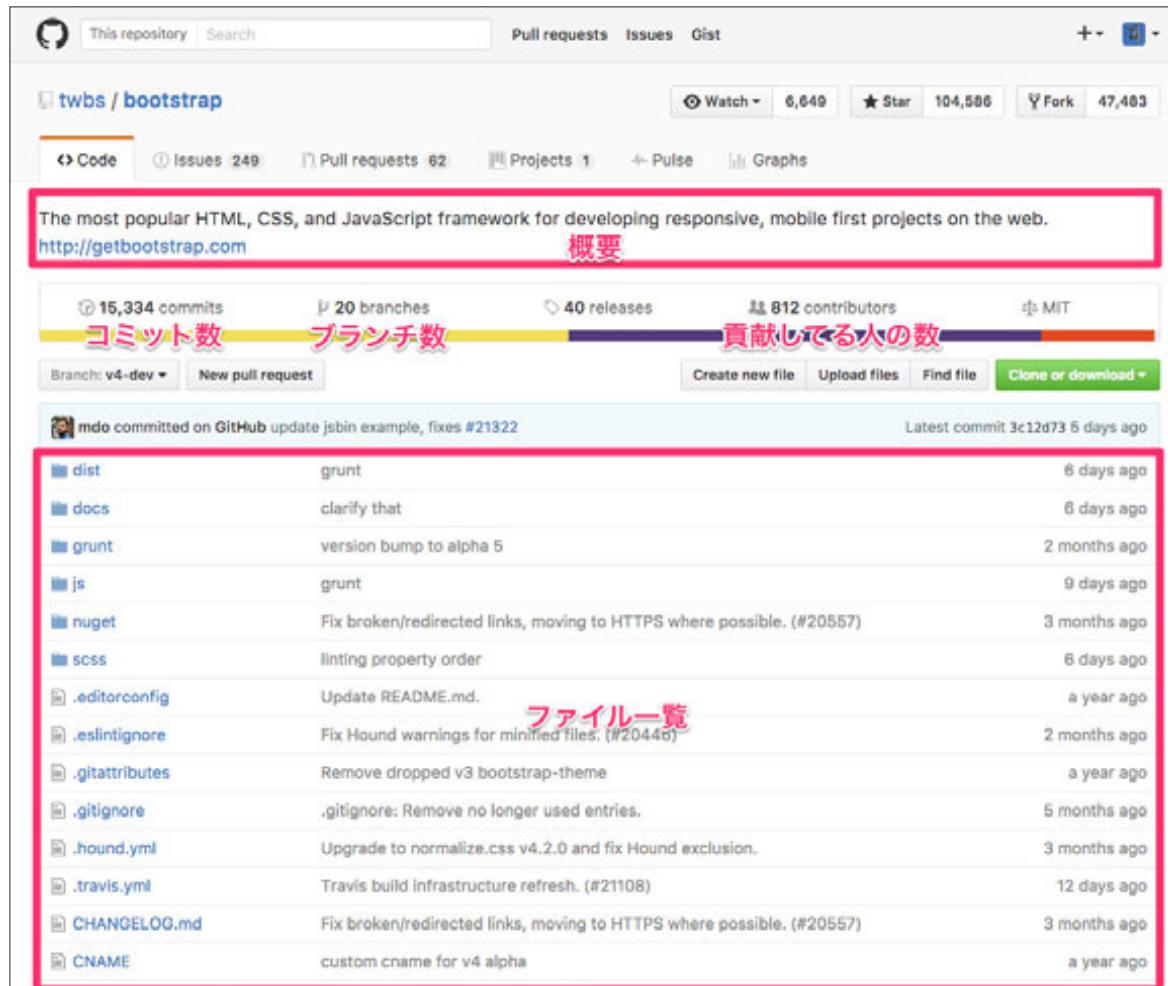


図 11 Bootstrap リポジトリの Code タブ

[5] Issues

このタブを選択すると、リポジトリに関するイシューを表示できます。

Issue（イシュー）は開発中のソフトウェアのバグトラッキングや課題管理などを行うための機能です。バグトラッキングや課題管理などを行なうためのツールには Redmine や JIRA などがありますが、これらのツールに含まれる機能を組み込んだようなものが GitHub の Issues です。

イシュー一覧の中の特定のイシューをクリックすると、個別のイシューページへ遷移できます。

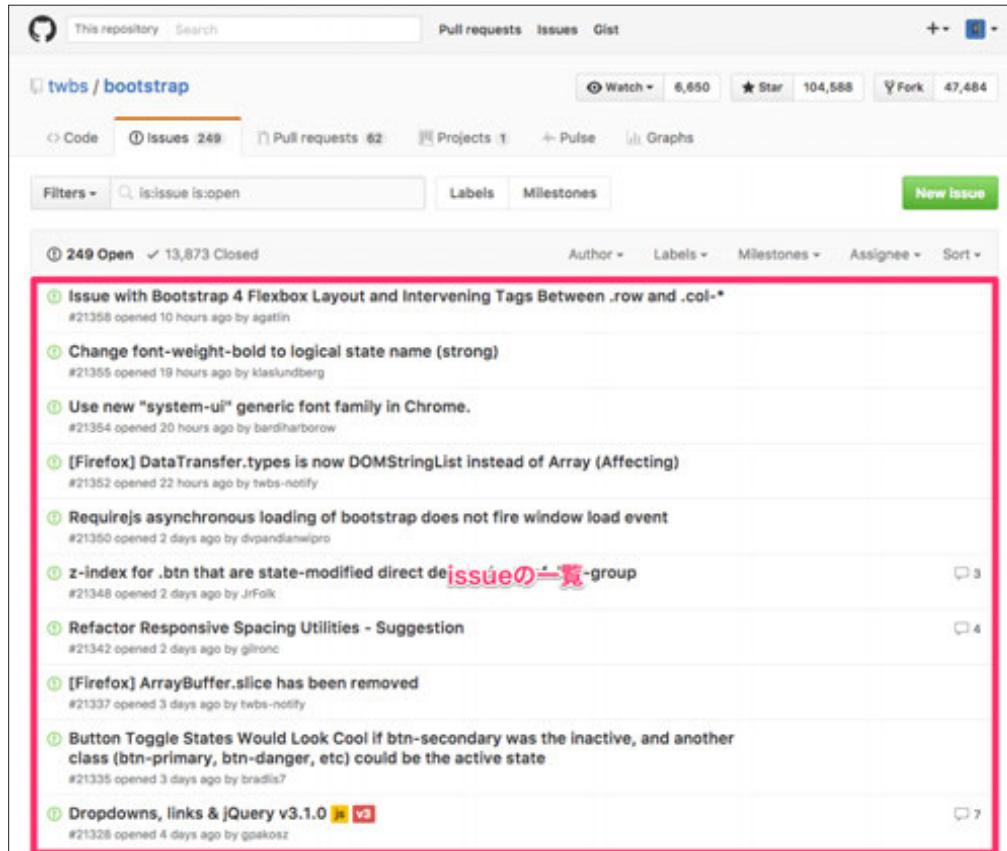


図 12 Bootstrap のリポジトリの Issues タブ

個別のイシューページにはコメントを書き込む機能があり、特定のトピックに関するコミュニケーションを行えます。

The screenshot shows a GitHub issue page for issue #306 titled 'Sticky footer'. The page includes a sidebar for 'Labelsなどの設定' (Labels and Milestones) which is highlighted with a pink box. The sidebar shows the 'docs' label is selected. The main area displays comments from users hugovincent, pkneale-ce, and dvanderbeek, each with their profile picture and timestamp. The comments discuss the implementation of sticky footers.

図 13 Bootstrap リポジトリ上の Issue の例

【6】Pull requests

このタブを選択すると、リポジトリに関するプルリクエストを表示できます。

プルリクエストはコードの変更点に関するレビューや議論を行うための機能です。

イシューの場合と同様に、特定のプルリクエストをクリックすると個別のプルリクエストページへ遷移できます。

The screenshot shows the GitHub interface for the twbs/bootstrap repository. The 'Pull requests' tab is active, displaying 62 open pull requests. A specific pull request, #21341, titled 'dropdown.js: Avoid calling jQuery("#")', is highlighted with a red border. The page includes various filtering and sorting options, such as 'Filters' (set to 'is:pr is:open'), 'Author', 'Labels', 'Milestones', 'Assignee', and 'Sort'. The pull request list shows details like the title, author, and creation date for each item.

図 14 Bootstrap リポジトリの Pull requests タブ

イシューの場合と同様に、個別のプルリクエストのページにもコメントを書き込む機能があります。

他の開発者とコミュニケーションを行い、必要であれば追加の修正などを行います。コードの変更を開発の本流に取り込める状態になったらマージが行われ、プルリクエストのページ上でコミュニケーションが終了します。

Update jQuery CDN to use jQuery.com instead of Google
#21306

Merged mdo merged 1 commit into twbs:v4-dev from coliff:patch-11 9 days ago

Conversation 2 Commits 1 Files changed 1

coliff commented 11 days ago

Pull request作成者のコメント

Fixes: #21130

1 like

Update jQuery CDN to use jQuery.com instead of Google ... ✓ ce65b57

mdo added css v4 docs ja and removed css labels 9 days ago

mdo added this to the v4.0.0-alpha.6 milestone 9 days ago

mdo merged commit d9eb97b into twbs:v4-dev 9 days ago View details 2 checks passed

bardiharborow commented 9 days ago

他の参加者のコメント

This uses a SHA256 hash instead of SHA384 (which seems to be the established pattern here). It should be:

sha384-3ceskX3iaEnIog=0chP8opvBy3M17Ce34nMjpBIwVTHfGYWQSGjwHDVRnpK0HJg7

mdo commented 9 days ago Member +1

Labels

- docs
- ja
- v4

Projects

Milestone

v4.0.0-alpha.6

Notifications

Subscribe

You're not receiving notifications from this thread.

3 participants

図 15 Bootstrap リポジトリ上のプルリクエストの例

[7] Projects

Projects はプロジェクト管理サービス「Trello」「Waffle」のような「カンバン方式」のタスク管理機能です。

リポジトリ内に複数の Project を作成できます。Project 一覧の中の特定のプロジェクトをクリックすると個別のプロジェクトページへ遷移できます。

twbs / bootstrap

Watch 6,650 Star 104,588 Fork 47,484

Code Issues 249 Pull requests 62 Projects 1 Pulse Graphs

Find a project

1 project

Alpha 6 Updated on Nov 1 No description projectの一覧

図 16 Bootstrap リポジトリの Projects タブ

個別のプロジェクトページでは、ワークフローに合わせた「カラム」を作成し、カラム上に「カード」を配置できます。カードはイシューやプルリクエストからも作成できるので、プロジェクト全体の状態を可視化できます。

The screenshot shows the GitHub project page for the Bootstrap repository. At the top, there are tabs for Pull requests, Issues, Gist, Projects (which is selected), Pulse, and Graphs. Below the tabs, there's a summary for Alpha 6: 3 Tracking issues, 68 Fixed/Merged issues, 7 Needs review issues, and 8 Todo items. The main area is a Kanban board with four columns:

- Tracking**: Contains issues like "v4 Alpha 6 ship list" and "Further navbar refinement".
- Fixed/Merged**: Contains issues like "Spacing documentation for all sides needs updating" and "Move from \$.proxy to es6 arrow functions". A specific card in this column is highlighted with a red border and labeled "カード".
- Needs review**: Contains issues like "Made input[type="color"] squared instead of blocking across" and "Add @supports feature query for Carousel CSS 3D transforms".
- Todo**: Contains items like "Update grid to use same xs omit as the utilities.", "Moving to /docs sub-folder for launch", and "Follow up #20982 w/ active/op nav fixes".

Each card displays the issue number, title, author, and labels (e.g., docs, has-pr, v4). The "Fixed/Merged" column is currently highlighted with a pink border.

図 17 Bootstrap リポジトリの Project の例

【8】Pulse

このタブを選択すると、リポジトリ上での直近のアクティビティを表示できます。

特定の期間内に行われた変更や解決された課題の一覧などを確認できます。

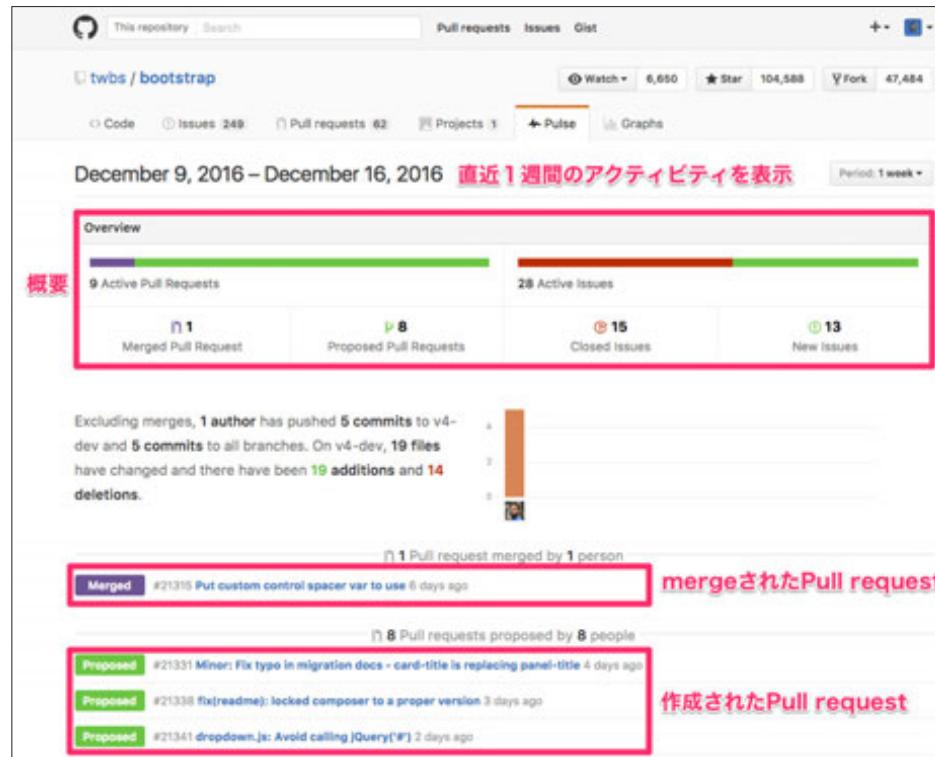
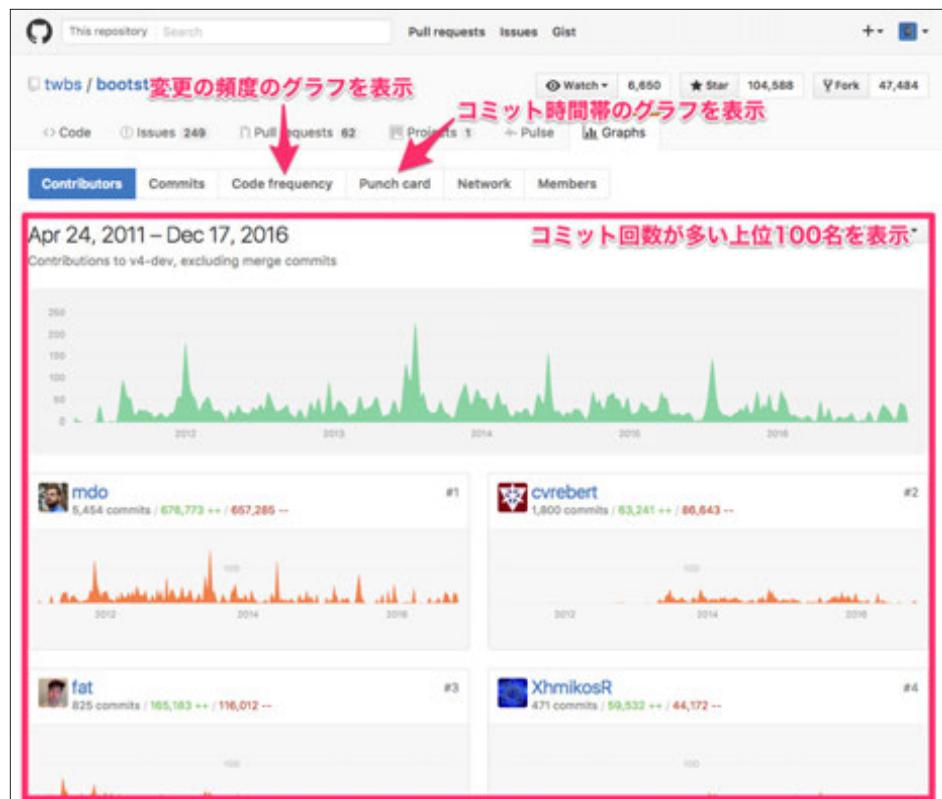


図 18 Bootstrap リポジトリの Pulse タブ

[9] Graphs

このタブを選択すると、リポジトリに関するグラフを表示できます。

コミット数、変更の頻度、コミットの多い時間帯などを確認できます。



リポジトリページから使用できる各機能の解説は以上です。

次回は GitHub 上の「リモートリポジトリ」での作業

本稿では GitHub の各機能の概要を解説しました。今のところ「こういう機能があるんだなあ」ぐらいに思つていただければ問題ありません。次回以降の記事で具体的な操作を交えながら解説していきます。

次回は GitHub 上の「リモートリポジトリ」での作業を解説する予定です。お楽しみに!

参考

- 『Pro Git』(written by Scott Chacon and Ben Straub and published by Apress)
- GitHub Help

09. これでもう怖くない、Git / GitHub におけるリモートリポジトリの作成、確認、変更、更新時の基本 5 コマンド

(2017年01月26日)

本連載では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説しています。今回は、リモートリポジトリに関する基本コマンドとして、git remote -v、git remote add、git push、git clone、git pullの使い方を紹介。

GitHub の「リモートリポジトリ」を触ってみよう

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終える頃には、GitやGitHubの基本的な操作が身に付いた状態になっていると思います。

前回の「2017年、GitHubを始めるために最低限知っておきたい各機能」ではGitHubの各機能の概要を解説しました。連載第9回目の本稿では、GitHub上に「リモートリポジトリ」を作成し、「リモートリポジトリ」に対する基本操作を試していきます。

本稿で解説する作業を行うには「GitHubのアカウント作成」や「SSH設定」を事前にやっておく必要があります。これらの準備手順については連載第7回の「はじまりはいつもブリック? GitリポジトリホスティングサービスGitHubとは」で解説しています。

リモートリポジトリを新規作成する

最初に、リモートリポジトリを新規作成する手順を解説していきます。

ローカルにあるファイルをリモートリポジトリに配置することによって、プロジェクトメンバー同士の共同作業を開始したり、異なる環境から最新のファイルを取得するための環境を整えたりすることができます。

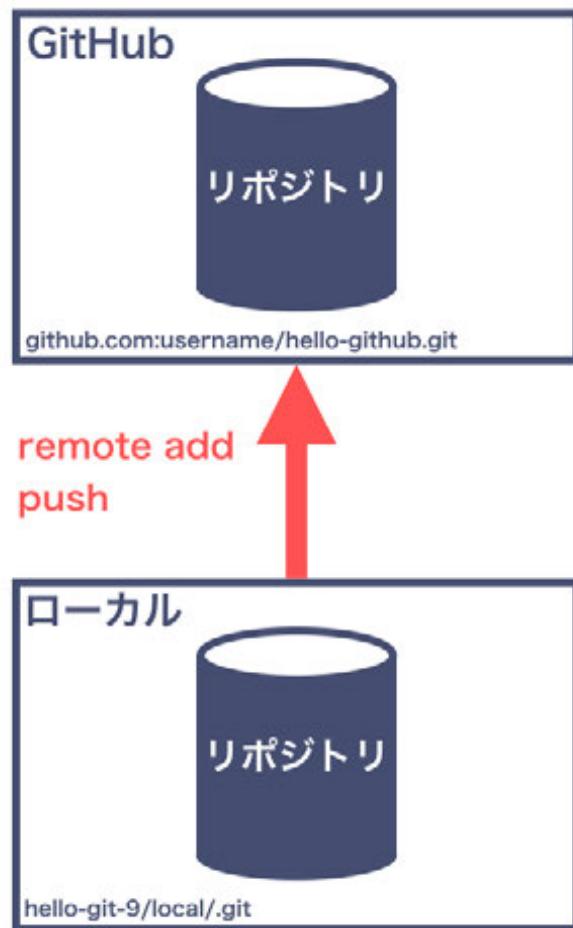


図 1 リモートリポジトリのセットアップ

ローカルリポジトリを作成する

ローカルリポジトリを作成します。このリポジトリで管理するファイルは、後ほど作成する GitHub 上の「リモートリポジトリ」に反映します。

適当なディレクトリに移動します。そして今回の作業用のディレクトリを作成し、そこへ移動します。ここでは「hello-git-9」というディレクトリを作成しました。

```
$ cd /Users/hirayashingo/Documents/
$ mkdir hello-git-9
$ cd hello-git-9
$ pwd
/Users/hirayashingo/Documents/hello-git-9
```

さらに、その中にリポジトリ用のディレクトリを作成し、そこへ移動します。ここでは「local」というディレクトリを作成しました。

```
$ mkdir local
$ cd local/
$ pwd
/Users/hirayashingo/Documents/hello-git-9/local
```

Git リポジトリを作成し、「README.md」ファイル（後述）を作成します。

```
$ git init
Initialized empty Git repository in /Users/hirayashingo/Documents/hello-git-9/
local/.git/
$ echo "# hello-github" > README.md
```

「README.md」ファイルをコミットします。

```
$ git add README.md
$ git commit -m "first commit"
[master (root-commit) 5fd9303] first commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
```

GitHub の GUI でリモートリポジトリを作成する

GitHub にログインし、「Create a new ...」をクリックし、表示されるメニュー上の「New repository」をクリックします。

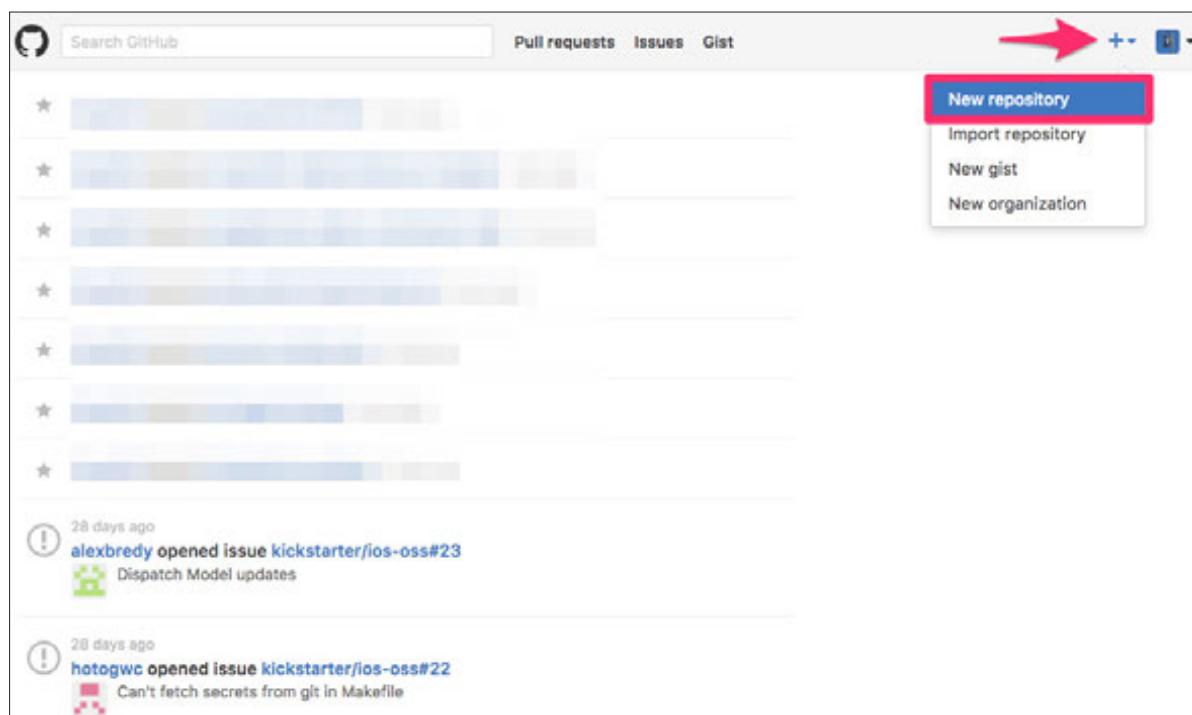


図 2 メニュー上の「New repository」項目

リポジトリ作成ページが表示されます。「Repository name」欄にリポジトリの名前を入力します。ここでは「hello-github」という名前を入力しました。

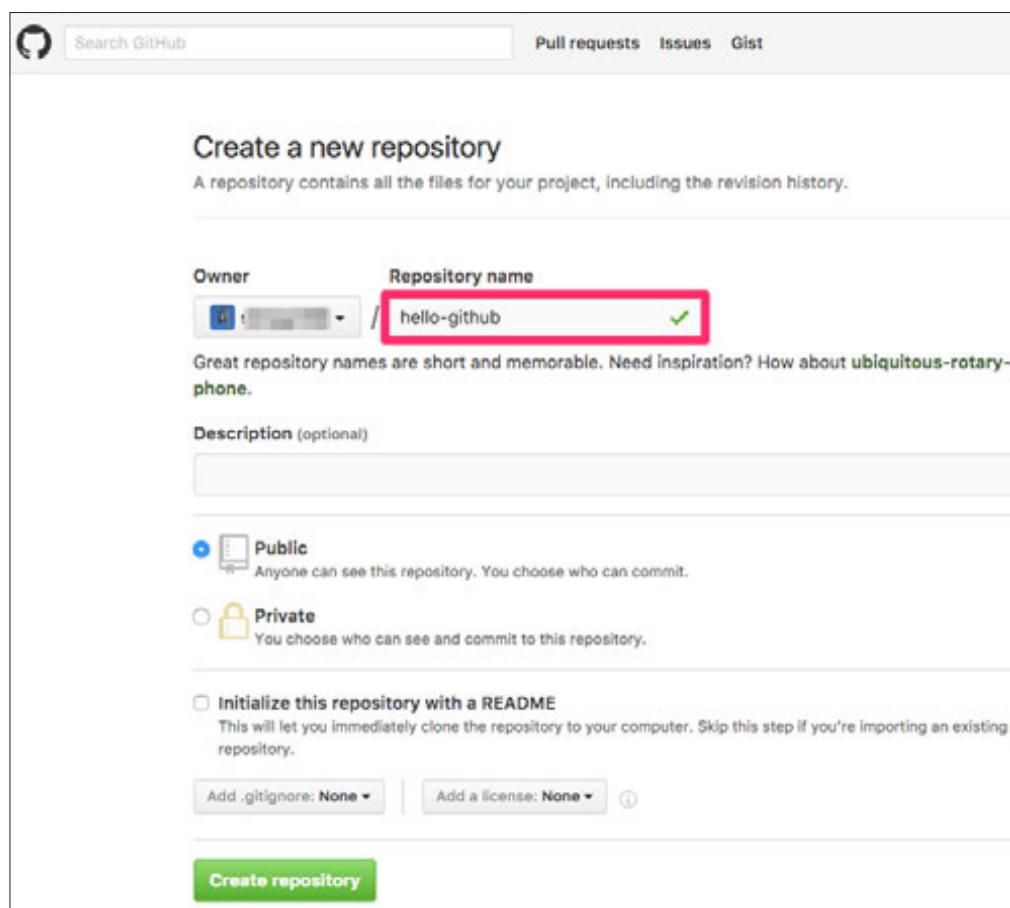


図 3 リポジトリ作成ページ

ページ中央にはリポジトリのタイプを選択するコントロールがあります。

「Public」を選択すると公開リポジトリが作成されます。この場合、リポジトリの中身は誰でも見られるようになります。

「Private」を選択すると非公開リポジトリを作成できます。非公開リポジトリを作成するには有料のプランに登録する必要があります。

今回は公開リポジトリを作成します。「Repository name」以外の入力項目・選択項目は編集せず、「Create repository」をクリックします。

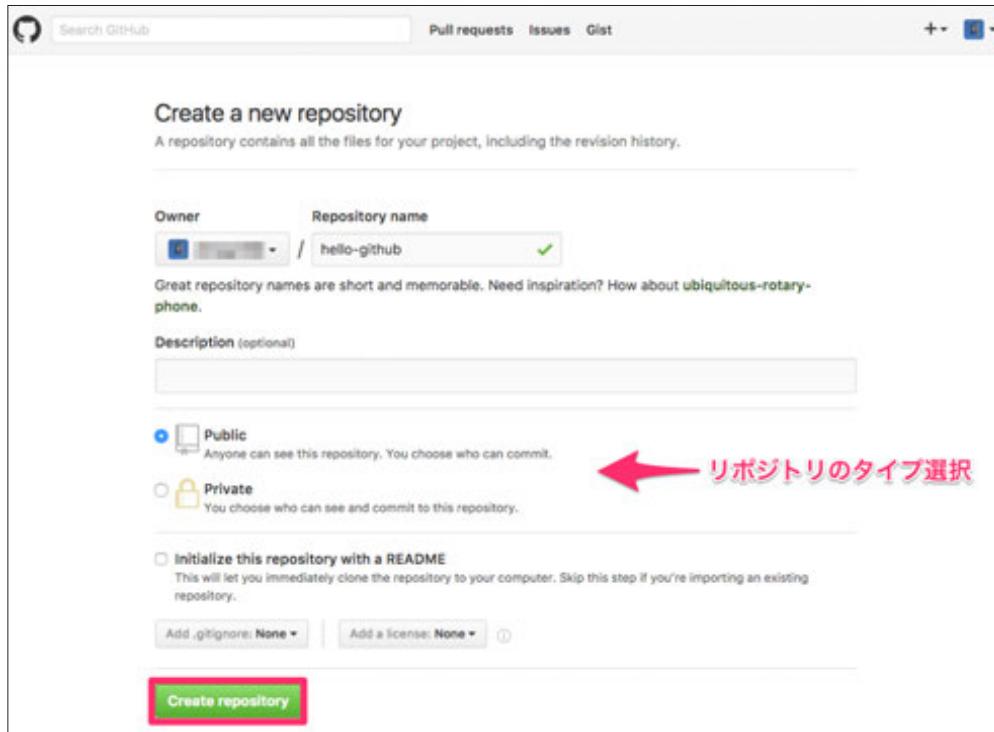


図 4 リポジトリのタイプ選択

リポジトリ作成が完了すると、リポジトリのページが表示されます。

現在、リポジトリの中には何も入っていないため、「リポジトリ作成後によく行う操作の例」が表示されています。

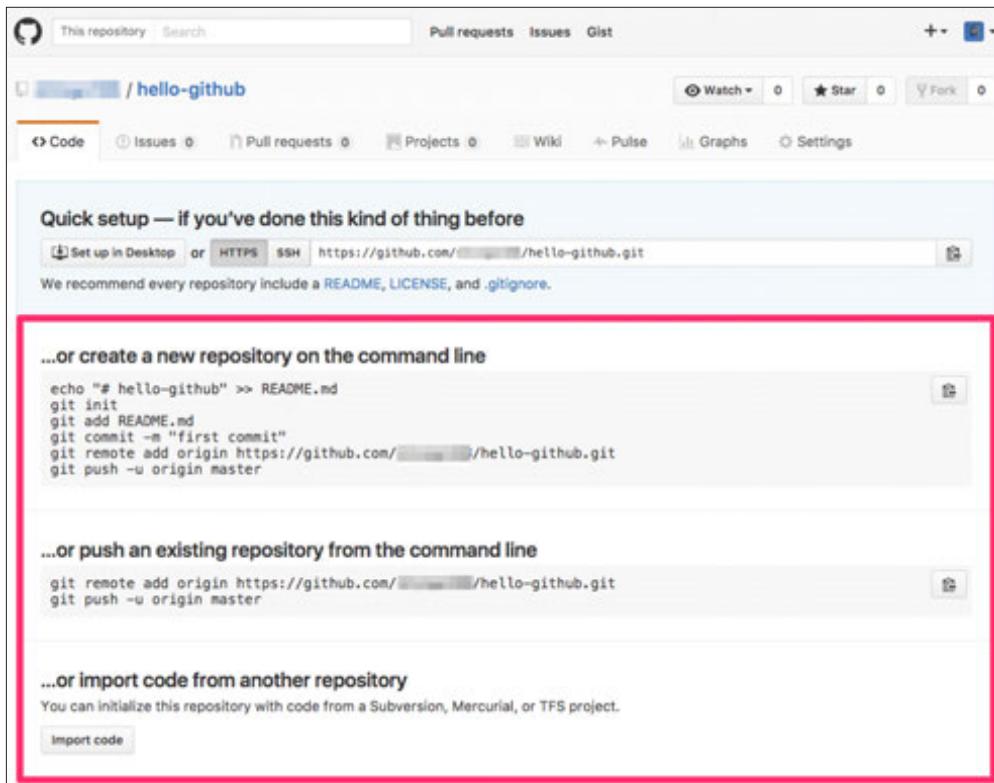


図 5 リポジトリ作成直後のリポジトリページ (HTTPS 接続のための説明)

SSH を使って GitHub 上にリポジトリに接続するので「SSH」をクリックします。

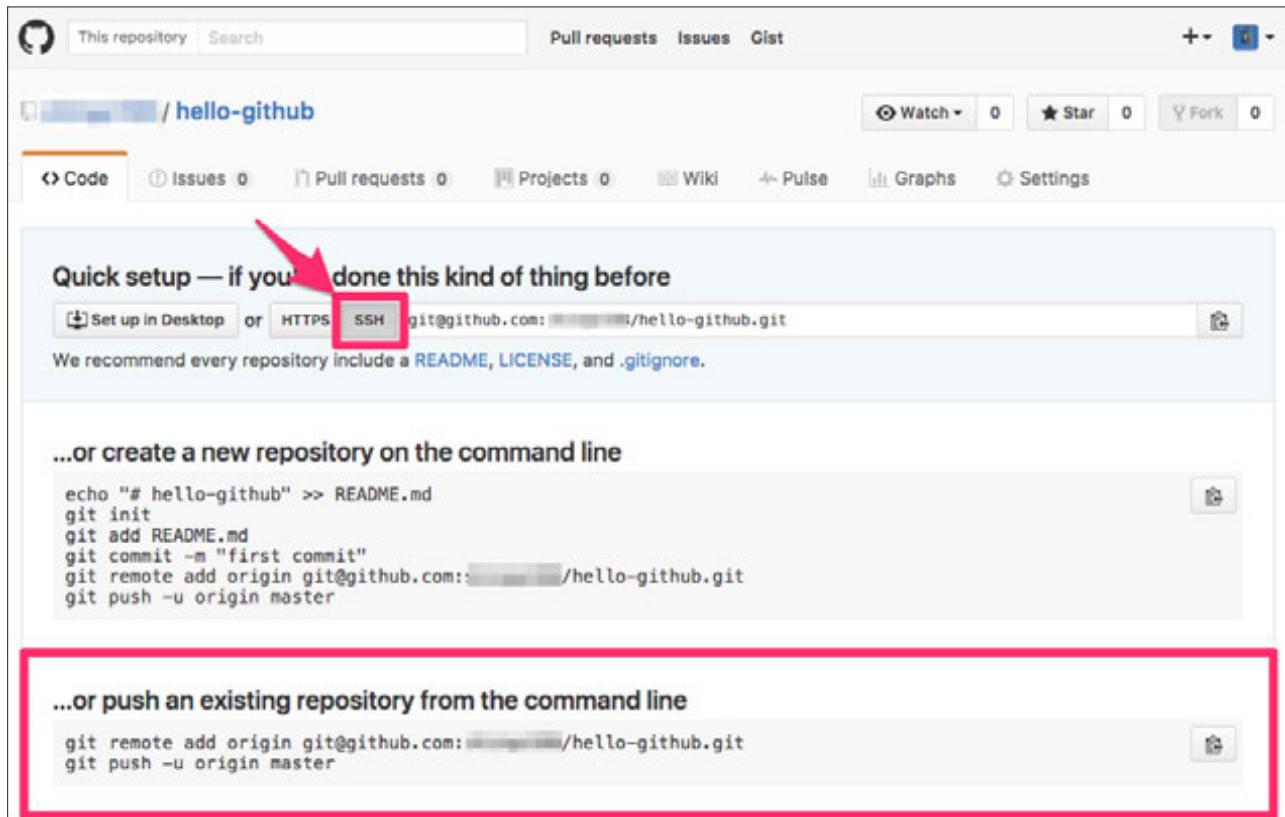


図 6 リポジトリ作成直後のリポジトリページ（SSH 接続のための説明）

今回は、既にローカルにリポジトリが存在します。次の「リモートリポジトリを登録する」では、見出し「…or push an existing repository from the command line」の下に書かれているコマンドを使用していきます。

リモートリポジトリを登録する

「ローカルリポジトリを作成する」で作成したリポジトリに GitHub 上のリモートリポジトリを登録する作業を行います。

- リモートリポジトリが登録済みかを確認——git remote -v コマンド

まずは「git remote -v」コマンドを使用して、登録済みのリモートリポジトリを確認してみます。当然ですが、現時点では何も表示されません。

```
$ git remote -v
```

- リモートリポジトリを登録——git remote add コマンド

「git remote add <リモート名> <リポジトリ URL>」コマンドを使用してリモートリポジトリを登録しましょう。「<リポジトリ URL> に存在するリモートリポジトリ」が「<リモート名> という名前」で登録されます。

GitHub 上のリポジトリの URL は「git@github.com:< ユーザー名 >/< リポジトリ名 >.git」という形式になります。< ユーザー名 > はユーザー名に置き換えてください。

```
$ git remote add origin git@github.com:username/hello-github.git
```

- 再度、登録済みか確認

再度「git remote -v」コマンドを使用すれば、リモートリポジトリが登録されたことを確認できます。

```
$ git remote -v
origin  git@github.com:username/hello-github.git (fetch)
origin  git@github.com:username/hello-github.git (push)
```

リモートリポジトリに反映する——git push コマンド

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します。

「git push <リモート名> <ブランチ名>」コマンドを使用します。<リモート名>にはリモートリポジトリ登録時に付けた名前を指定します。<ブランチ名>には反映先のブランチ名を指定します。

以下のコマンドを実行すると、ローカルリポジトリの「master」ブランチの内容が、リモートリポジトリ「origin」の「master」ブランチに反映されます。

```
$ git push origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 230 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:username/hello-github.git
 * [new branch]      master -> master
```

リポジトリのページへ戻り更新してみましょう。ローカルリポジトリの内容が反映されているはずです。

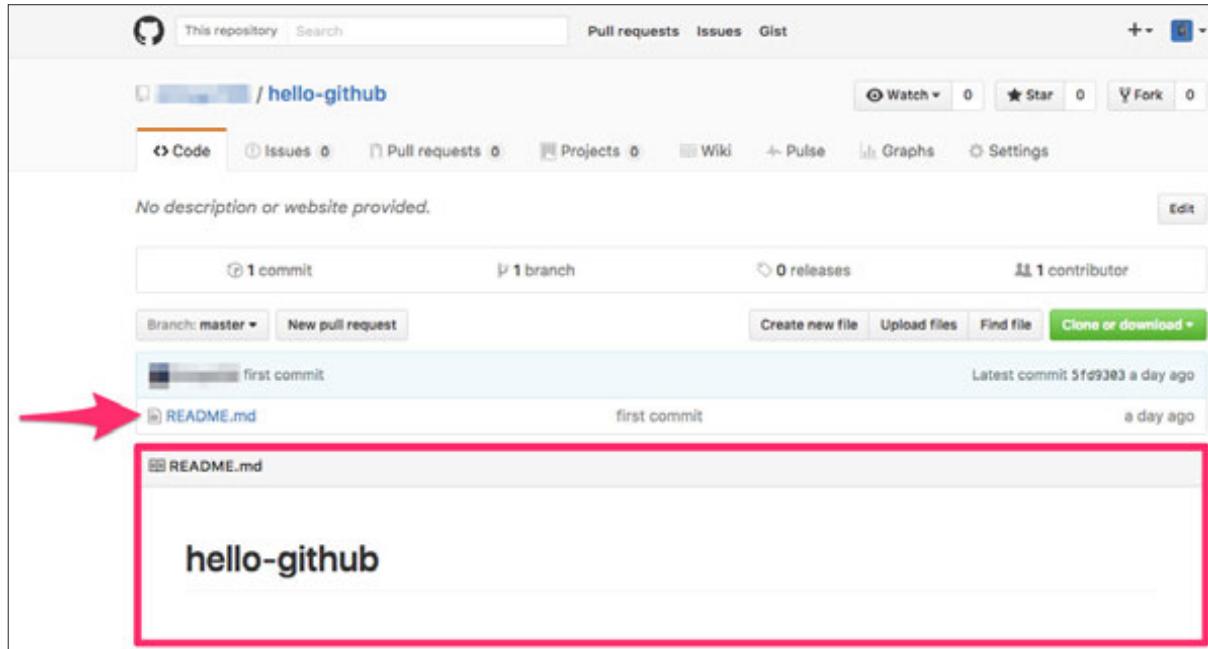


図 7 リポジトリページ

README.md ファイルとは

リポジトリのルートに「README.md」という名前のファイルを配置すると、図 7 のようにリポジトリページのファイル一覧の下にファイルの内容が表示されます。

「README.md」ファイルを配置することによって、リポジトリに関する説明を来訪者に伝えることができます。例えば、ソフトウェアライブラリのリポジトリであれば、ソースコードをビルドする前に必要な環境構築手順や、クラスの使用例などの説明が記述されます。

また、拡張子「.md」は「[Markdown 形式](#)」の文書用の拡張子です。ファイルの内容を Markdown 形式で記述することによって、文章を構造化することができます。今回、「README.md」ファイルの内容は、「# hello-github」にしました。「# < 文章 >」という記述は [HTML](#) の [<h1> タグ](#) 相当の見出しとして扱われます。

Markdown 形式の記述は、イシューやプルリクエストのコメントでも使えます。GitHub で利用可能な記法は以下のページで解説されています。

- <https://guides.github.com/features/mastering-markdown/>

既存のリモートリポジトリを取得する

ここまでで、ローカルリポジトリを元にリモートリポジトリを新規作成しました。次は、既存のリモートリポジトリをローカルに取り込む手順を解説します。

「リモートリポジトリ上の既存プロジェクトのソースコードを取得する」「別の PC で行う作業のためにリモートリポジトリからソースコードを取得する」などのケースをイメージしてみてください。

今回は、「リモートリポジトリを作成する」で作成したリモートリポジトリを「ローカルリポジトリを作成する」で扱ったディレクトリとは別のディレクトリに取り込みます。

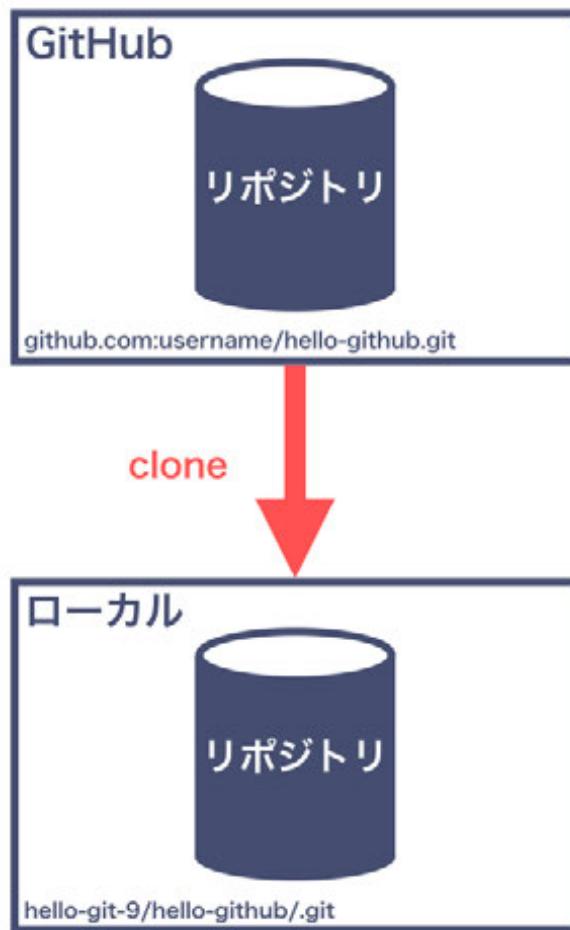


図 8 既存のリモートリポジトリを取得

リモートリポジトリの URL を確認する

リモートリポジトリの URL は GitHub のリポジトリページ上の「Clone or Download」をクリックすると表示できます。

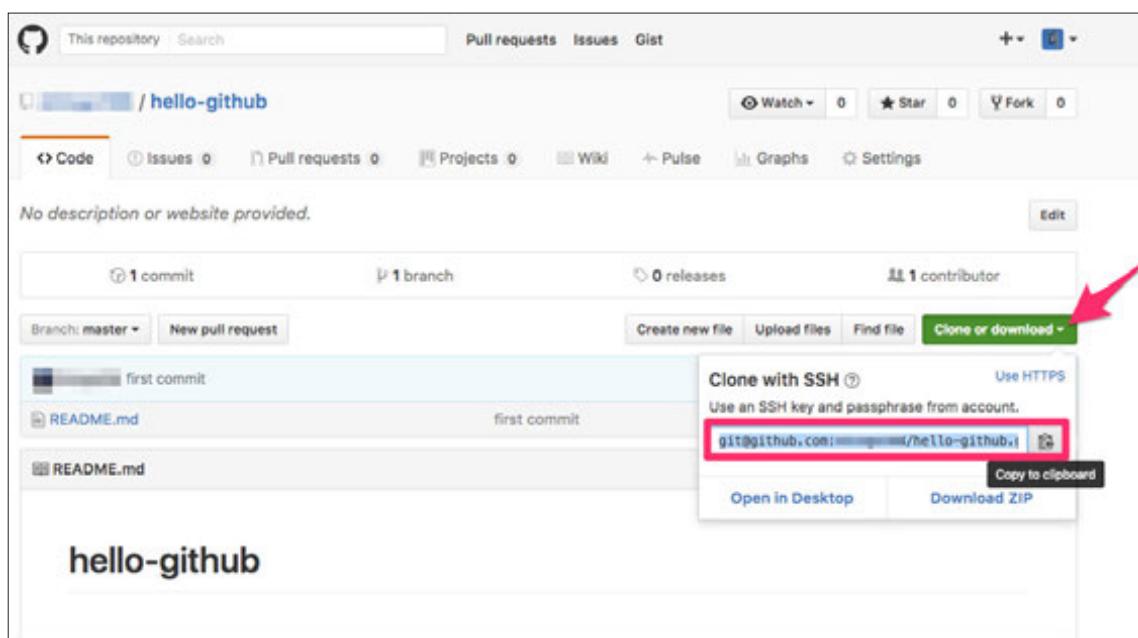


図 9 リポジトリページでリポジトリ URL を表示

リモートリポジトリを複製してローカルに取り込む——git clone コマンド

1つ上の階層に移動します。「hello-git-9」には「local」ディレクトリだけが存在する状態になっています。

```
$ pwd  
/Users/hirayashingo/Documents/hello-git-9/local  
$ cd ..  
$ pwd  
/Users/hirayashingo/Documents/hello-git-9  
$ ls  
local
```

「git clone <リポジトリ URL>」コマンドを使用してリモートリポジトリを複製してローカルに取り込みます。

```
$ git clone git@github.com:username/hello-github.git  
Cloning into 'hello-github'...  
remote: Counting objects: 3, done.  
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0  
Receiving objects: 100% (3/3), done.
```

「hello-git-9」の中に「hello-github」ディレクトリが作成されました。

```
$ ls  
hello-github local
```

「hello-github」ディレクトリの中は Git 管理下になっていて、「ローカルリポジトリを作成する」で作成したファイルがあります。

```
$ cd hello-github/  
$ pwd  
/Users/hirayashingo/Documents/hello-git-9/hello-github  
$ ls -a  
. .. .git README.md  
$ cat README.md  
# hello-github
```

既存のリモートリポジトリを複製してローカルに取り込む操作は、これで完了です。次の「リモートリポジトリを更新する」では、リモートリポジトリを更新していく中で行う操作を解説します。

リモートリポジトリを変更・更新する

最後に、リモートリポジトリを使用して共同作業を進めていく上で基本となる操作を解説します。

ローカルで行った変更をリモートリポジトリに反映すれば、共同作業者が変更を確認できるようになります。リモートリポジトリの変更をローカルに取り込めば、共同作業者が行った変更をローカルで確認できるようになります。

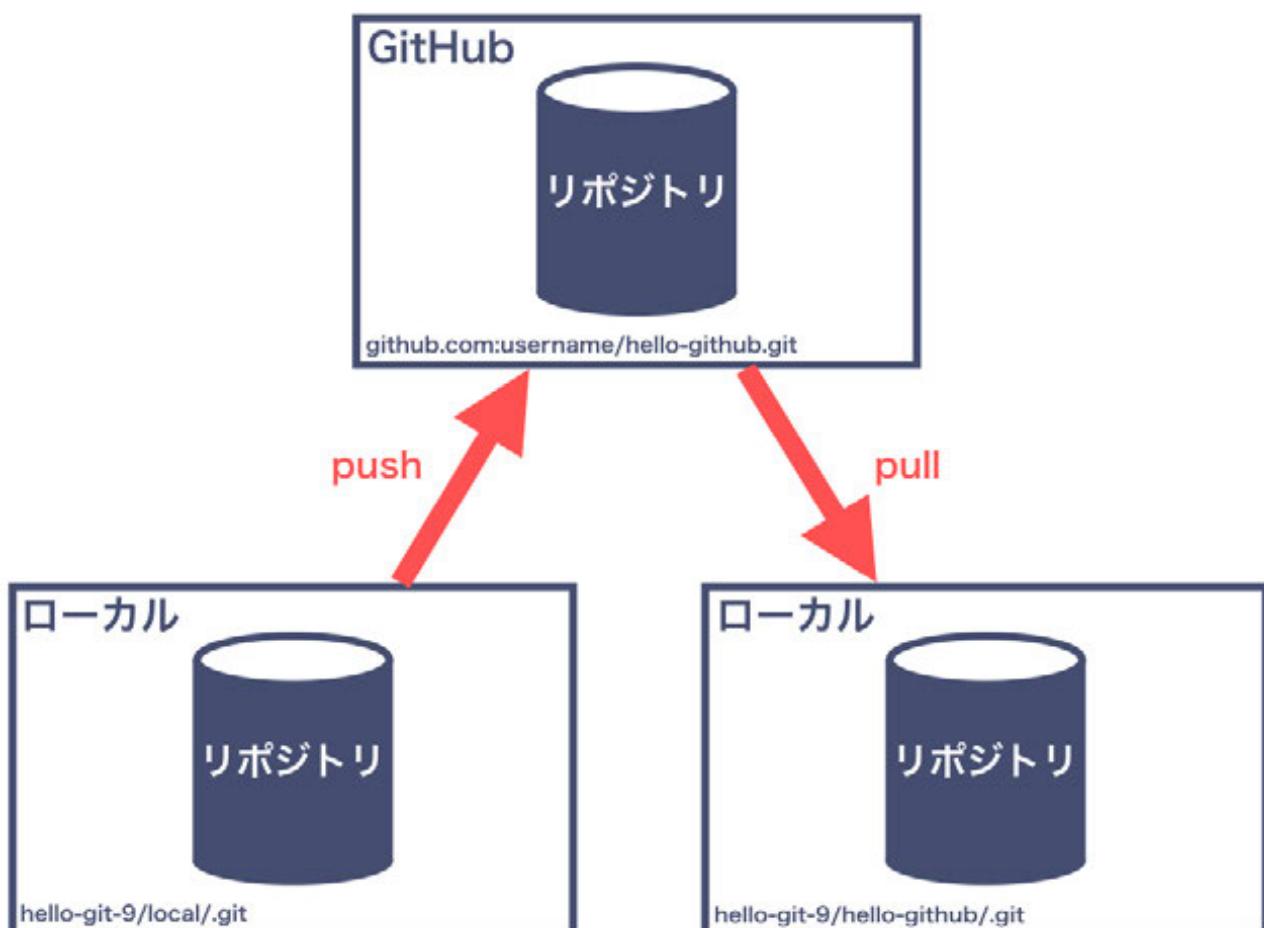


図 10 リモートリポジトリを更新

ファイルを変更・コミットし、リモートリポジトリに反映する

「hello-github」ディレクトリ内の「README.md」ファイルを変更し、git add コマンドでステージ、git commit -m コマンドでコミットします。

```
$ pwd
/Users/hirayashingo/Documents/hello-git-9/hello-github
$ echo "GitHub の機能を試すためのリポジトリです。" >> README.md
$ cat README.md
# hello-github
GitHub の機能を試すためのリポジトリです。
$ git add README.md
```

```
$ git commit -m "リポジトリの説明を追加"  
[master 8671af2] リポジトリの説明を追加  
1 file changed, 1 insertion(+)
```

先ほど紹介した git push コマンドでリモートリポジトリへ反映します。

```
$ git push origin master  
Counting objects: 3, done.  
Delta compression using up to 8 threads.  
Compressing objects: 100% (2/2), done.  
Writing objects: 100% (3/3), 355 bytes | 0 bytes/s, done.  
Total 3 (delta 0), reused 0 (delta 0)  
To github.com:username/hello-github.git  
 5fd9303..8671af2 master -> master
```

リポジトリのページへ戻り更新してみましょう。「README.md」ファイルに対する変更が反映されているはずです。

The screenshot shows the GitHub repository page for 'hello-github'. At the top, there's a navigation bar with 'This repository', 'Search', 'Pull requests', 'Issues', 'Gist', and a '+' button. Below the navigation is the repository name 'hello-github'. To the right are buttons for 'Watch' (0), 'Star' (0), 'Fork' (0), and 'Clone or download'. The main content area displays repository statistics: 2 commits, 1 branch, 0 releases, and 1 contributor. It also shows the latest commit information: 'Latest commit 8671af2 3 minutes ago'. Below this, a list of files includes 'README.md' with a note 'リポジトリの説明を追加' (Description added) and a timestamp '3 minutes ago'. A red arrow points to the commit message of the first commit, which reads 'GitHubの機能を試すためのリポジトリです。' (This repository is for testing GitHub features).

図 11 リポジトリページ

リモートリポジトリの更新をローカルに取り込む——git pull コマンド

「local」ディレクトリに移動します。

```
$ pwd  
/Users/hirayashingo/Documents/hello-git-9/hello-github  
$ cd ..  
$ cd local/  
$ pwd  
/Users/hirayashingo/Documents/hello-git-9/local
```

「README.md」ファイルの内容は古いままです。

```
$ cat README.md  
# hello-github
```

「git pull <リモート名> <ブランチ名>」コマンドを使用して

リモートリポジトリの最新データを取り込みます。

```
$ git pull origin master  
remote: Counting objects: 3, done.  
remote: Compressing objects: 100% (2/2), done.  
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0  
Unpacking objects: 100% (3/3), done.  
From github.com:username/hello-github  
 * branch          master      -> FETCH_HEAD  
   5fd9303..8671af2  master      -> origin/master  
Updating 5fd9303..8671af2  
Fast-forward  
 README.md | 1 +  
 1 file changed, 1 insertion(+)
```

「README.md」ファイルの内容が更新されました。

```
$ cat README.md  
# hello-github  
GitHub の機能を試すためのリポジトリです。
```

本稿で紹介した Git のコマンド一覧

本稿では GitHub 上に「リモートリポジトリ」を作成し、「リモートリポジトリ」に対する基本操作を試していきました。

本記事で初めて登場した Git のコマンドは以下の通りです。

- `git remote -v` : 登録済みのリモートリポジトリの「リモート名」「リポジトリ URL」を表示する
- `git remote add <リモート名> <リポジトリ URL>` : リモートリポジトリを登録する
- `git push <リモート名> <ブランチ名>` : 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する
- `git clone <リポジトリ URL>` : <リポジトリ URL> に存在するリモートリポジトリをローカルに複製する
- `git pull <リモート名> <ブランチ名>` : リモートリポジトリの内容をローカルリポジトリに反映する

次回の記事では「プルリクエスト」について解説する予定です。お楽しみに!

参考

- [『Pro Git』](#) (written by Scott Chacon and Ben Straub and published by Apress)
- [GitHub Help](#)

10.GitHub を使うなら最低限知っておきたい、 プルリクエストの送り方とレビュー、マージの基本

(2017年02月27日)

本連載では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説しています。今回は、ブランチ作成、README.mdファイルへの変更、GitHub上のリポジトリへの反映を行って、プルリクエストを作成し、レビューやマージを行う手順を紹介します。

GitHub の「リモートリポジトリ」を触ってみよう

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終える頃には、GitやGitHubの基本的な操作が身に付いた状態になっていると思います。

前回の記事「これでもう怖くない、Git / GitHubにおけるリモートリポジトリの作成、確認、変更、更新時の基本5コマンド」では「リモートリポジトリ」に対する基本操作を解説しました。

連載第10回目の本稿では「プルリクエスト」の基本機能や手順について解説します。

本稿で解説する作業は、前回の記事で作成したリポジトリをベースに進めています。リポジトリを作成する作業をまだ行っていない場合は、前回の記事を読みながら準備を進めてみてください。

プルリクエスト作成の準備

ブランチ作成、README.mdファイルへの変更、GitHub上のリポジトリへの反映を行って、プルリクエストを作成する準備を行います。

ブランチ作成

前回の記事で作成したリポジトリへ移動します。

```
$ cd /Users/hirayashingo/Documents/hello-git-9/hello-github
$ pwd
/Users/hirayashingo/Documents/hello-git-9/hello-github
```

プルリクエスト用のブランチ「edit-readme」を作成し、このブランチに切り替えます。

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
$ git checkout -b edit-readme
```

カレントブランチが「edit-readme」ブランチになりました。

```
$ git status
On branch edit-readme
nothing to commit, working tree clean
```

README.md ファイルへの変更

「README.md」ファイルに1行追加し、git add コマンドでステージ、「git commit -m」コマンドでコミットします。

```
$ echo "Git 連載記事の作業用のリポジトリです。" >> README.md
$ cat README.md
# hello-github
GitHub の機能を試すためのリポジトリです。
Git 連載記事の作業用のリポジトリです。
$ git add README.md
$ git commit -m "リポジトリの説明をさらに追加"
[edit-readme 149d3a6] リポジトリの説明をさらに追加
 1 file changed, 1 insertion(+)
```

GitHub 上のリポジトリへの反映

git push コマンドでリモートリポジトリへ反映します。以下のコマンドを実行すると、今回行った変更がリモートリポジトリ「origin」の「edit-readme」ブランチに反映されます。

```
$ git push origin edit-readme
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 402 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To github.com:username/hello-github.git
 * [new branch]      edit-readme -> edit-readme
```

これでプルリクエストを作成する準備が整いました。

プルリクエストを作成する

ここで GitHub 上のリポジトリページを表示してみましょう。以下のような表示になっているかと思います。

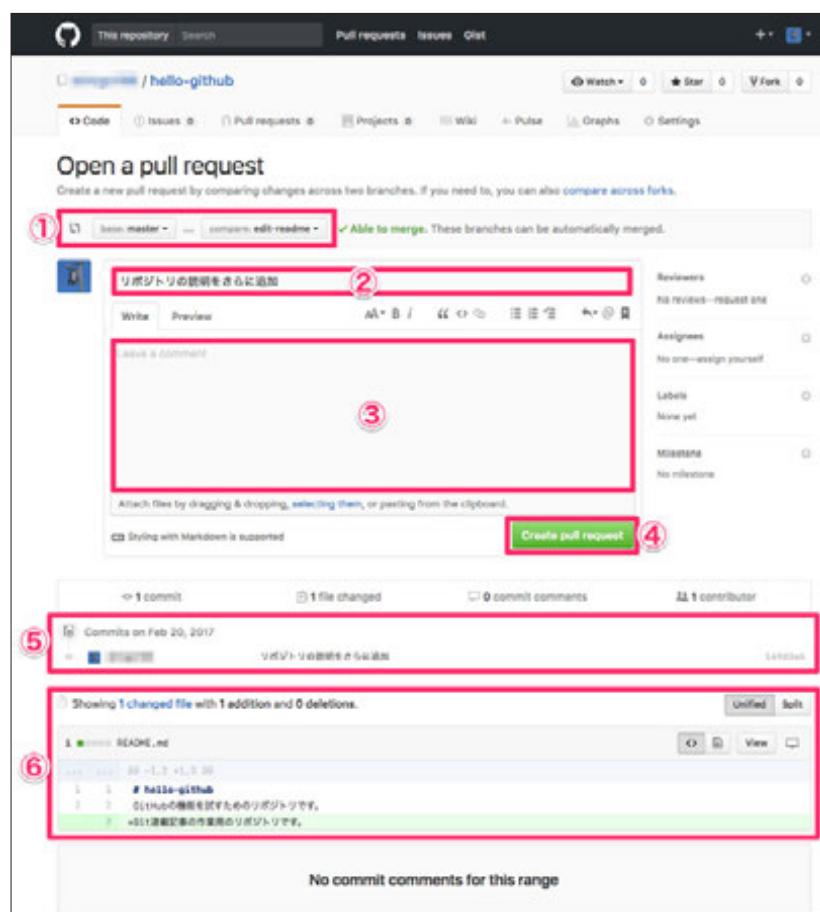
「Your recently pushed branches:」という表示は、メインのブランチ（今回の場合は master）以外のブランチが push された場合、しばらくの間表示されます。

「Compare & pull request」ボタンをクリックし、プルリクエスト作成ページへ移動します。

プルリクエスト作成ページの構成

プルリクエスト作成ページを表示できました。ページの構成は以下のようになっています。

1. ブランチ
2. タイトル
3. 概要
4. 「Create pull request」ボタン
5. コミットの履歴
6. ファイルの差分



【1】 ブランチ

「マージ先ブランチ」と「プルリクエスト対象ブランチ」がここに表示されます。

今回の場合、これから作成するプルリクエストが承認されると、「edit-readme」ブランチの内容が「master」ブランチへマージされることになります。

【2】 タイトル

プルリクエストのタイトルをここに入力します。

今回のようにプルリクエストの対象となるコミットが 1 つだけの場合、タイトルの初期値はコミットコメントになります。

【3】 概要

変更内容、関連するイシューや資料へのリンクなど、レビュー担当者（プルリクエストを確認する人）へ伝えておきたいことをここに書きます。

Markdown 形式で記述すれば、文章を構造化することができます。

【4】 「Create pull request」ボタン

このボタンを押すとプルリクエストが作成されます。

【5】 コミットの履歴

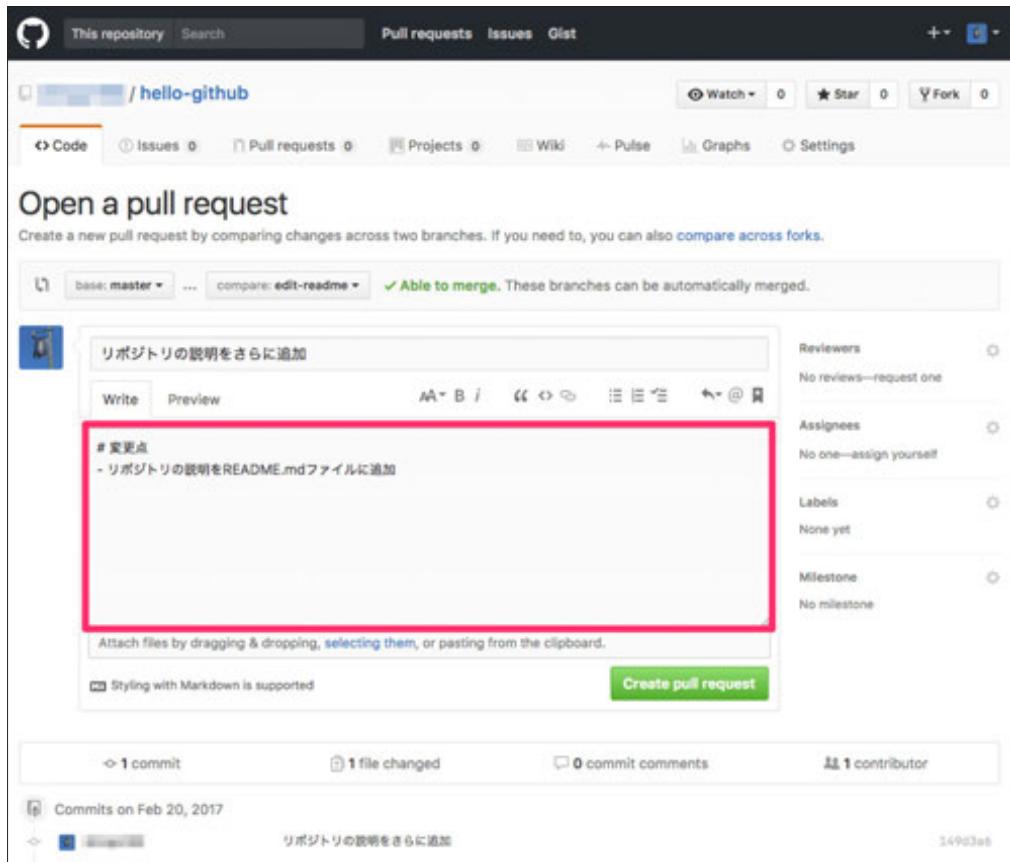
プルリクエストの対象となるコミットがここに表示されます。

【6】 ファイルの差分

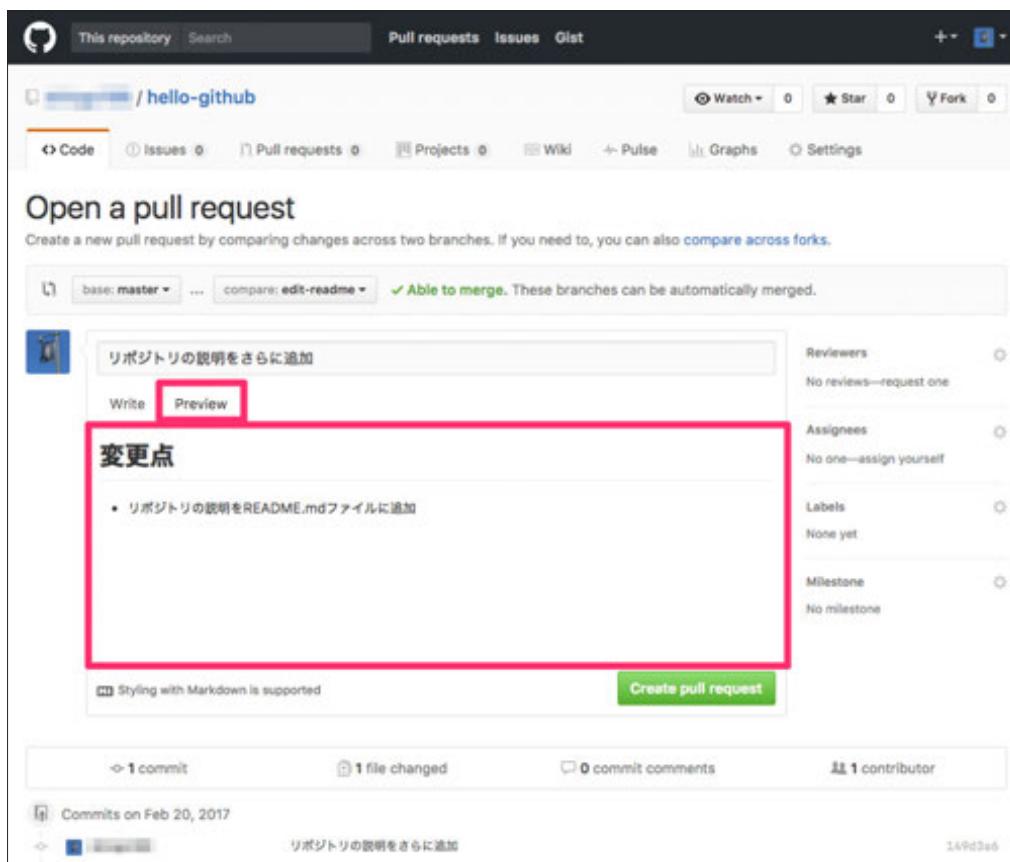
ファイルの変更箇所が、ここに表示されます。README.md ファイルに 1 行追加したことがよく分かるかと思います。

見出しひとリスト項目 1 件を Markdown 形式で記入してプルリクエストを作成

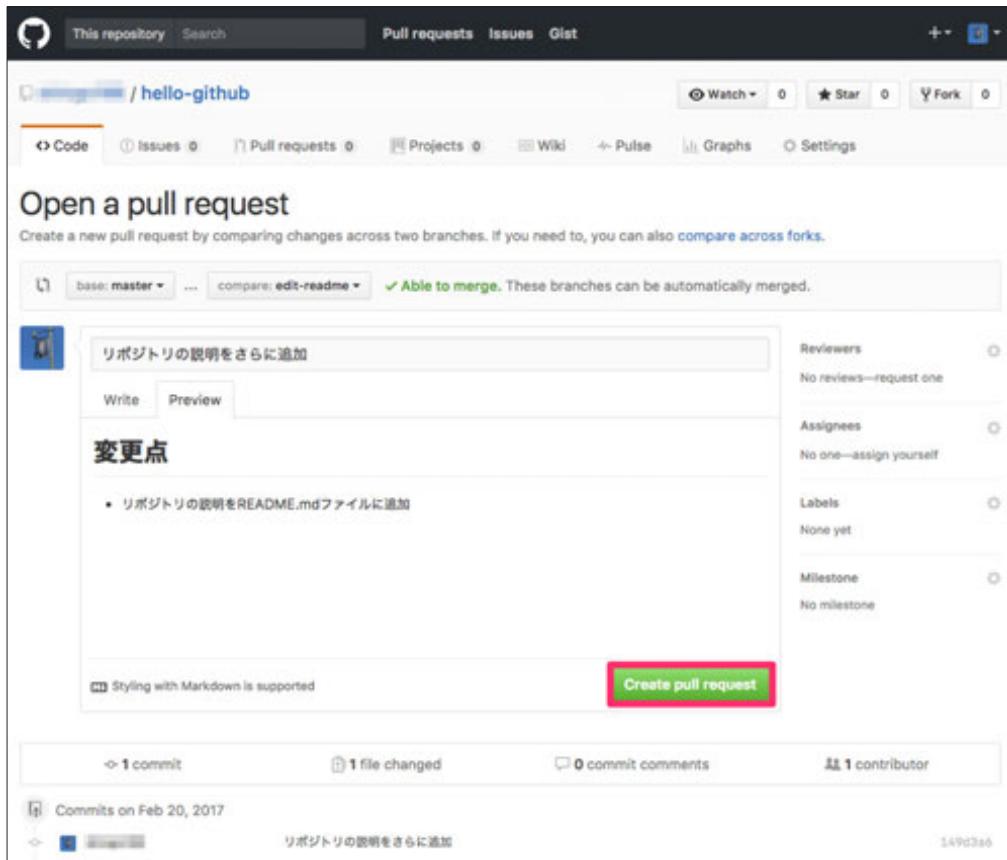
今回は以下のように、見出しひとリスト項目 1 件を Markdown 形式で記入しました。



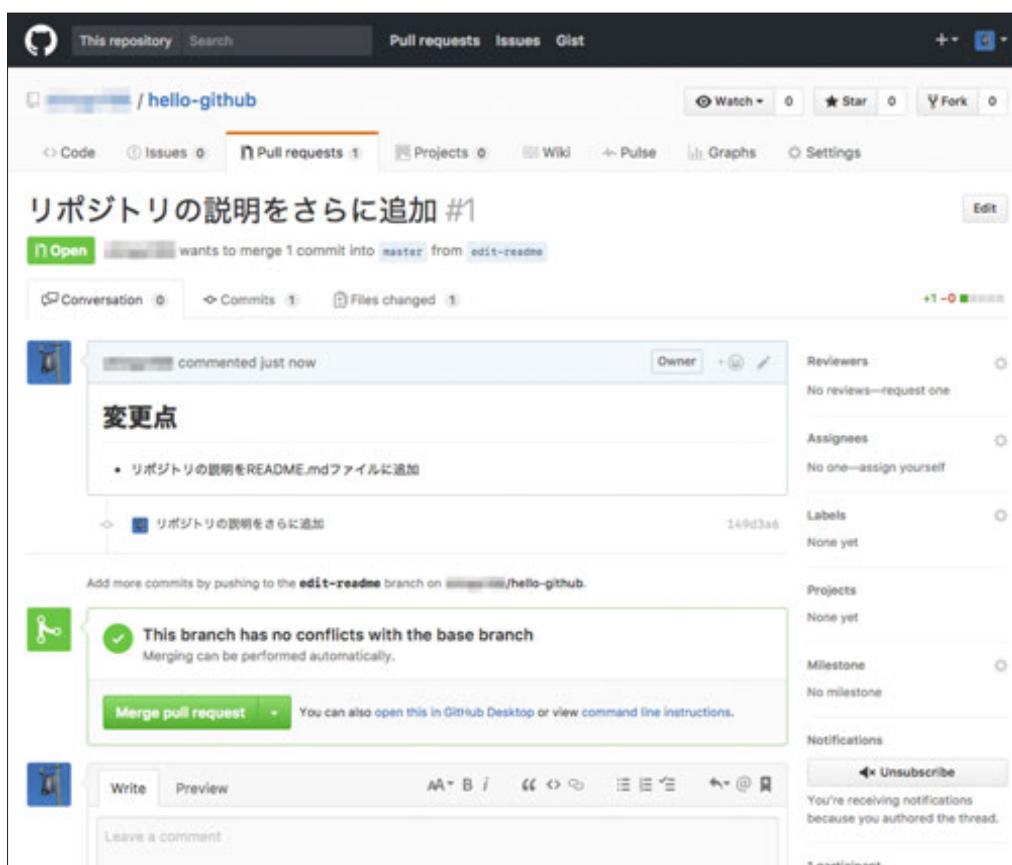
「Preview」をクリックすると、どのように描画されるかを確認できます。



「Create pull request」ボタンクリックしてプルリクエストを作成します。



プルリクエストの作成が完了しました。作成したプルリクエストのページが表示されます。



プルリクエストを作成する作業の解説は以上です。

プルリクエストをレビューする

プルリクエストをレビューする人の視点からプルリクエストのページを見ていきます。

「Conversation」タブでレビューを行う

まずは、デフォルトで表示される「Conversation」タブの内容を解説していきます。

ページの構成は以下のようになっています。

1. タイトル
2. ブランチ
3. タブ
4. 概要
5. 各種ログ
6. 「Merge pull request」ボタン
7. コメント欄

The screenshot shows a GitHub repository page for 'hello-github'. The 'Pull requests' tab is selected. A specific pull request is open, with its title 'リポジトリの説明をさらに追加 #1' highlighted by a red circle labeled ①. Below the title, there's a green 'Open' button and a status message 'wants to merge 1 commit into master from edit-readme' with a red circle labeled ②. Underneath the title, there are tabs for 'Conversation' (with 0 messages), 'Commits' (with 1 commit), and 'Files changed' (with 1 file). A red circle labeled ③ highlights the 'Conversation' tab. The main content area shows a comment from a user named 'commented 36 minutes ago' with a red circle labeled ④. The comment text is: 'リポジトリの説明をREADME.mdファイルに追加'. Below the comment, there's a red circle labeled ⑤ pointing to a link: 'リポジトリの説明をさらに追加'. To the right of the comment area, there are sections for 'Reviewers', 'Assignees', 'Labels', 'Projects', 'Milestone', and 'Notifications'. At the bottom of the comment section, there's a green 'Merge pull request' button with a red circle labeled ⑥. A note below it says 'You can also open this in GitHub Desktop or view command line instructions.' In the bottom right corner of the comment area, there's a 'Comment' button with a red circle labeled ⑦. The entire comment area is highlighted with a large red box.

【1】 タイトル

プルリクエストのタイトルが 1 番上に表示されます。変更の内容がすぐに理解できるタイトルだとうれしいですね。

【2】 ブランチ

ここを見ると、どのブランチからどのブランチへマージされるのかが分かります。

【3】 タブ

デフォルトで「Conversation」タブが選択されています。「Commits」と「Files changed」タブについては後ほど解説します。

【4】 概要

ここを見ると、より詳しいプルリクエストの情報を知ることができます。

【5】 各種ログ

ここを見ると、プルリクエストに関連する各種ログを確認できます。今回の場合は、コミット 1 件のログだけ表示されています。

【6】 「Merge pull request」ボタン

このボタンを押すとマージが実行されます。今回の場合は、「edit-readme」ブランチの内容が「master」ブランチへマージされます。

【7】 コメント欄

このテキストボックスにコメントを記入し、「Comment」ボタンをクリックするとコメントを追加できます。プルリクエスト作成時の「概要」と同様に、Markdown 形式で記述できます。

コメント機能は、プルリクエストに関わる人同士のさまざまなコミュニケーションのために使用できます。例えば以下のようなコメントを書き込むことができます。

- レビュー担当者が、実装者に対して、修正してほしいことを書き込む
- 実装者が、レビュー担当者に対して、修正したことを探せるコメントを書き込む
- レビュー担当者が、実装者に対して、マージしても問題ないことを書き込む

プルリクエストの対象となるコミットの一覧「Commits」タブ

このタブを選択すると、プルリクエストの対象となるコミットの一覧を表示できます。

The screenshot shows a GitHub pull request page for a repository named 'hello-github'. The 'Pull requests' tab is selected. A green button labeled 'Open' is visible. Below it, there are tabs for 'Conversation' (0), 'Commits' (1), and 'Files changed' (1). The 'Commits' tab is highlighted with a red box. A red box also surrounds the commit list area. The commit details show a single commit from 'リポジトリの説明をさらに追加' made on Feb 20, 2017, with the commit message 'リポジトリの説明をさらに追加' and the SHA '149d3a6'.

ファイルの変更箇所を確認できる「Files changed」タブ

このタブを選択すると、ファイルの変更箇所を確認できます。

The screenshot shows the same GitHub pull request page for 'hello-github'. The 'Files changed' tab is selected, highlighted with a red box. Below it, there are tabs for 'Conversation' (0), 'Commits' (1), and 'Files changed' (1). A red box surrounds the file changes section. The changes are listed for 'README.md' with the message: 'Changes from all commits - 1 file - +1 -0'. The diff shows the addition of '# hello-github' and 'GitHubの機能を試すためのリポジトリです。' followed by '+Git連携記事の作業用のリポジトリです。'.

プルリクエストをレビューする作業の解説は以上です。

プルリクエストをマージする

誰がプルリクエストをマージするかは、開発チームのルールによります。

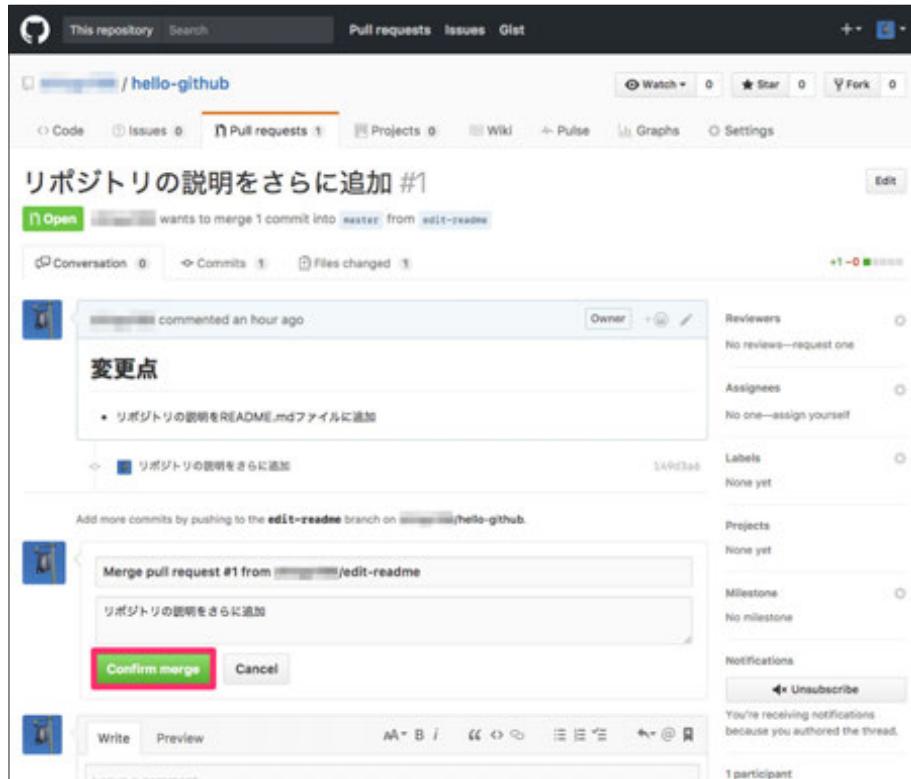
筆者が所属するチームの場合、プルリクエストは以下のフローで行うことが多いです。

1. 実装者（ファイルへの変更を行った人）がプルリクエストを作成する
2. レビュー担当者（実装者以外）が変更内容をレビューする
3. 必要に応じて実装者は修正を行う
4. マージできる状態になったら、実装者が「Merge pull request」ボタンを押してマージを行う

今回は1人チームなので、自分でマージしてしまいましょう。「Merge pull request」ボタンをクリックしてください。

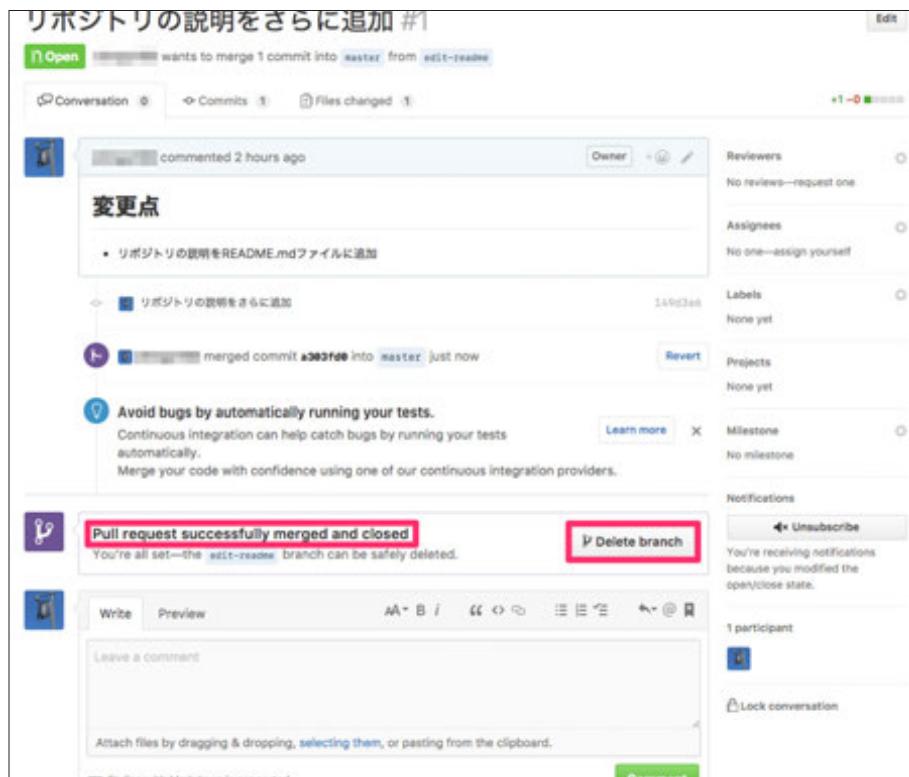
The screenshot shows a GitHub pull request merge dialog. At the top, it says 'リポジトリの説明をさらに追加 #1' (Repository description added #1) and 'Open'. Below that, it shows 'Conversation 0', 'Commits 1', and 'Files changed 1'. A message from a user says 'commented just now' with a link to 'edit-readme'. The main area is titled '変更点' (Changes) and lists a single commit: 'リポジトリの説明を README.md ファイルに追加'. To the right, there are sections for 'Reviewers' (No reviews—request one), 'Assignees' (No one—assign yourself), 'Labels' (None yet), 'Projects' (None yet), 'Milestone' (No milestone), and 'Notifications' (You're receiving notifications because you authored the thread). At the bottom, a green button says 'Merge pull request' with a dropdown menu, and a note says 'This branch has no conflicts with the base branch'. There's also a 'Leave a comment' section at the very bottom.

マージ確認状態になります。マージして問題なければ「Confirm merge」をクリックします。



マージが完了しました。

「Delete branch」をクリックすると、プルリクエストに使用した「edit-rename」ブランチを削除できます。このブランチはもう不要なので「Delete branch」をクリックしてブランチを削除します。



ブランチの削除が完了しました。

The screenshot shows a GitHub pull request merge commit page. At the top, there are tabs for Conversation (0), Commits (1), Files changed (1), and a summary of +1 -0 changes. Below the tabs, a message from a user is shown: "commented 2 hours ago". To the right of the message are buttons for Owner, + (add collaborator), and edit.

变更点 (Changes)

- リポジトリの説明を README.md ファイルに追加

下方显示了两个操作记录：

- リポジトリの説明をさらに追加 (149d386)
- merged commit a300fd0 into master 3 minutes ago (Revert)

右侧栏显示了以下信息：

- Reviewers: No reviews—request one
- Assignees: No one—assign yourself
- Labels: None yet
- Projects: None yet
- Milestone: No milestone
- Notifications: Unsubscribe (You're receiving notifications because you modified the open/close state.)
- 1 participant (User icon)
- Lock conversation

下方是评论输入框，显示 "Leave a comment" 和 "Comment" 按钮。底部有一个提示："ProTip! Add .patch or .diff to the end of URLs for Git's plaintext views."

プルリクエストをマージする作業の解説は以上です。

次回は「GitHub Issues (イシュー)」について

本稿では「プルリクエスト」の基本機能やフローを解説しました。今回扱うことができなかったプルリクエストの他の機能などは、また別の機会に紹介する予定です。

次回は「GitHub Issues (イシュー)」を解説する予定です。お楽しみに!

参考

『Pro Git』(written by Scott Chacon and Ben Straub and published by Apress)

11. 開発者のタスク管理がしやすくなる GitHub Issues の基本的な使い方

(2017年03月29日)

本連載では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説しています。今回は、GitHub Issuesの基本的な使い方について、イシューの作成、更新、削除の手順、一覧ページの見方などを紹介します。

GitHubにおける開発者のタスク管理も GitHub 上で

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終える頃には、GitやGitHubの基本的な操作が身に付いた状態になっていると思います。

前回の記事「GitHubを使うなら最低限知っておきたい、プルリクエストの送り方とレビュー、マージの基本」では「プルリクエスト」の基本機能や手順について解説しました。連載第11回目の今回は「イシュー(Issues)」(課題)に対する基本操作を解説します。

GitHub Issuesを使うことで、開発者のタスク管理やコミュニケーションがしやすくなります。

本稿で解説する作業は、前回までの記事で作成したGitHub上のリポジトリをベースに進めています。リポジトリを作成する作業をまだ行っていない場合は、前回までの記事を読みながら準備を進めてみてください。

イシューの作成

イシューを作成してみましょう。GitHub上のリポジトリページを表示し「Issues」タブをクリックします。

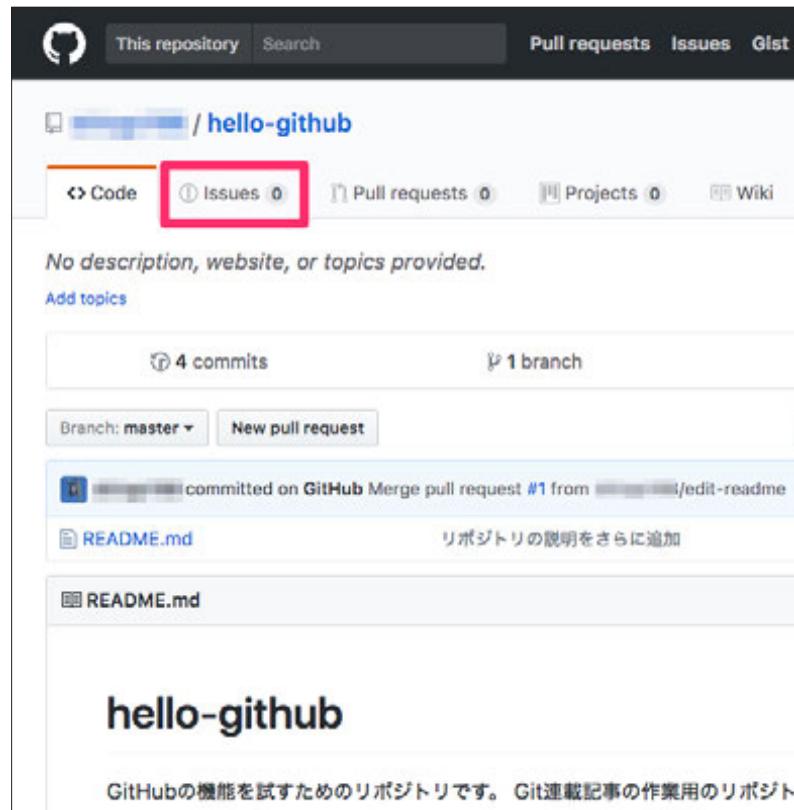


図 1 リポジトリページ

イシューの一覧ページが表示されました。「New issue」をクリックします。

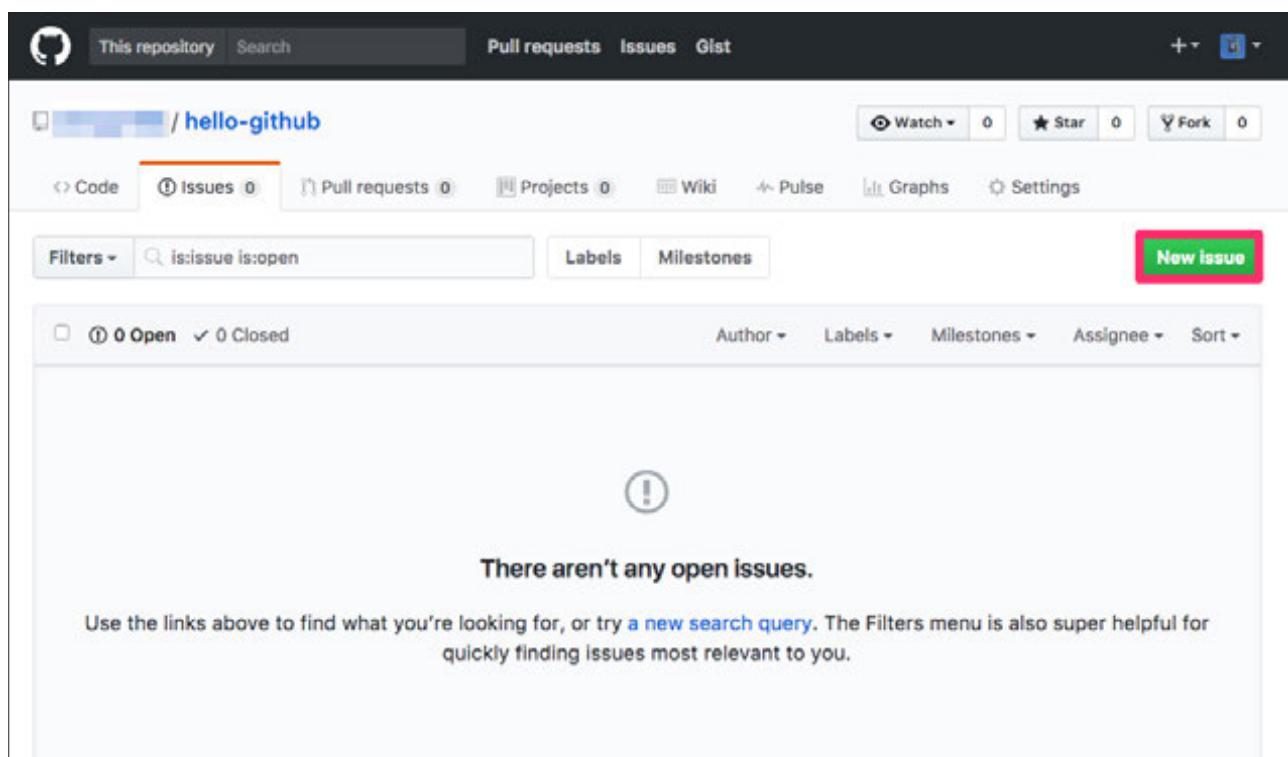


図 2 イシューの一覧ページ

イシュー作成ページを表示できました。

ページの構成は以下のようになっています。前回記事で扱った「プルリクエスト作成ページ」と同じような構成ですね。

1. タイトル
2. 説明
3. 担当者、ラベルなどの設定
4. 「Submit new issue」ボタン

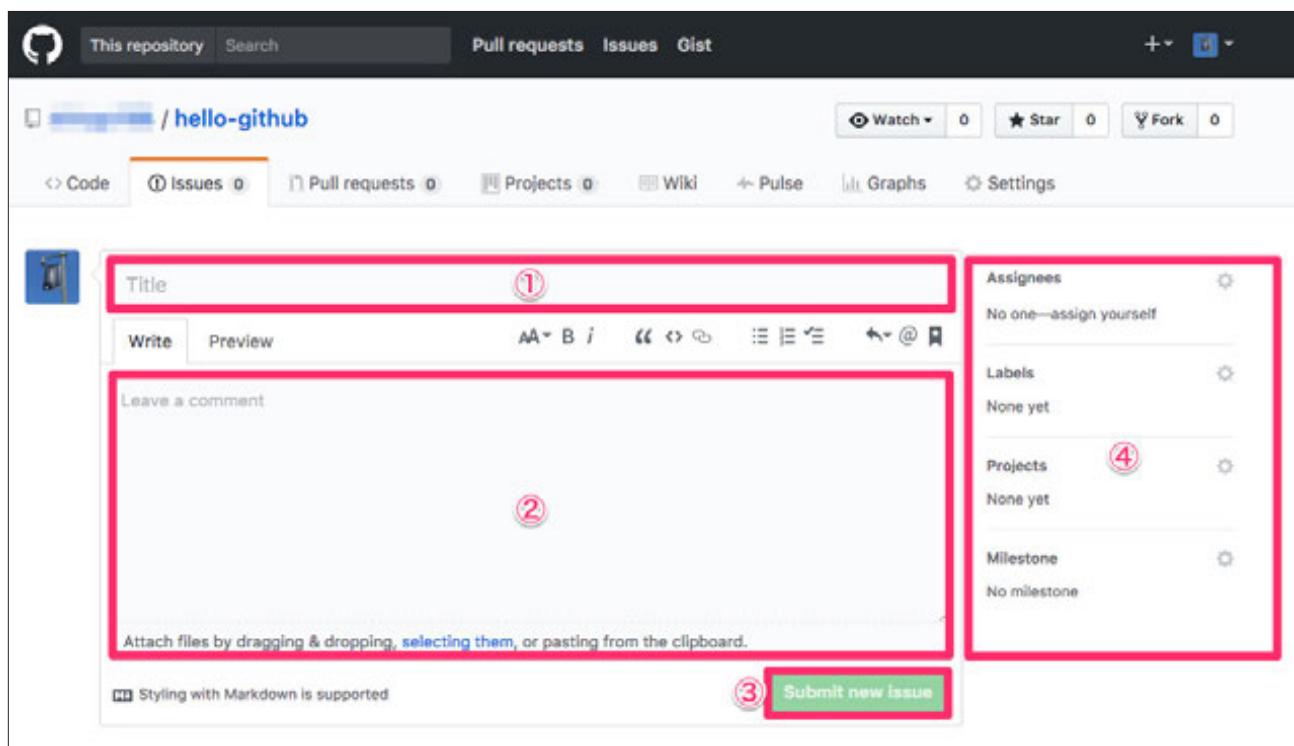


図 3 イssue作成ページ

今回は架空の「音楽プレイヤーアプリ」のタスクを例に、イシューを1件作成していきます。

【1】タイトル

今回は例として「音楽再生画面という画面を1つ追加する」という「新機能追加タスク」をイシューにしてみることにします。

タイトルとして「音楽再生画面の作成」を設定します。

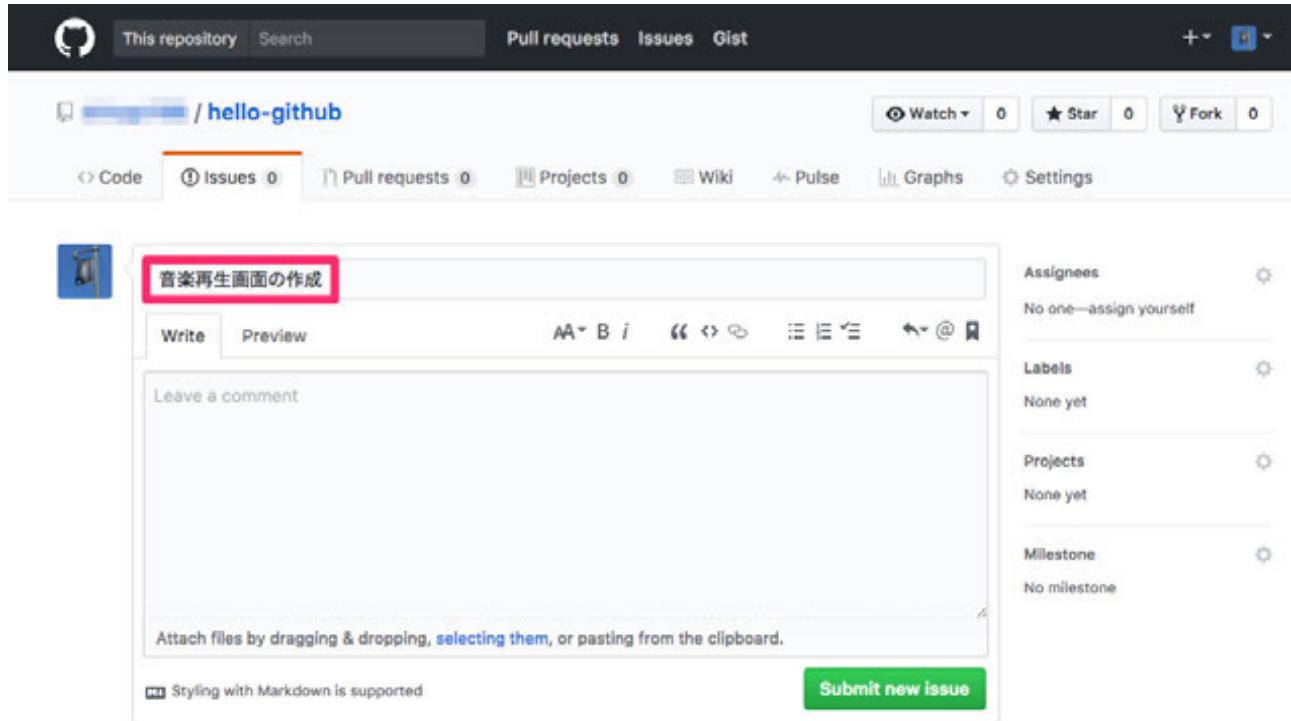


図 4 タイトルを設定

【2】説明

イシューの「説明」を追加していきます。

説明記入用のテキストボックスの機能はプルリクエスト作成の場合と同様です。Markdown 形式で記入できます。

- タスクリストを追加する

以下のフォーマットでリストを記述すると、イシューの説明やコメントに「タスクリスト」を挿入できます。

- [] 項目 1
- [] 項目 2
- [] 項目 3

今回は以下のように、タスクを完了するまでにやることのリストを追加してみます。



This screenshot shows the GitHub Issues page for the repository 'hello-github'. A new issue has been created with the title '音楽再生画面の作成'. The issue body contains the following text:

```
#やること
- [] プレイヤー機能を実装する
- [] プロジェクトにUI素材を追加する
- [] UIコンポーネントを配置する
- [] 動作を確認する
```

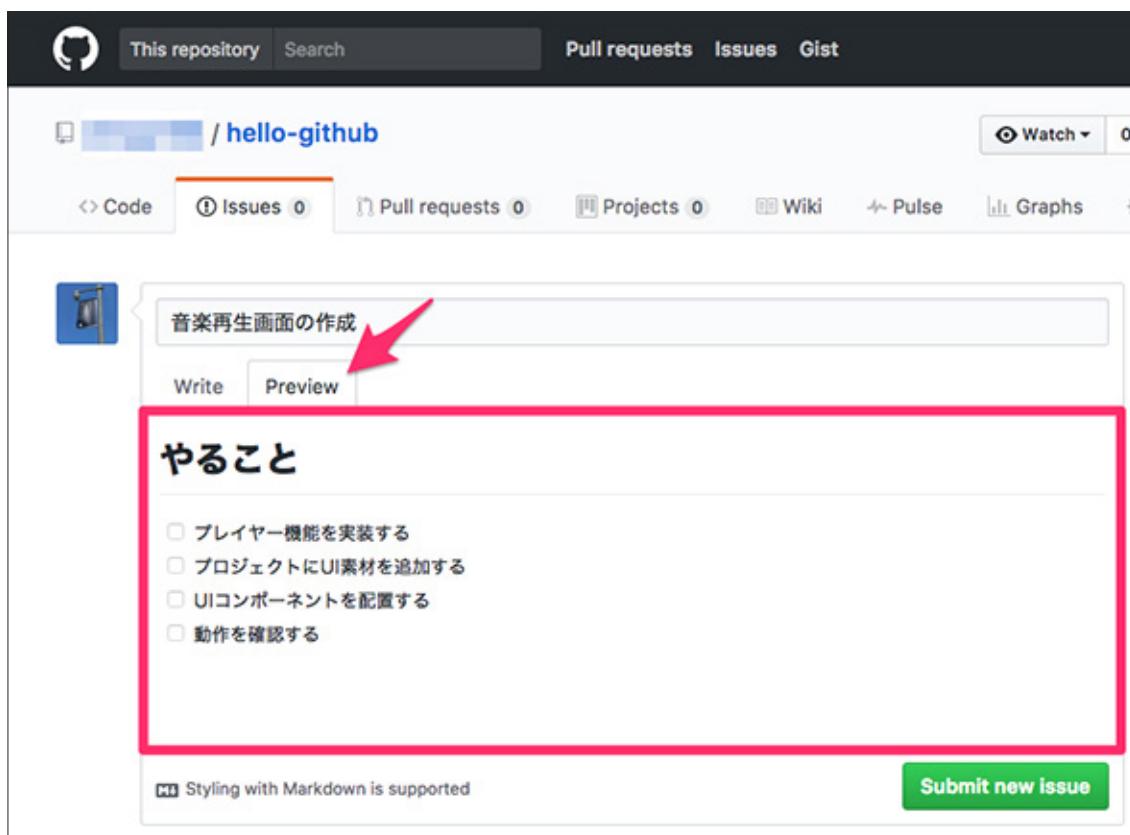
The 'Preview' tab is selected, and the rendered content is displayed below:

やること

- プレイヤー機能を実装する
- プロジェクトにUI素材を追加する
- UIコンポーネントを配置する
- 動作を確認する

図 5 タスクリストを追加

「Preview」をクリックすると、以下のようにレンダリングされます。



This screenshot shows the GitHub Issues page for the repository 'hello-github'. The 'Preview' tab is selected, and the rendered content is displayed below:

やること

- プレイヤー機能を実装する
- プロジェクトにUI素材を追加する
- UIコンポーネントを配置する
- 動作を確認する

図 6 タスクリスト追加後のレンダリング結果

- ファイルを添付する

次にファイルの添付機能を使ってみます。GitHub のヘルプページの説明によると、イシューには以下の形式のファイルを添付できます。

- 画像
 - PNG (.png)
 - GIF (.gif)
 - JPEG (.jpg)

- 文書
 - Microsoft Word (.docx)
 - Microsoft Powerpoint (.pptx)
 - Microsoft Excel (.xlsx)
 - テキストファイル (.txt)
 - PDF (.pdf)

- ZIP (.zip, .gz)

今回は例として PNG ファイルを添付してみます。

ファイルを添付するには、ファイルを「説明記入用のテキストボックス」にドラッグ&ドロップします。

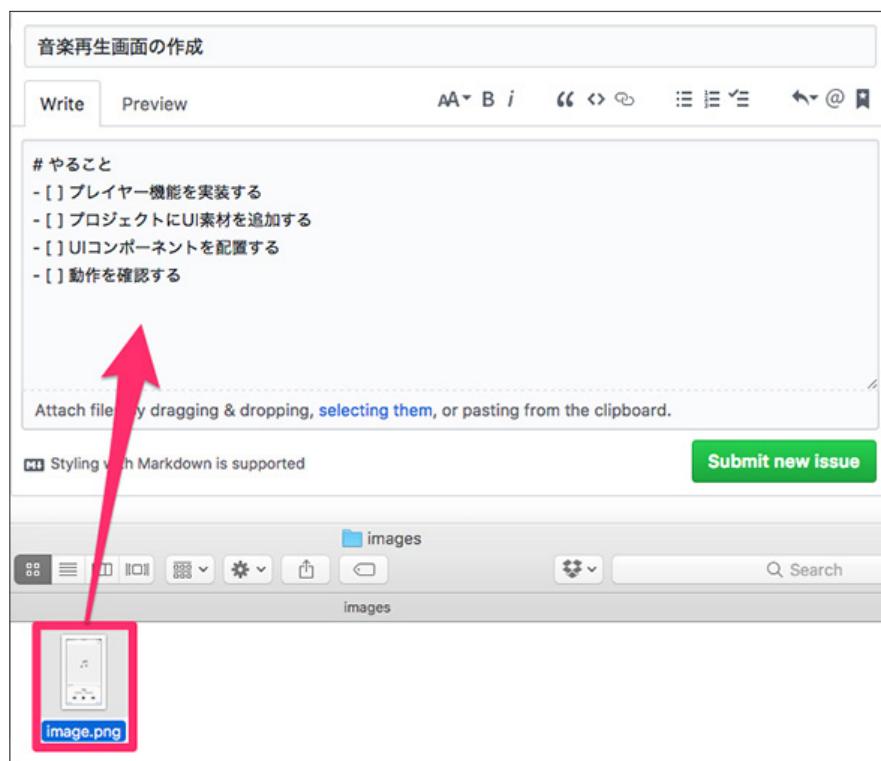


図 7 ファイルの添付

ファイルのアップロードが開始すると、以下のように「![Uploading {ファイル名} ...]()」という形式の文字列が入ります。

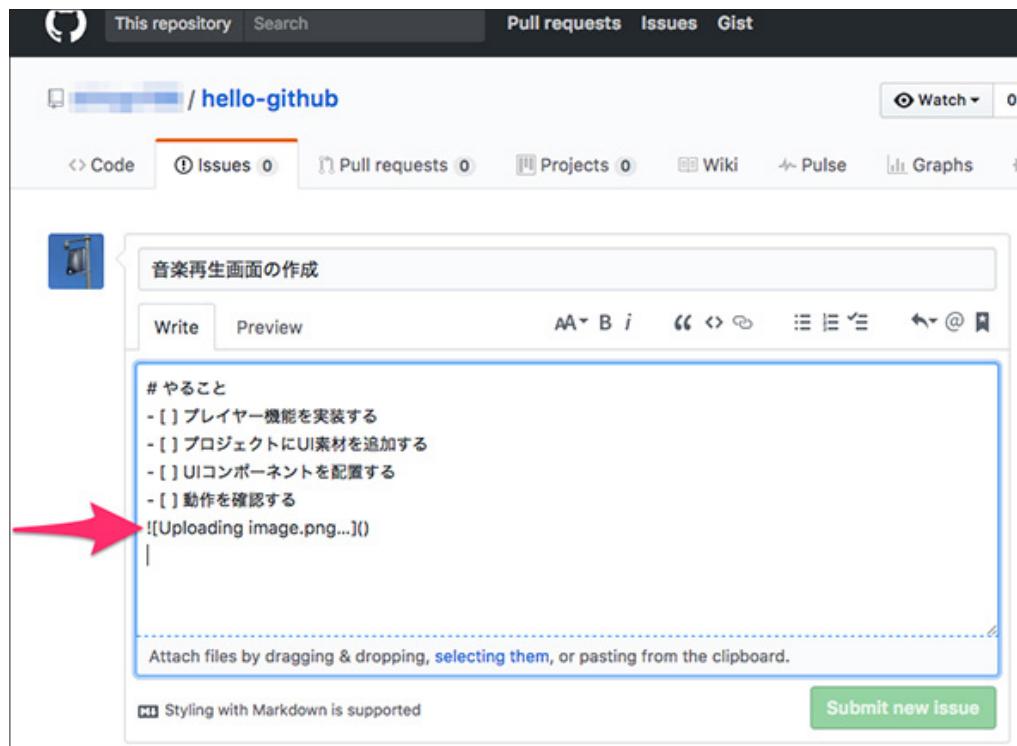


図 8 ファイルのアップロード

アップロードが完了すると、以下のように「![{ファイル名}]({ファイルの URL})」という形式の文字列になります。

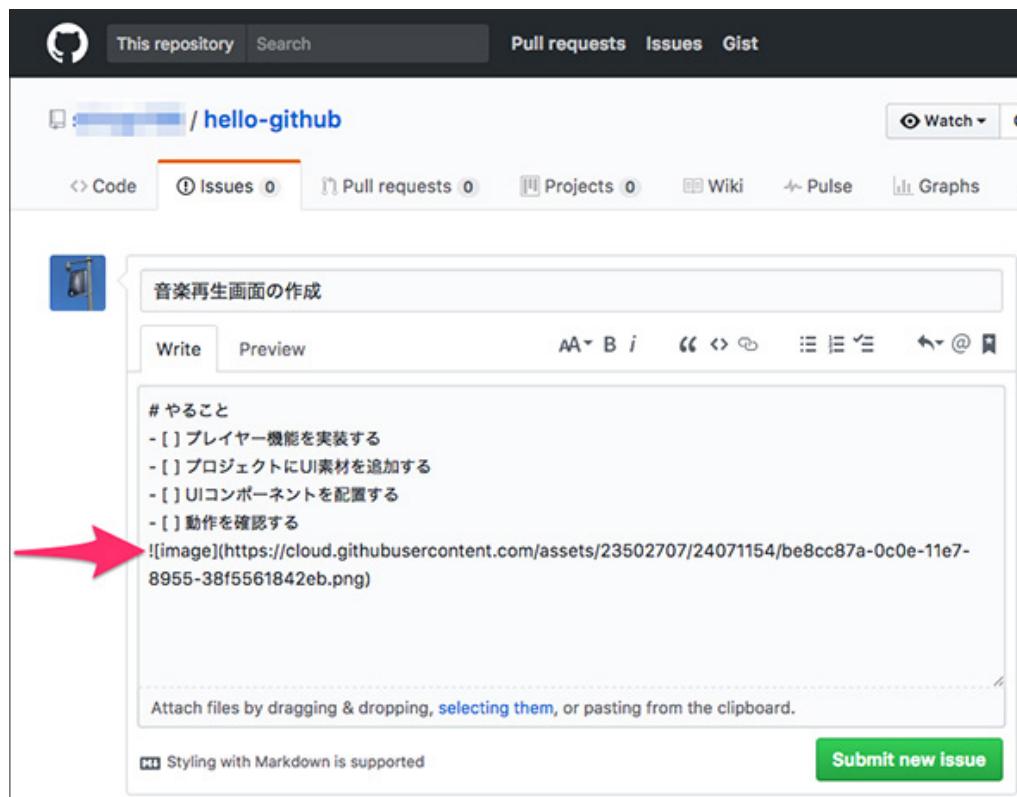


図 9 ファイルのアップロード完了

改行と見出しを追加して、構成を整えます。

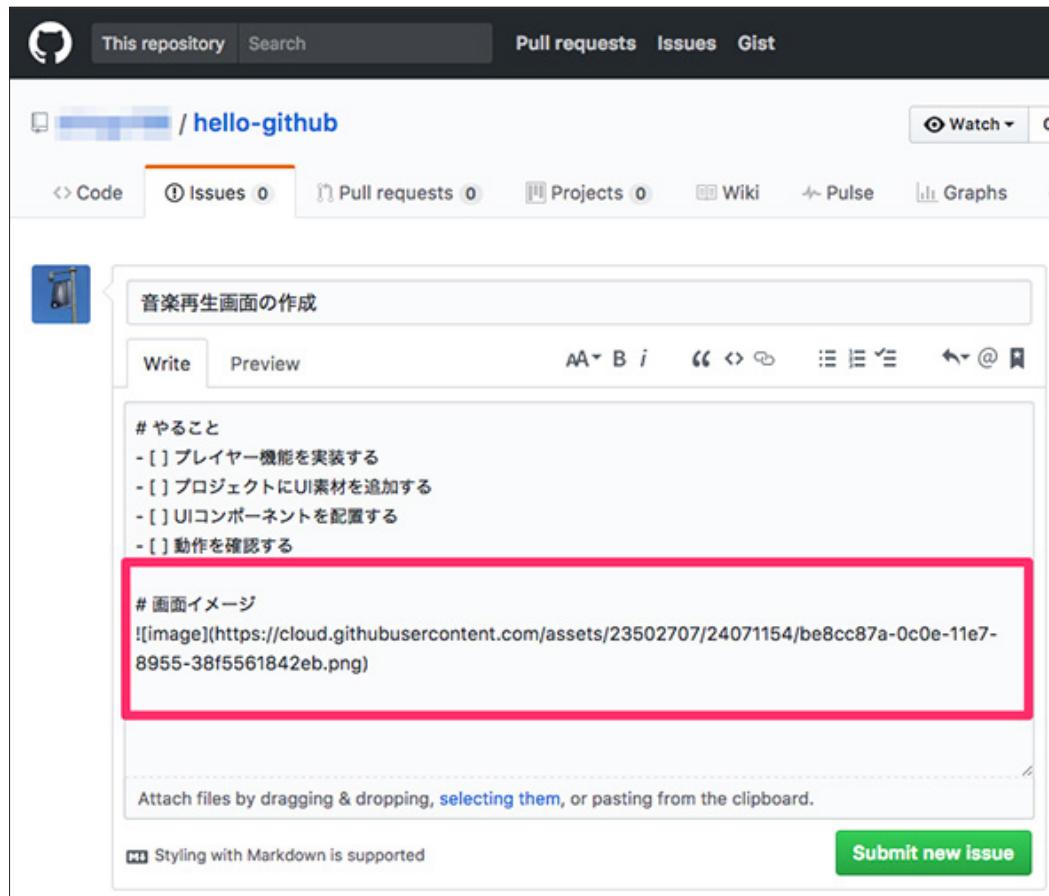


図 10 改行と見出しを追加

「Preview」をクリックすると、以下のようにレンダリングされます。



図 11 ファイル添付後のレンダリング結果

・リンクを追加する

「[{タイトル}]({Web ページの URL})」というフォーマットで記述すると、Web ページへのリンクを挿入できます。

今回は参考資料へのリンクを追加してみます。

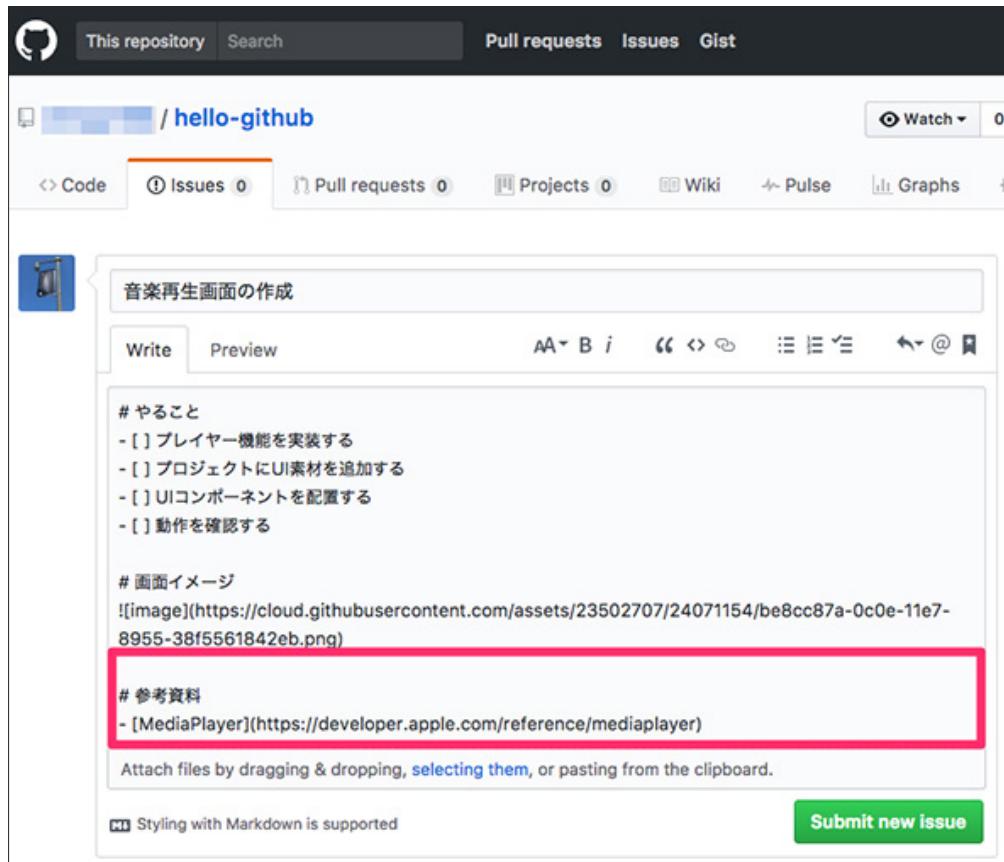


図 12 リンクを追加

「Preview」をクリックすると、以下のようにレンダリングされます。

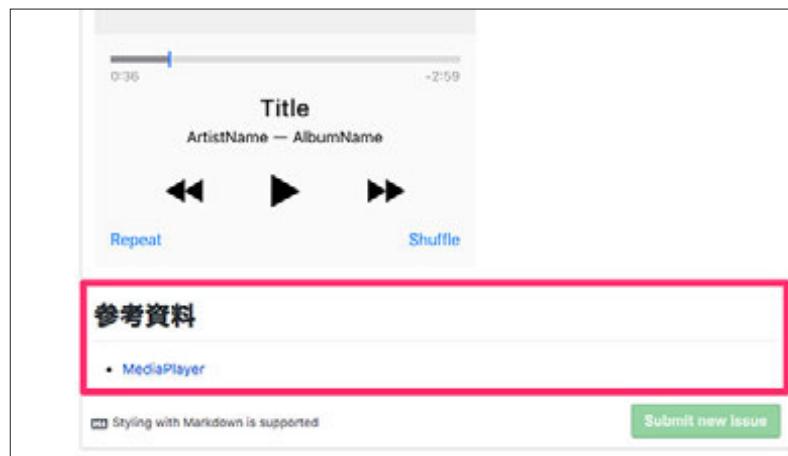


図 13 リンク追加後のレンダリング結果

【3】イシューの担当者とラベル

イシューの担当者（Assignees）とラベル（Labels）の設定を行ってみます。こちらもイシューとプルリクエストの両方で利用できる機能です。

・担当者を設定する

担当者を設定することによって、誰が担当するイシューなのかを明確にできます。また、担当者に設定された人は該当イシューに関する通知を受けることができるようになります。

担当者を設定するには、サイドバー上の「Assignees」またはその横の歯車アイコンをクリックします。

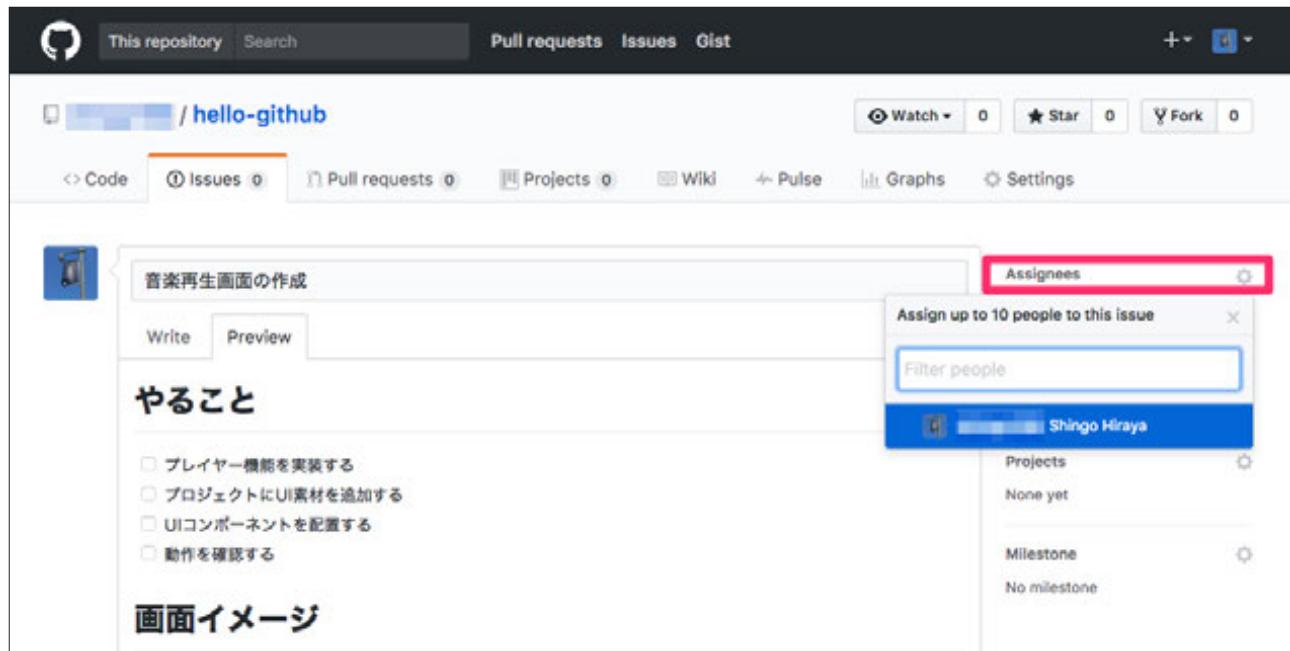


図 14 担当者選択メニューを表示

担当者にしたい人を選択して「×」をクリックします。

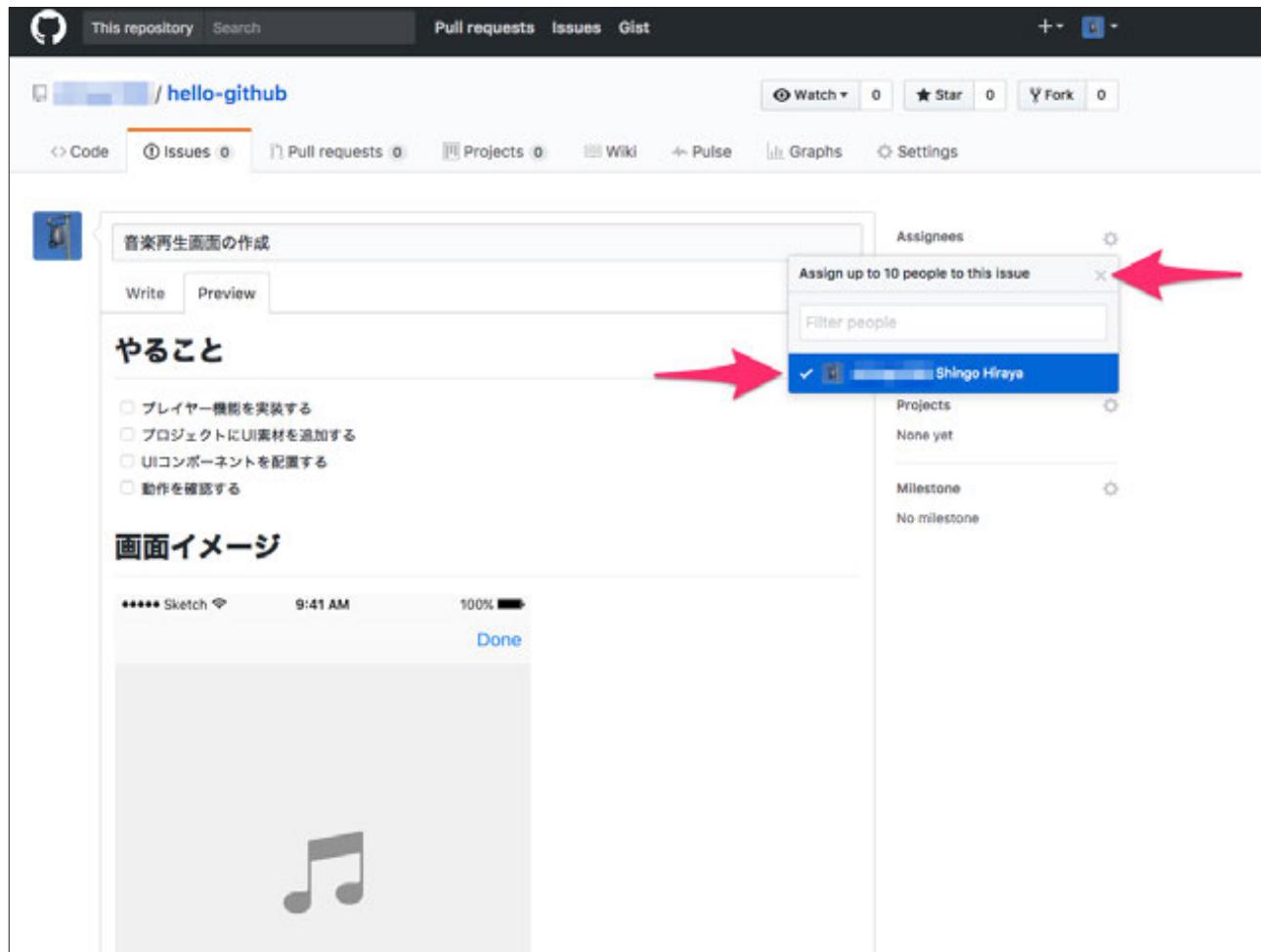


図 15 担当者を選択

担当者を設定できました。

The screenshot shows the GitHub interface for a repository named 'hello-github'. In the top navigation bar, 'Issues' is selected. On the right side of the main content area, there is a sidebar with several settings: 'Assignees' (which has a red box around it), 'Labels' (set to 'None yet'), and 'Projects' (set to 'None yet'). The main content area contains a note about creating a music player interface and a 'やること' (Things to do) section with two unchecked tasks: 'プレイヤー機能を実装する' and 'プロジェクトにUI素材を追加する'.

図 16 担当者設定完了

・ラベルを設定する

ラベルはイシューを分類するための機能です。イシューの数が増えてきた場合に役に立つと思います。

ラベルを設定するには、サイドバー上の「Labels」またはその横の歯車アイコンをクリックします。

This screenshot is similar to Figure 16, showing the GitHub interface for the 'hello-github' repository. The 'Labels' field in the sidebar is highlighted with a red box. A modal window titled 'Apply labels to this issue' is open on the right, containing a 'Filter labels' input field and a list of seven label categories: 'bug' (red square), 'duplicate' (grey square), 'enhancement' (blue square), 'help wanted' (green square), 'invalid' (light grey square), 'question' (pink square), and 'wontfix' (white square). The 'enhancement' label is currently selected, indicated by a blue border around its square icon.

図 17 ラベル選択メニューを表示

デフォルトで選択可能なラベルは 7 種類です。設定したいラベルを選択して「×」をクリックします。

今回は「enhancement」（改良）を選択してみます。

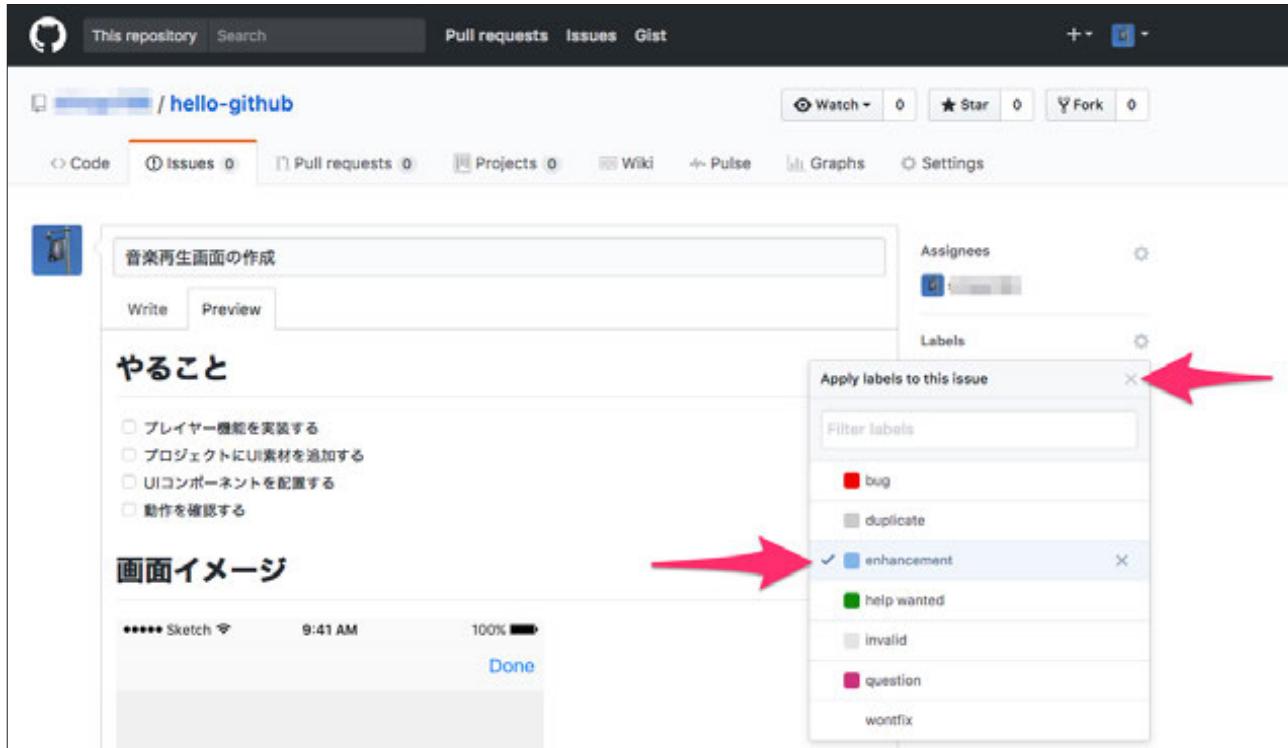


図 18 ラベルを選択

ラベルを設定できました。

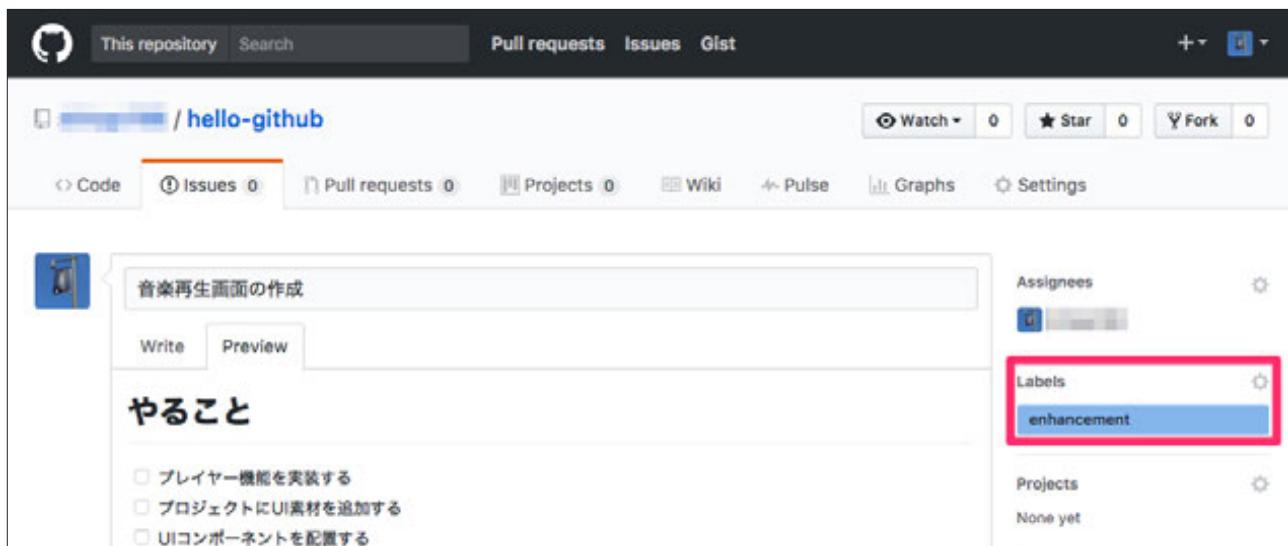


図 19 ラベル設定完了

【4】イシューを作成する

ここまでで「タイトル」「説明」「担当者」「ラベル」の設定が完了しました。

イシューの作成を実行しましょう。「Submit new issue」ボタンをクリックします。



図 20 イシューの作成実行

イシューの作成が完了しました。作成したイシューのページが表示されます。

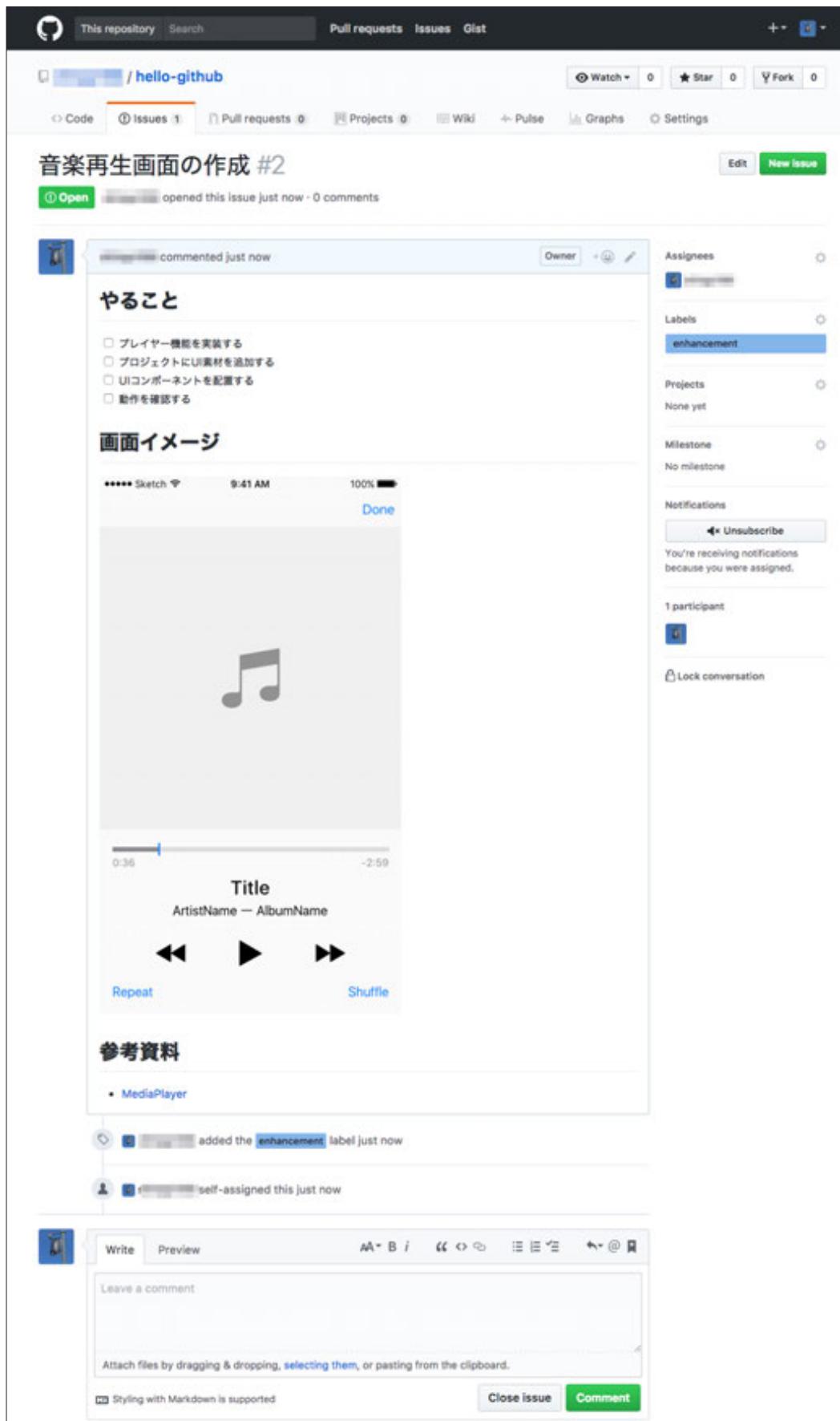


図 21 イシューの作成完了

イシューの更新

タスクリストを更新する

タスクリストの各アイテムの左にあるチェックボックスをクリックすると、そのアイテムを完了表示にできます。



図 22 タスクリストを更新

説明を更新する

説明を更新するには、ペンアイコンのボタンをクリックします。

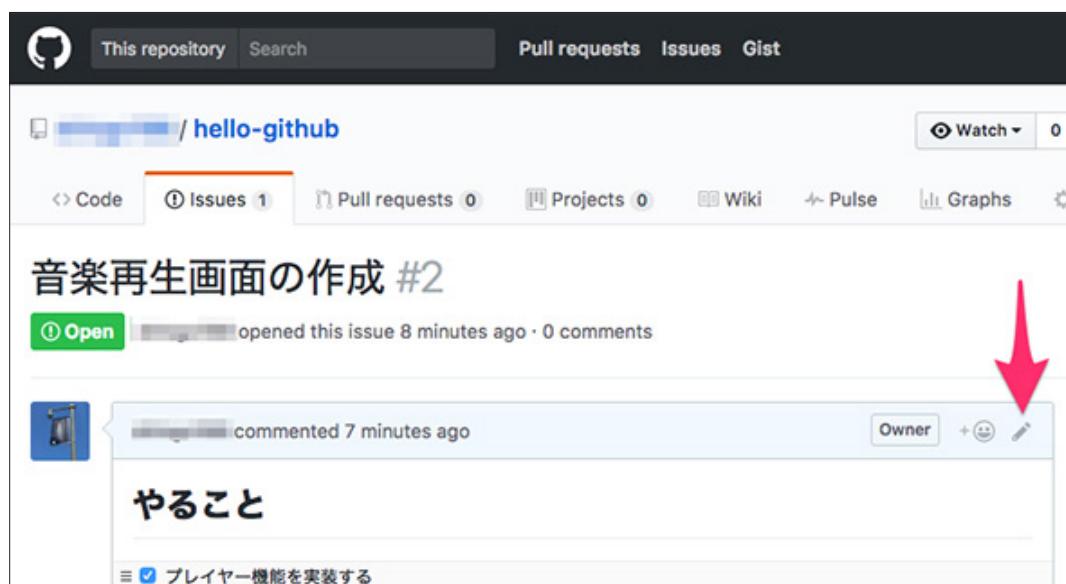


図 23 説明の更新

テキストボックスの中身を更新し、「Update comment」をクリックすると説明の更新が実行されます。



図 24 説明の更新実行

コメントを追加する

プルリクエストのページと同様に、イシューのページにもコメント欄があります。

ここにコメントを記入し、「Comment」ボタンをクリックするとコメントを追加できます。説明記入用のテキストボックスと同じ機能を使用できます。

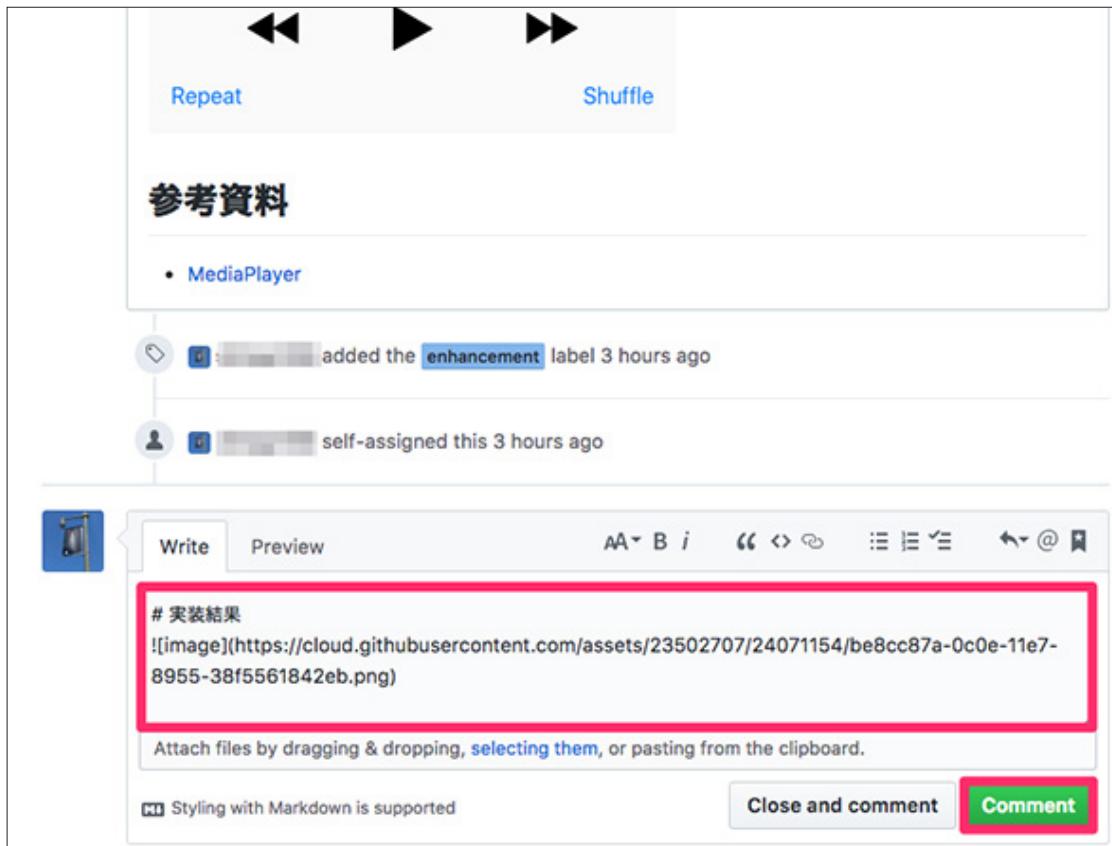


図 25 コメントを追加

イシューを閉じる

作成したイシューが不要になったり、イシューで管理しているタスクが完了したりしたら、一番下の「Close issue」をクリックしてイシューを閉じることができます。

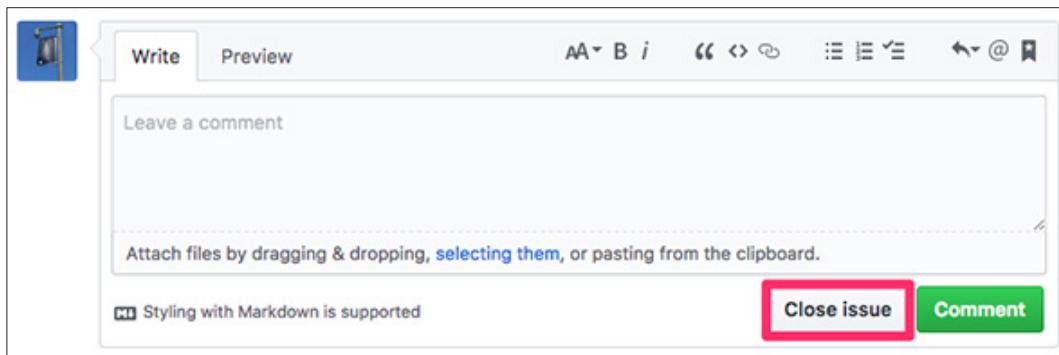


図 26 イシューを閉じる

イシューの一覧ページ

最後にイシューの一覧ページの機能を解説します。

リポジトリページの「Issues」タブをクリックして、イシューの一覧ページを表示します。

イシューの一覧ページでは、フィルタ機能を利用して目的のイシューを絞り込んだり、複数のイシューに対して一括操作を行ったりすることができます。

図 27 イシューの一覧ページ

フィルタ機能を利用する

イシューのリストの右上のボタンを押すと、イシューの作者、ラベル、担当者などで絞り込むためのメニューを表示できます。

イシューの作者で絞り込むには、以下のように「Author」をクリックし、リスト上のユーザーをクリックします。

図 28 イシューの作者で絞り込む

複数のイシューに対して一括操作を行う

複数のイシューに対して一括操作を行うには、対象のイシューの左のチェックボックスをチェックします。

イシューのリストの右上のボタンが一括操作用のボタンに変わるので、いずれかのボタンをクリックし、リスト上のアクションをクリックします。

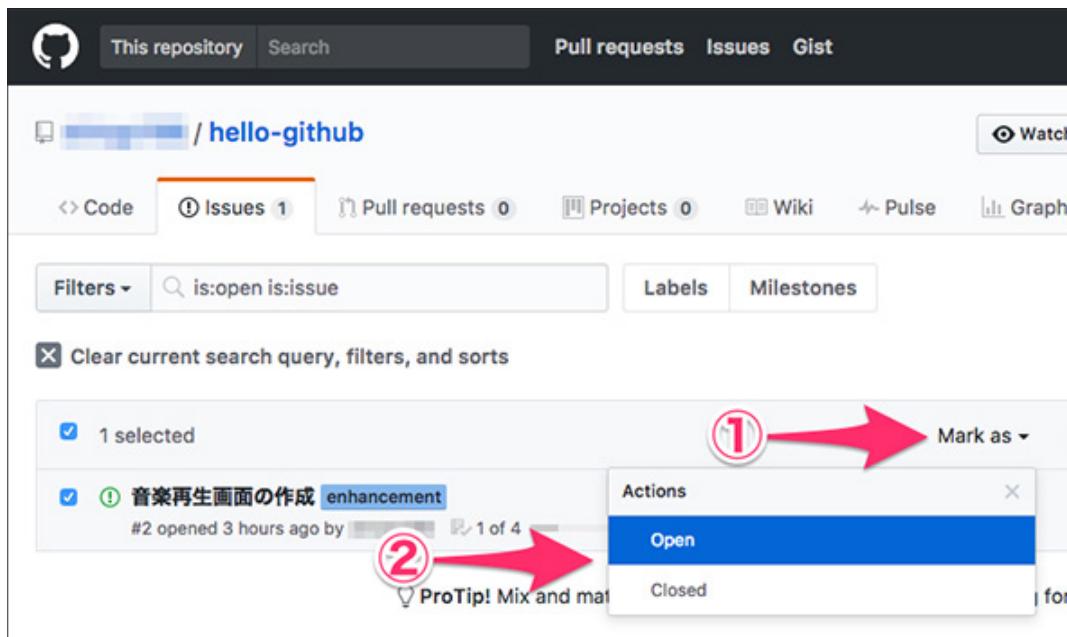


図 29 イシューの状態を一括変更する

クローズしたイシューを表示する

デフォルトでは、進行中のイシュー（クローズしていないイシュー）の一覧が表示されます。

クローズしたイシューを表示するには、イシューのリストの上の「Closed」をクリックします。

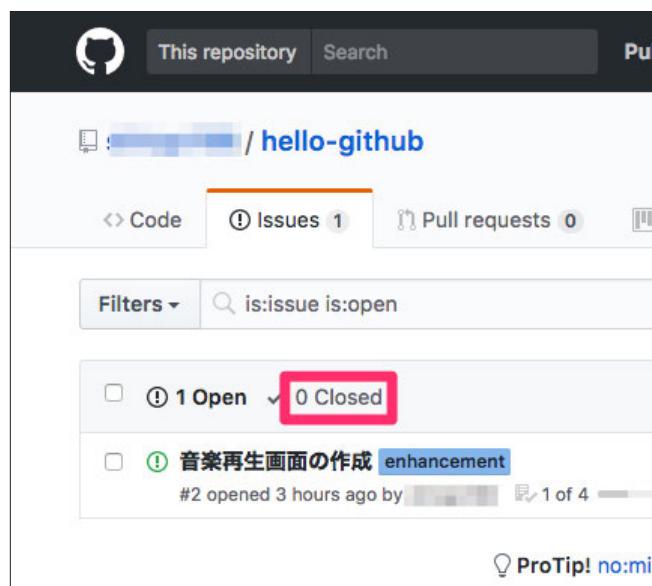


図 30 クローズしたイシューを表示

クローズしたイシューを表示するフィルタが適用されます。今回扱っているリポジトリにクローズしたイシューはないので、以下のような表示になります。

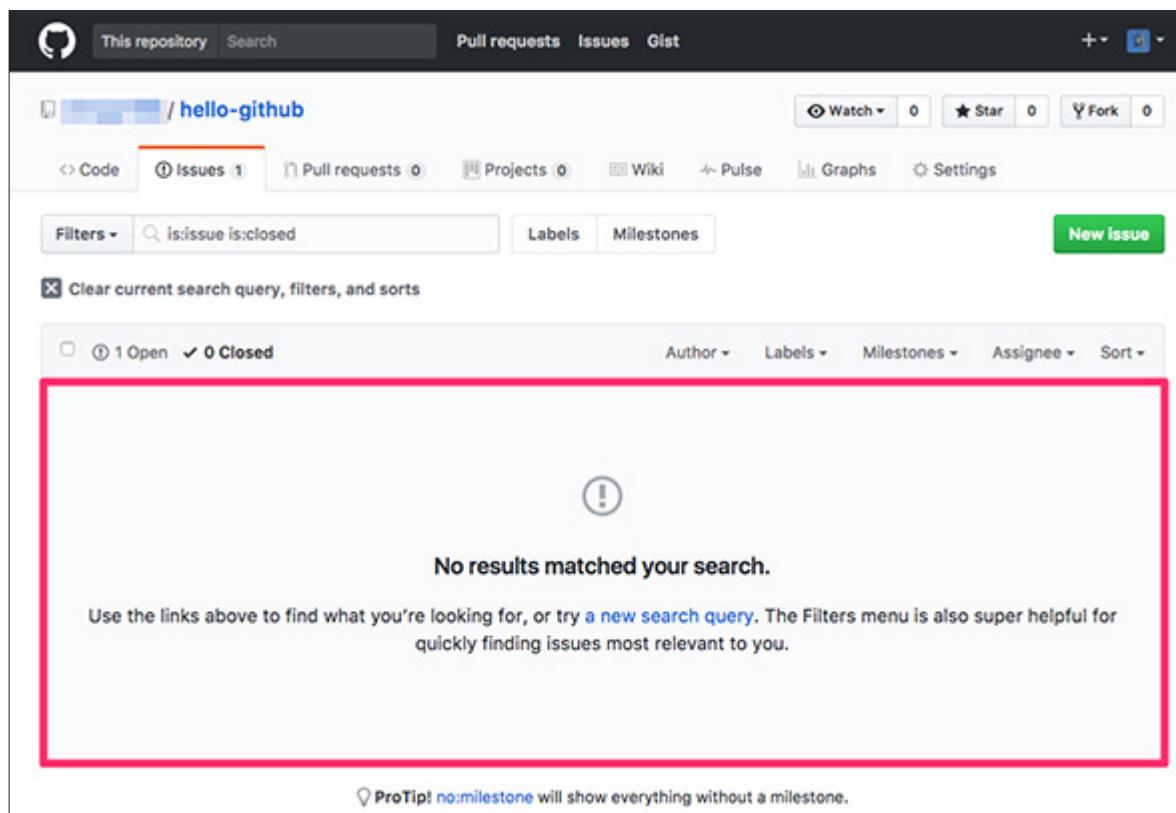


図 31 クローズしたイシューの一覧

次回は「イシューの周辺機能」について

本稿では「イシュー (Issues)」に対する基本操作を解説しました。次回は「イシューの周辺機能」を解説する予定です。お楽しみに!

参考

- [Managing your work with issues - GitHub Help](#)
- [Finding information in a repository - GitHub Help](#)

12. 開発者のスケジュール管理に超便利、GitHub Issues、Label、Milestone、Projects 使いこなし術

(2017年04月28日)

本連載では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説しています。今回は、イシューとプルリクエストの連携やテンプレート設定、GitHub Projects、ラベル、マイルストーンなどを紹介します。

GitHub のイシューとプルリクエストの周辺機能でスケジュール管理が容易に

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終える頃には、GitやGitHubの基本的な操作が身に付いた状態になっていると思います。

前回の記事「開発者のタスク管理がしやすくなる GitHub Issues の基本的な使い方」では「イシュー」に対する基本操作を解説しました。

連載第12回目の本稿では、イシューとプルリクエストの連携やテンプレート設定、GitHub Projects、ラベル、マイルストーンなど、「イシューとプルリクエストの周辺機能」を解説していきます。

イシューとプルリクエストの連携

まずは、プルリクエストがマージされたら関連するイシューを自動的にクローズする方法を紹介します。

これを実現するには、以下の形式の文字列を「プルリクエストのコメント」または「プルリクエストに関連するコミット」のメッセージに含めます。

```
{キーワード} # {イシュー番号}
```

GitHubのヘルプページの説明によると、以下の「キーワード」を使ってイシューを閉じることができます。

- close
- closes
- closed
- fix
- fixes
- fixed
- resolve
- resolves
- resolved

「イシュー番号」はイシューのページなどで確認できます。

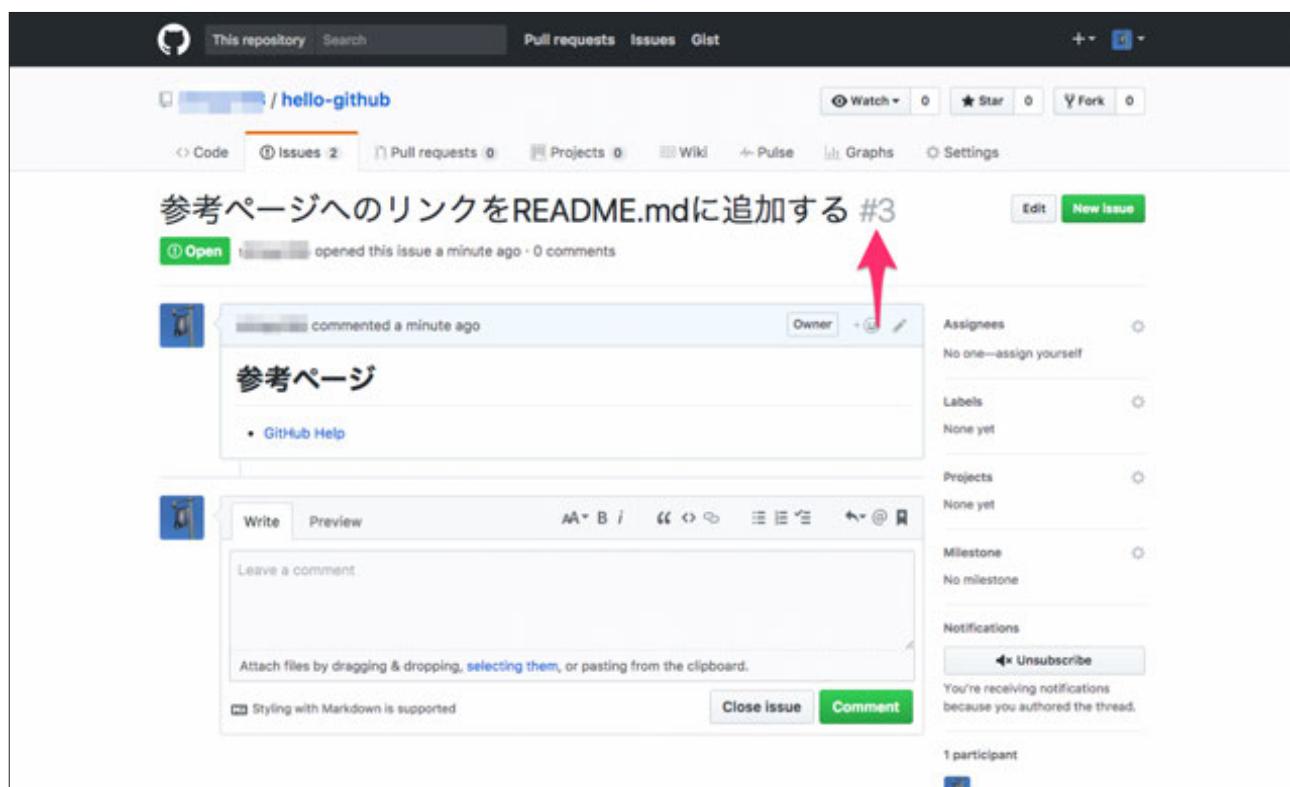


図 1 イシューのページ上の「イシュー番号」

例えば、イシュー番号 3 のイシューを閉じるには、以下の文字列をプルリクエストのコメントまたはコミットメッセージに含めます。

```
fix #3
```

以下の例では、プルリクエストのコメントに文字列「fix #3」を入れています。

This screenshot shows a GitHub pull request page for a repository named 'hello-github'. The pull request is titled 'Update README.md #4'. A comment from a user named 'fix #3' is highlighted with a red arrow. The comment text is 'fix #3'. Below the comment, there is a green box stating 'This branch has no conflicts with the base branch'. At the bottom of the page, there is a 'Merge pull request' button.

図 2 プルリクエストのコメントに文字列「fix #3」を記載

このプルリクエストをマージしてみます。

This screenshot shows the same GitHub pull request page as before, but it has been merged. The status bar at the top now says 'Merged'. The merge message is 'merged 1 commit into master from fix/readme just now'. The comment from 'fix #3' is still present. A new message from the same user 'fix #3' appears, stating 'merged commit 0a6693a into master just now'. There is also a note about continuous integration: 'Avoid bugs by automatically running your tests.' and 'deleted the fix/readme branch just now'.

図 3 マージ後のプルリクエストページ

イシュー番号 3 のイシューを開くと、クローズされていることを確認できます。

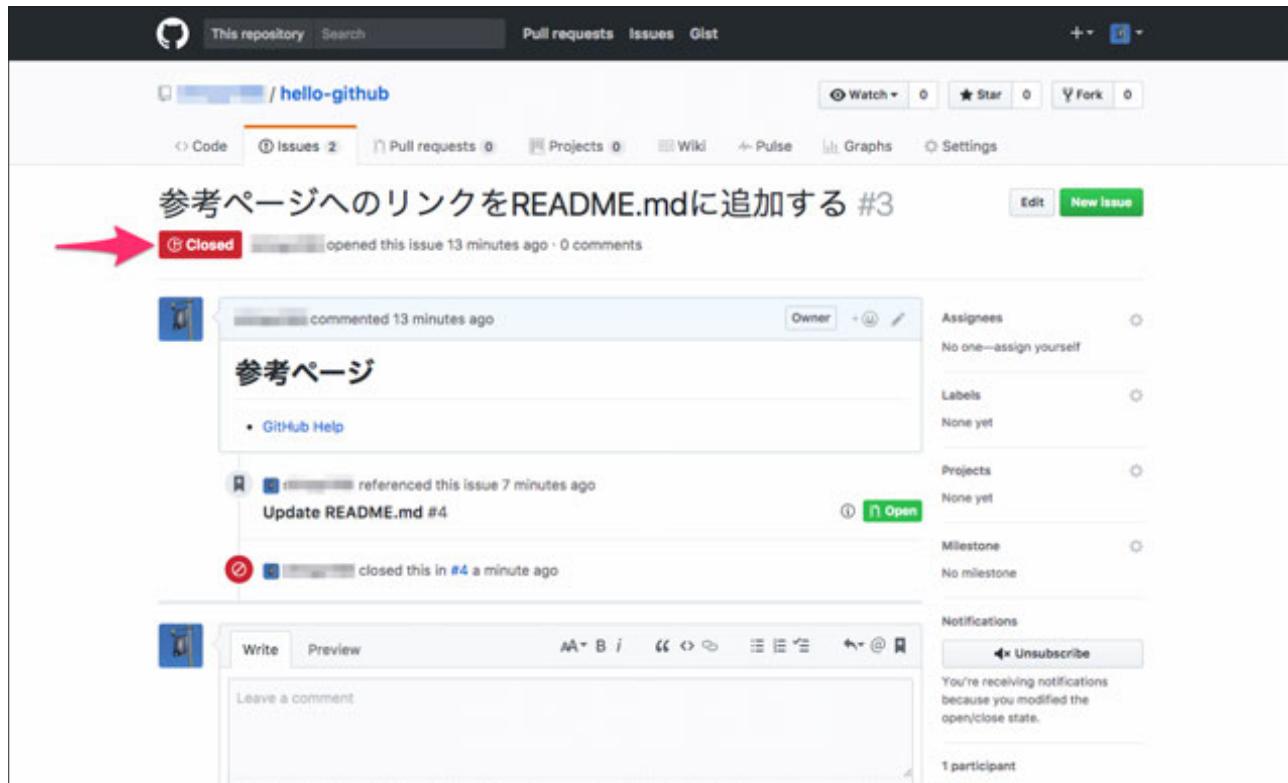


図 4 自動クローズ後のイシューのページ

イシューとプルリクエストのテンプレート設定

イシューまたはプルリクエストのテンプレートを設定するには、以下の名前のファイルをリポジトリ直下（または .github ディレクトリ内）に配置します。

- イシューのテンプレートの場合
 - ISSUE_TEMPLATE.md (拡張子は任意)
- プルリクエストのテンプレートの場合
 - PULL_REQUEST_TEMPLATE.md (拡張子は任意)

以下の例では、リポジトリ直下に「ISSUE_TEMPLATE.md」と「PULL_REQUEST_TEMPLATE.md」のファイルを配置しています。

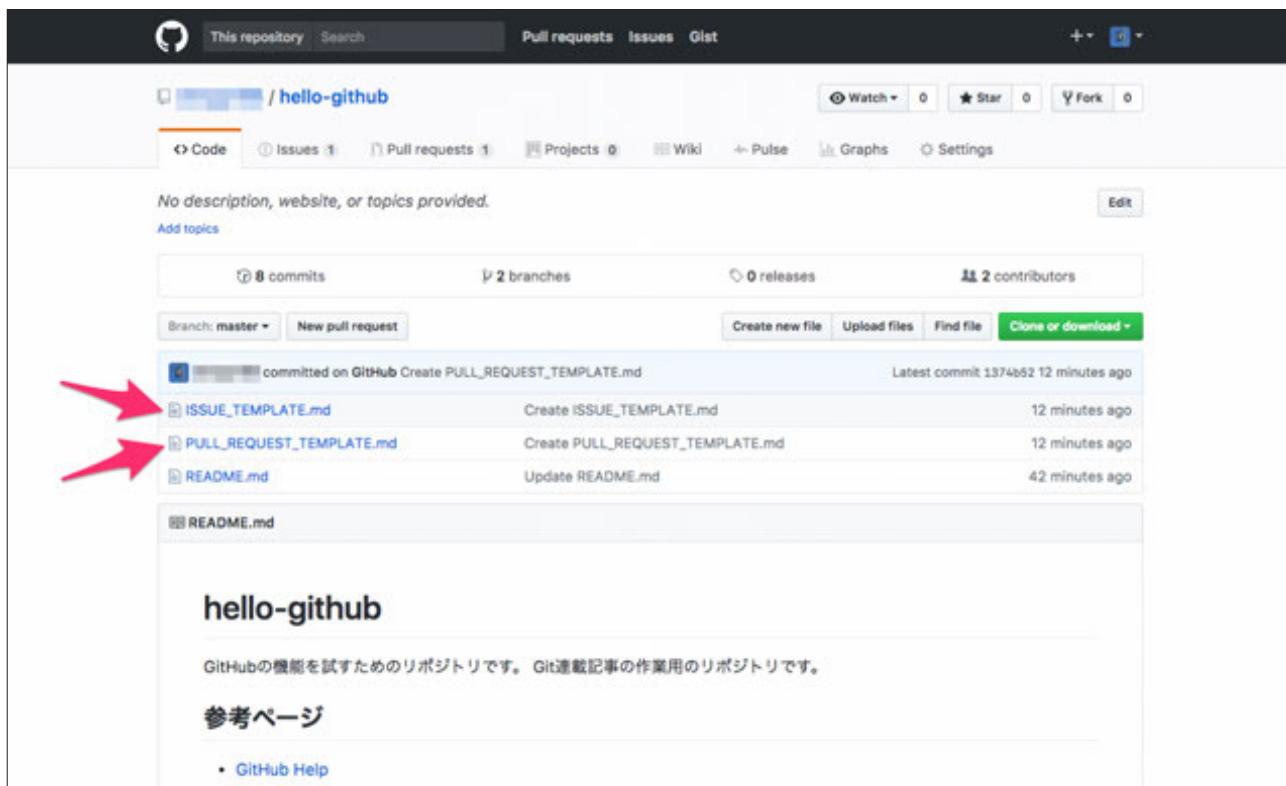


図 5 2つのファイルをリポジトリ直下に配置

イシューを新規に作成すると「ISSUE_TEMPLATE.md」の内容が自動で挿入されます。

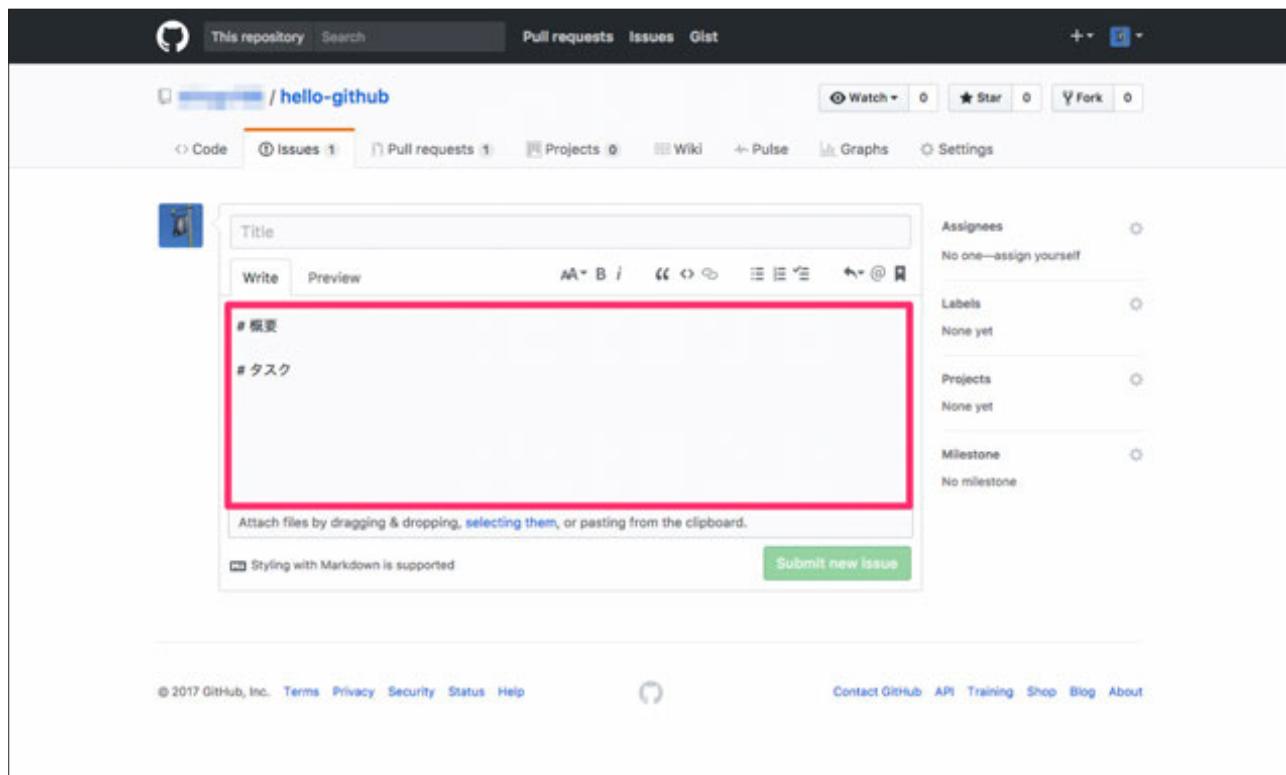


図 6 新規作成イシューへのテンプレート自動挿入

プルリクエストの場合も同様に、新規作成時に「PULL_REQUEST_TEMPLATE.md」の内容が自動で挿入されます。

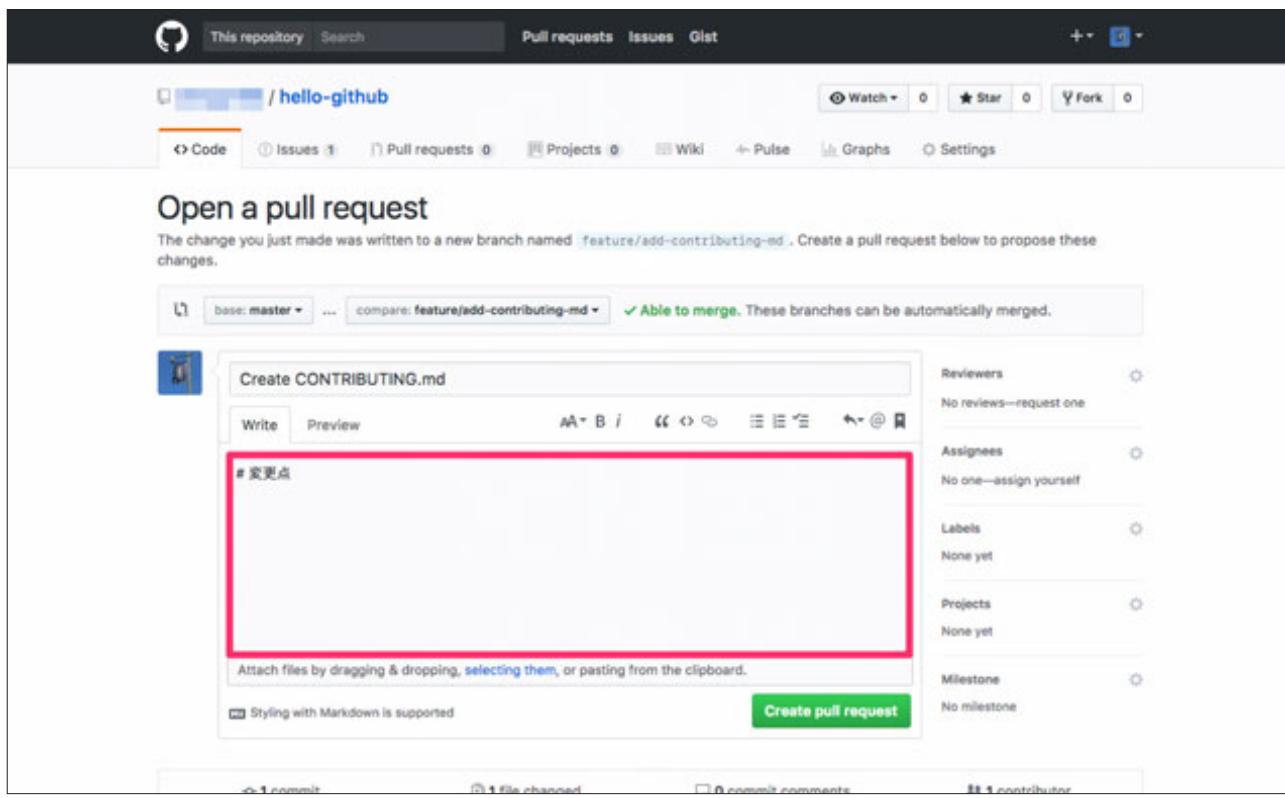


図 7 新規作成プルリクエストへのテンプレート自動挿入

GitHub Projects の基本的な使い方

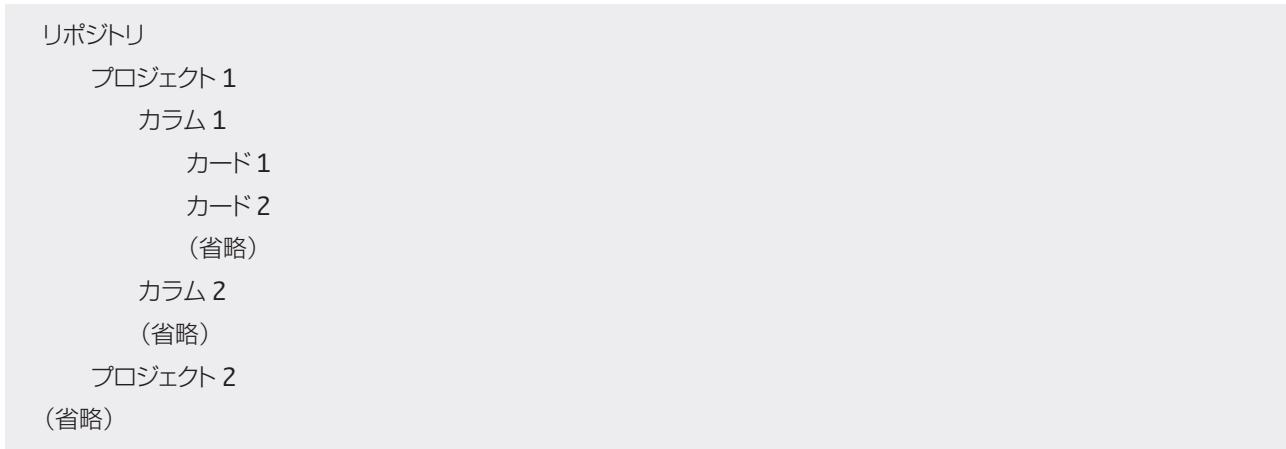
「Projects」はプロジェクト管理サービス「Trello」「Waffle」のような「カンバン方式」のタスク管理機能です。

Projects を使用して「ソフトウェア開発に関連するタスクの管理」を行えば、「今どのようなタスクが進行しているのか」「そのタスクが今どのような状態にあるのか」などを把握することができます。

Projects の機能は「プロジェクト」「カラム」「カード」の 3 つの要素で構成されます。

プロジェクトは 1 つのリポジトリに対して複数作成でき、プロジェクトには複数の「カラム」を追加できます。そしてさらに「カラム」の中に複数の「カード」(1 つのタスクを表す要素) を追加できます。「カラム」は「カード」をグルーピングするために使用します。

リポジトリ、プロジェクト、カラム、カードの階層をまとめると以下のようにになります。



プロジェクトを作成する

プロジェクトを作成してみましょう。GitHub 上のリポジトリページを表示し「Projects」タブをクリックします。

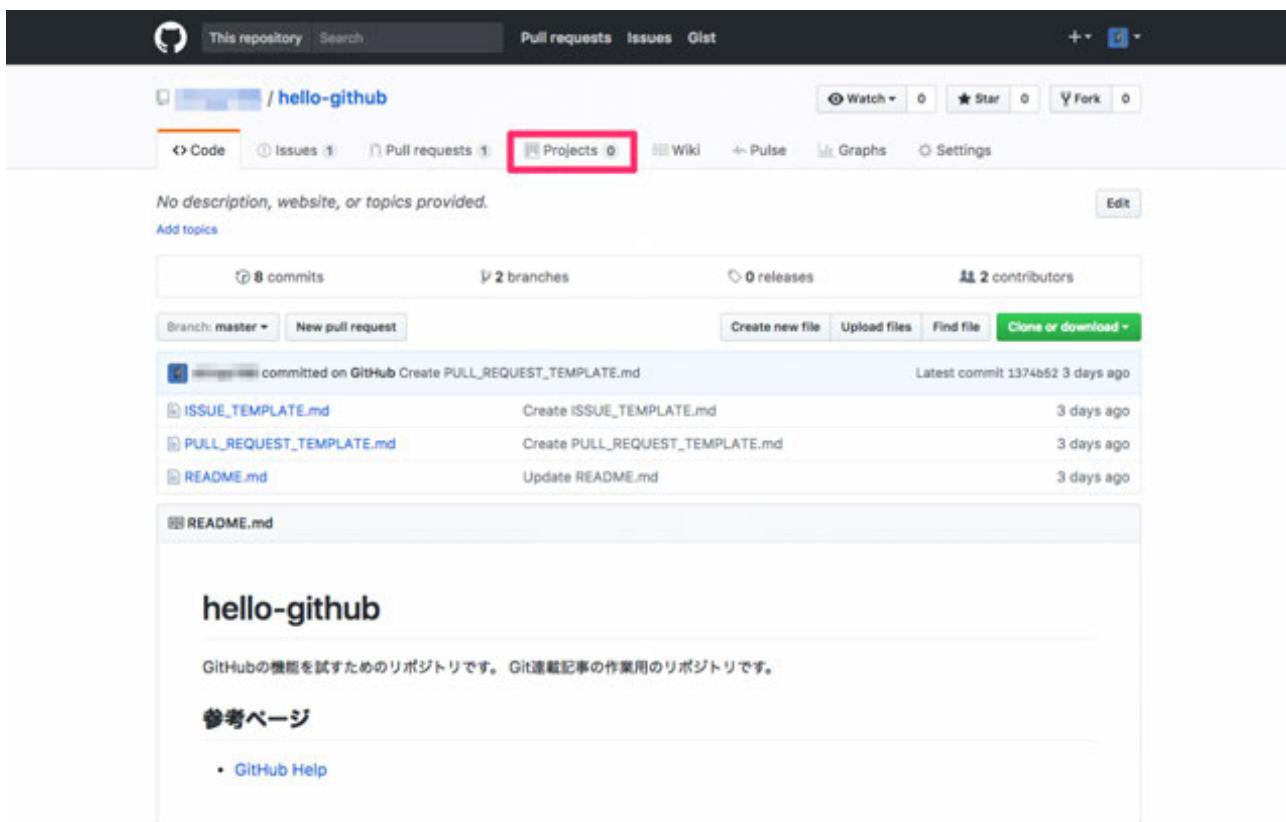


図 8 「Projects」タブをクリック

プロジェクトを 1 つも作成していない場合、以下のような表示になります。「Create a project」をクリックします。

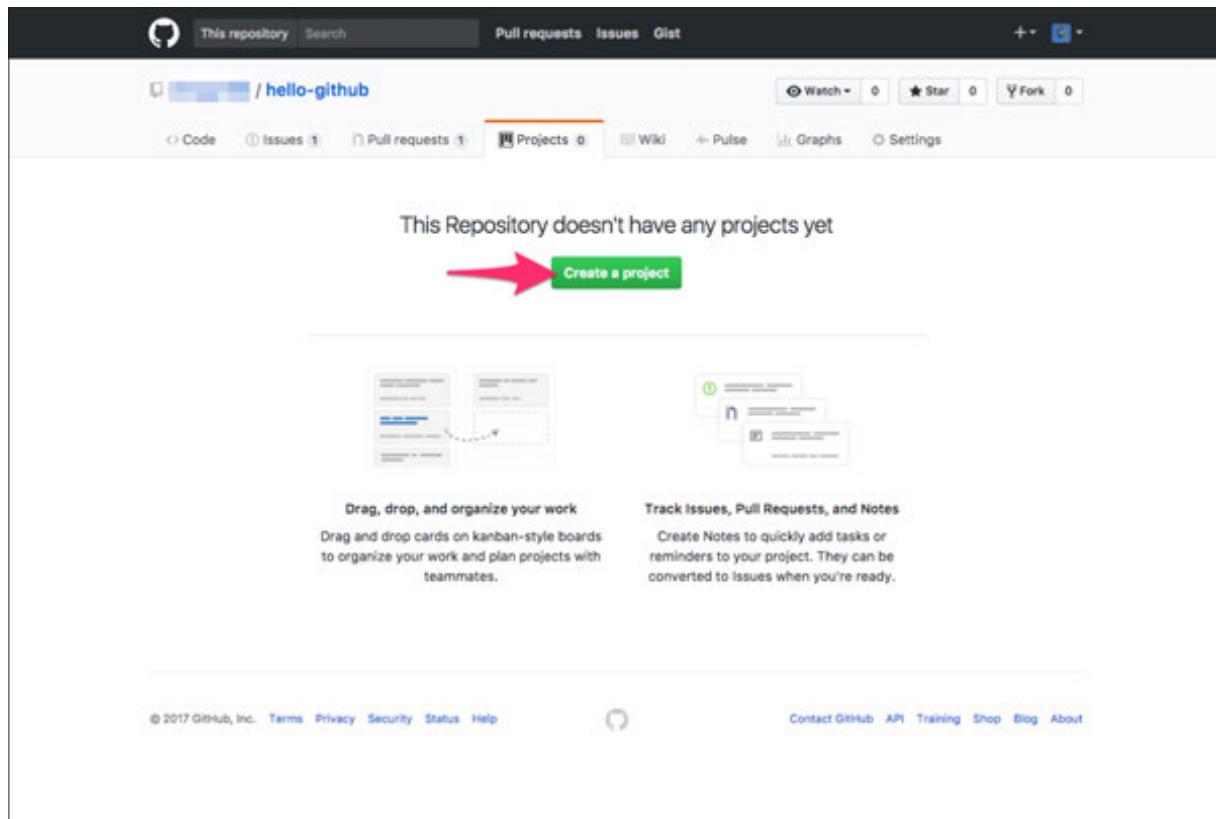


図9 「Projects」タブ

プロジェクト作成ページが表示されます。Name にプロジェクトの名前を、Description にプロジェクトの説明を記入し、「Save project」をクリックします。

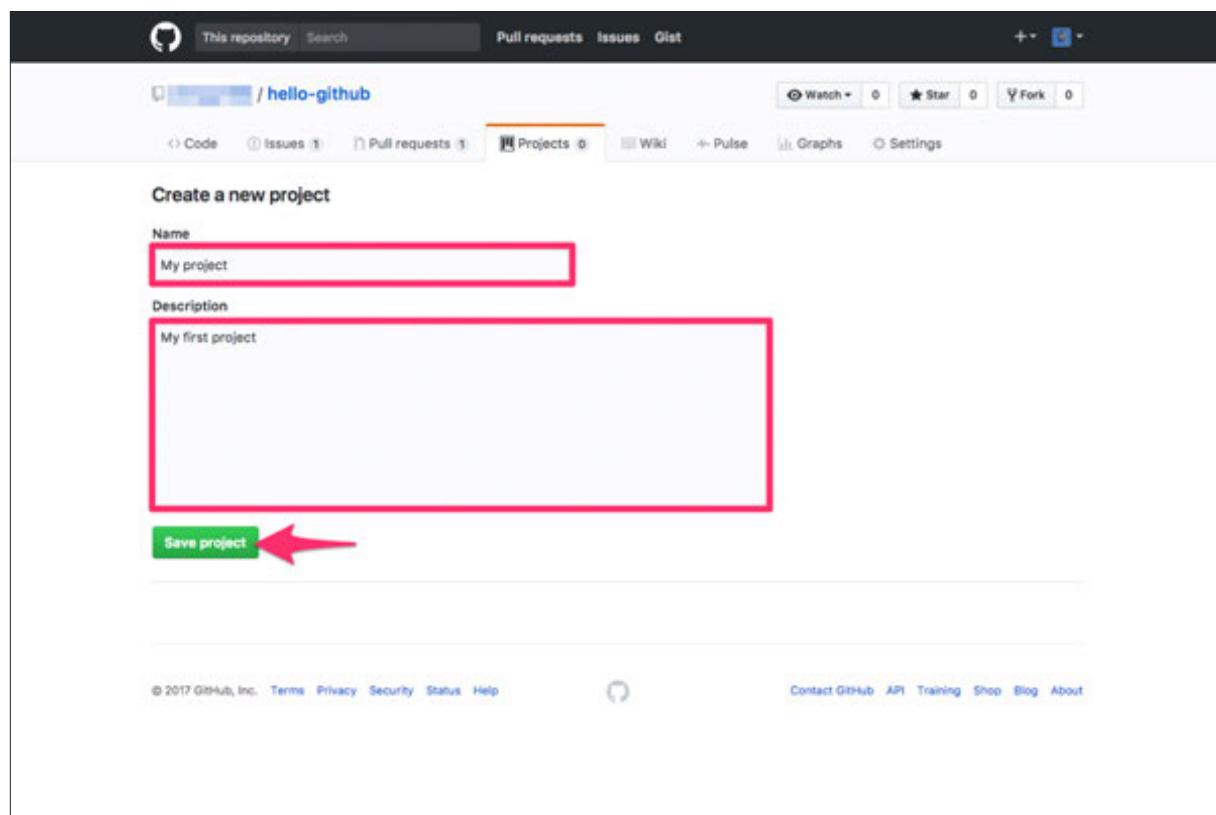


図10 プロジェクト作成ページ

プロジェクト作成が完了すると、今回作成したプロジェクトのページが表示されます。

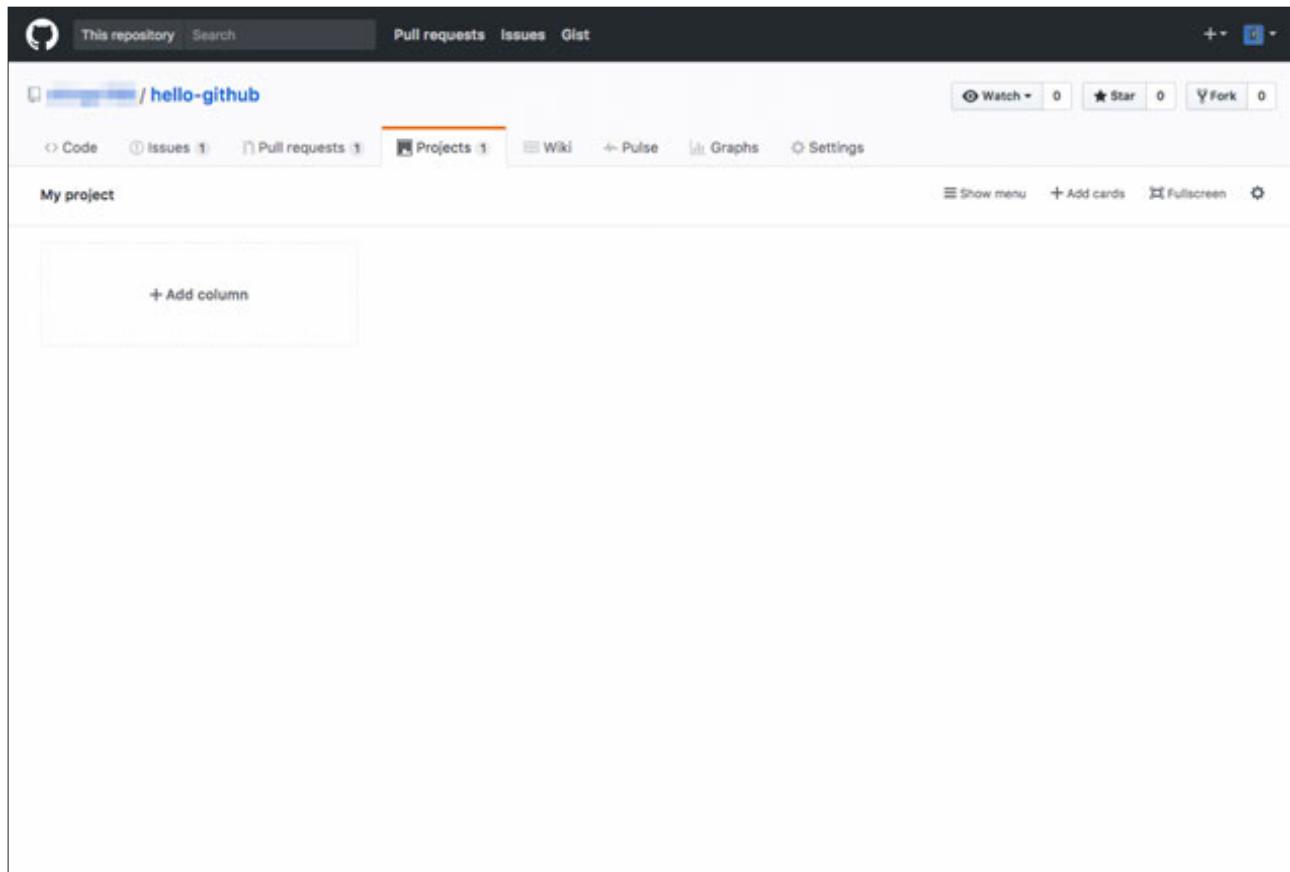


図 11 プロジェクトのページ

カラムを追加する

カラムの数や名称は、プロジェクトのワークフローに合わせて自由に設定できます。今回は、「ToDo」（実行予定のタスク）と「Done」（完了したタスク）の2つのカラムを作成してみます。

プロジェクト作成直後の場合、カラムは1つも存在しません。「Add column」をクリックしてカラムを作成しましょう。

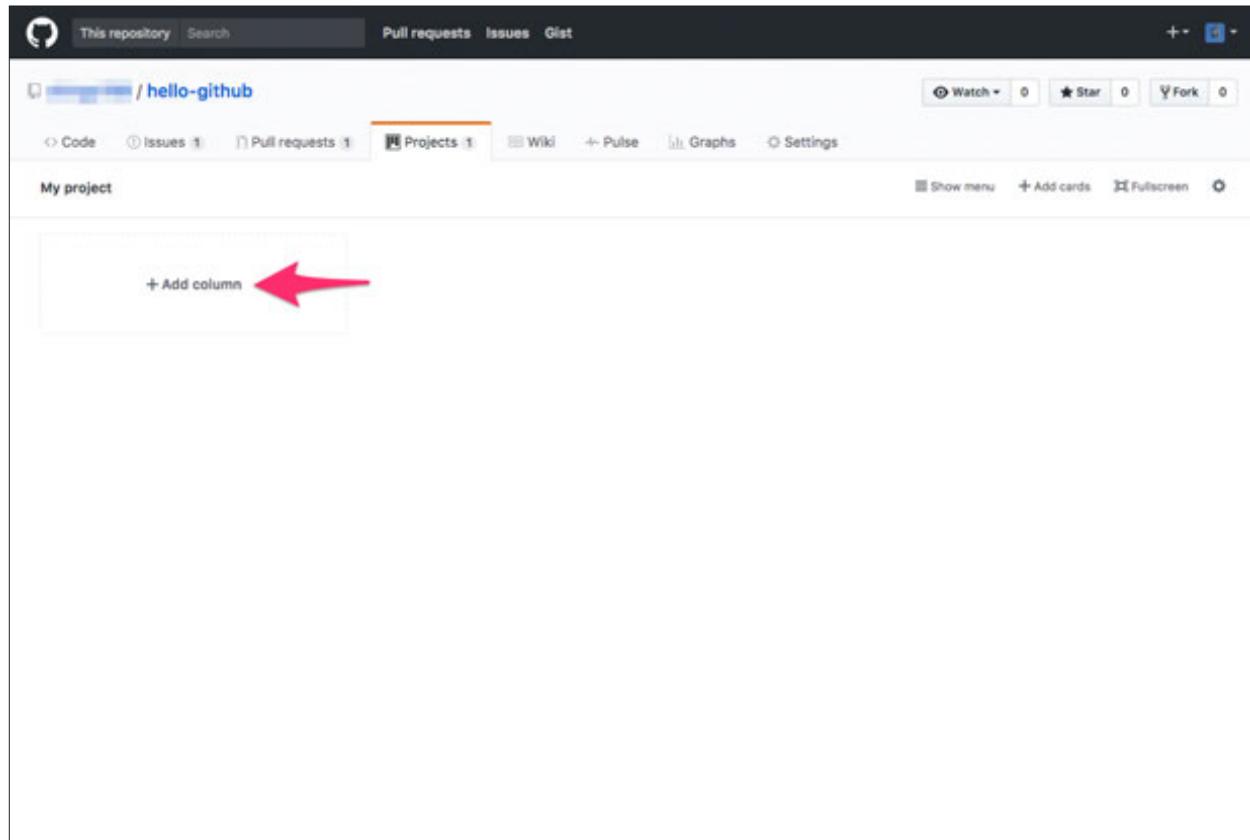


図 12 「Add column」をクリック

1 つ目のカラム名「ToDo」を入力し、「Create column」をクリックします。

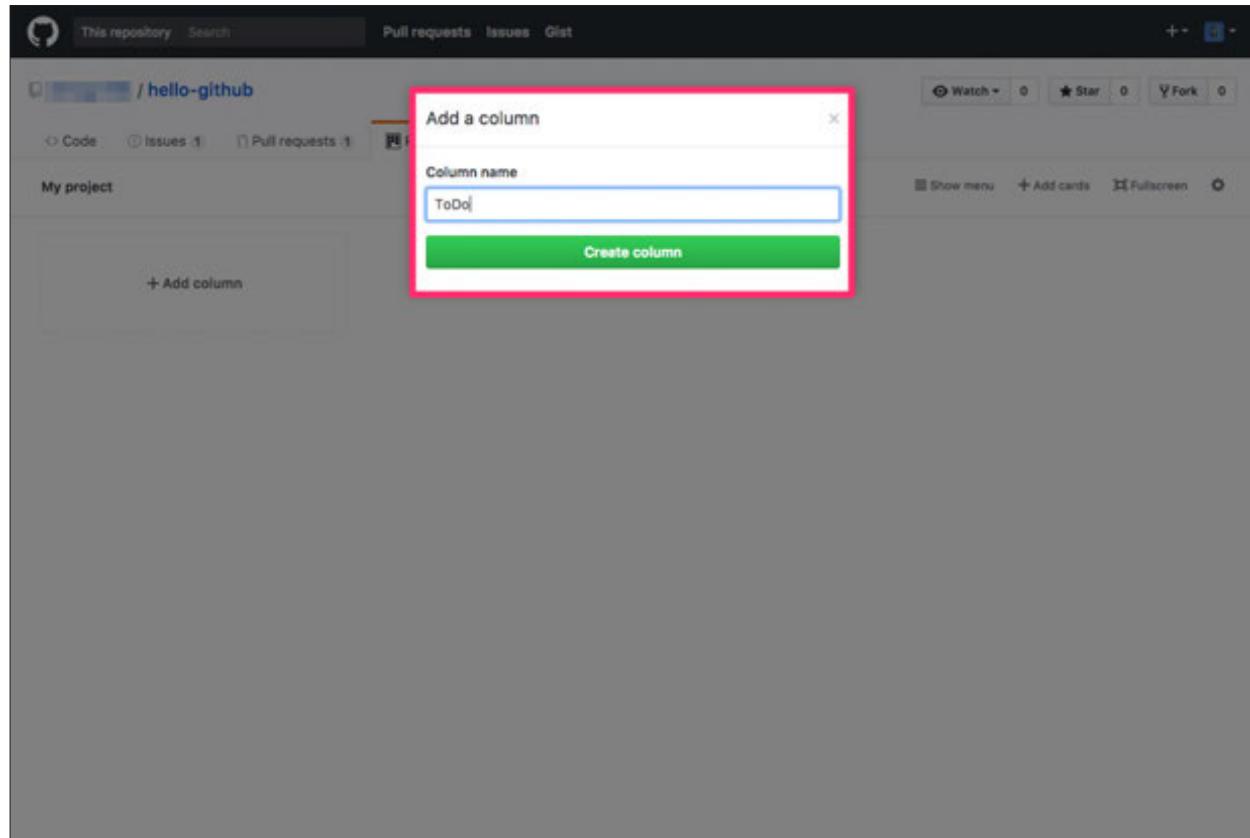


図 13 カラムを追加

カラム「ToDo」が追加されました。同様の操作でカラム「Done」を追加しましょう。

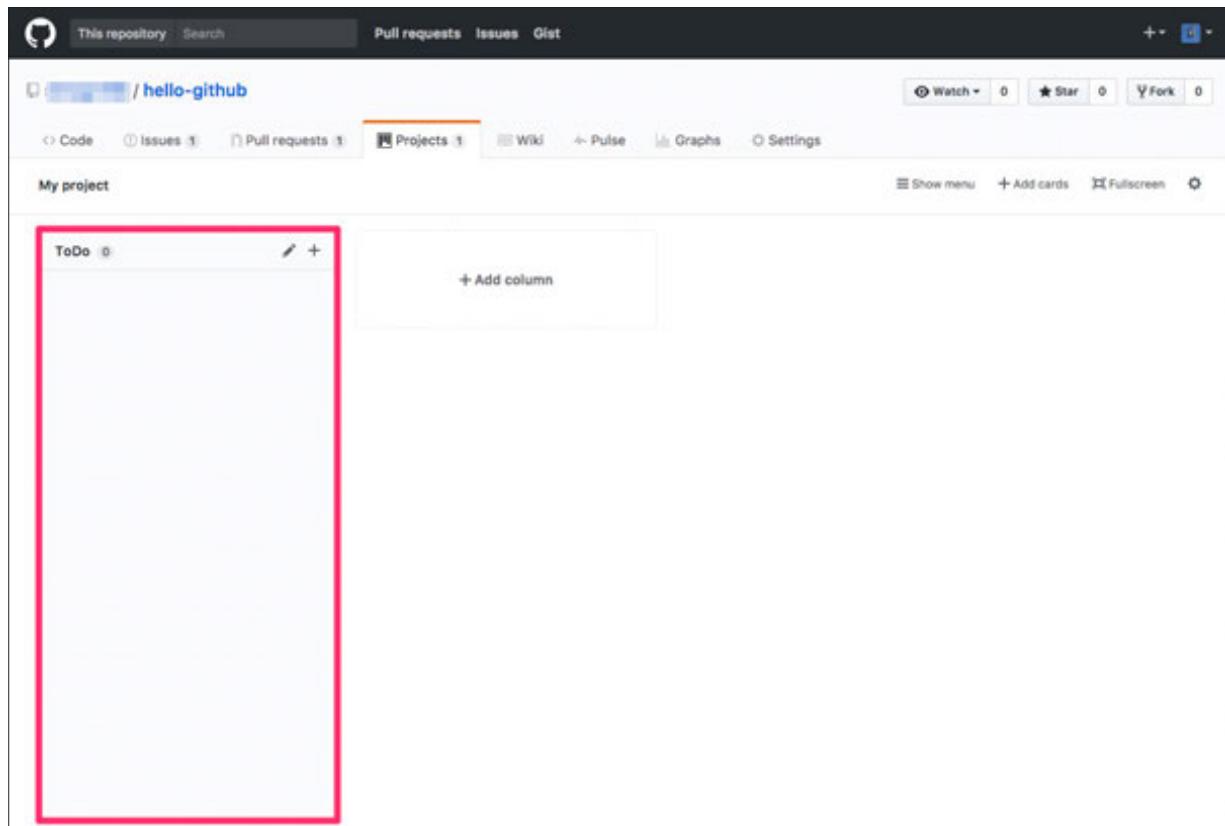


図 14 カラム「ToDo」追加後のプロジェクトのページ

カラム「Done」が追加されました。

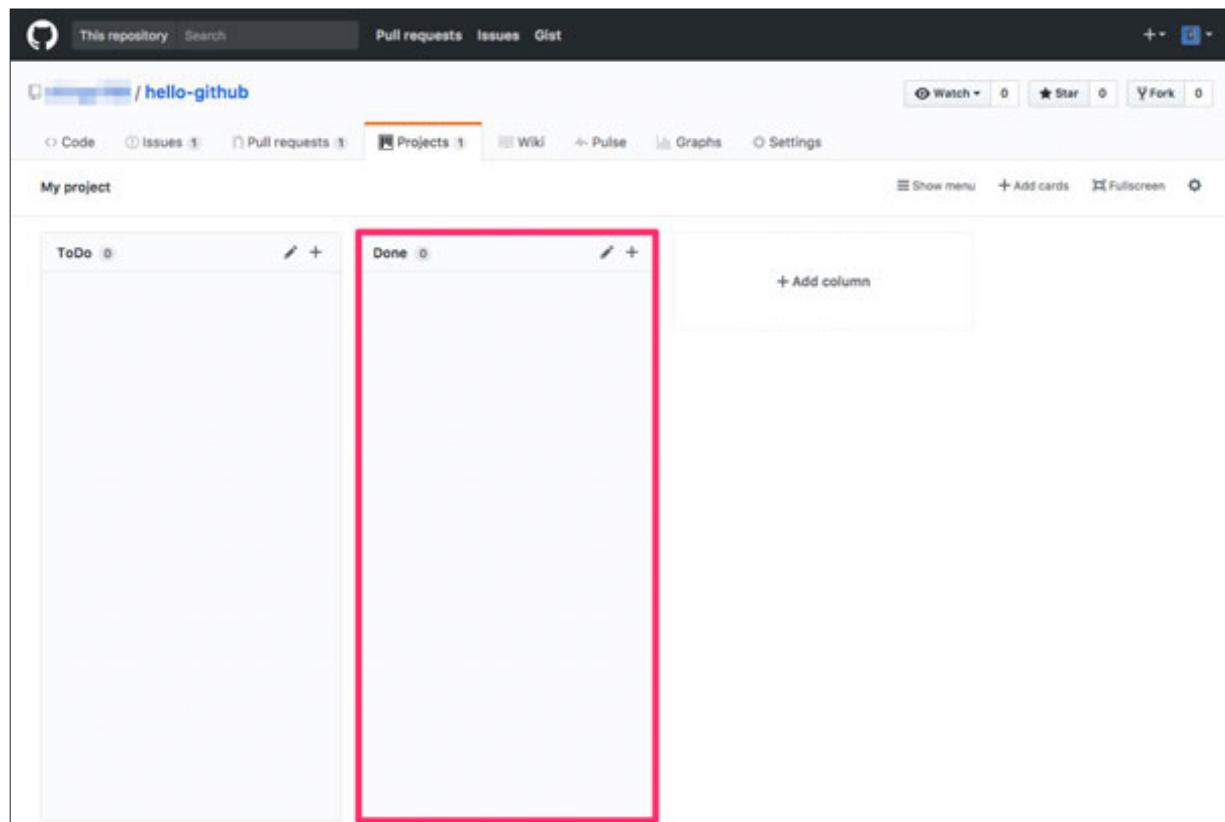


図 15 カラム「Done」追加後のプロジェクトのページ

カードを追加する

カラムの準備ができましたので「カード」を追加してみましょう。

カードを追加するにはカラムの右上の「+」ボタンをクリックします。ここではカラム「Todo」にカードを追加してみます。

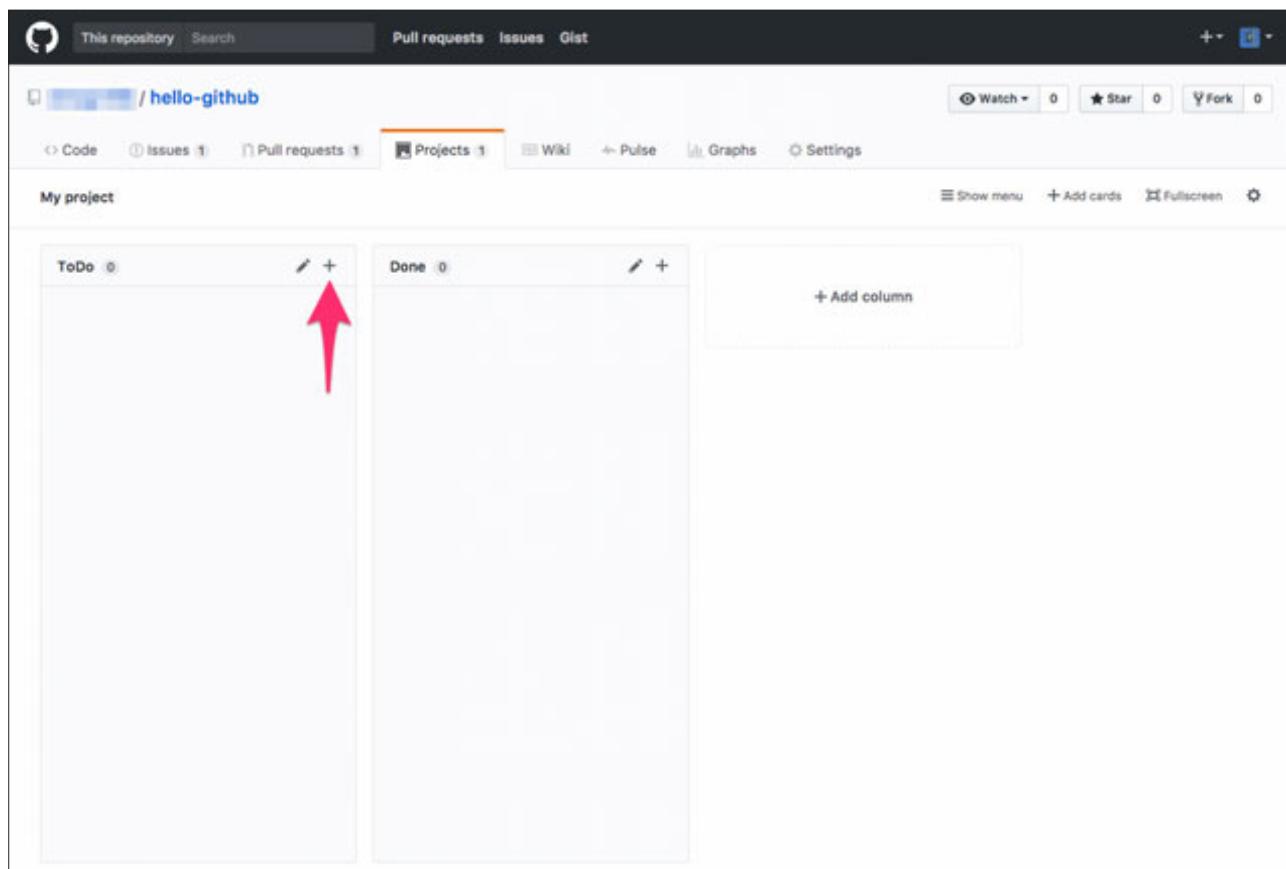


図 16 カラムの右上の「+」ボタンをクリック

カラムの上部にテキストボックスが表示されるので、タスクの内容を記入し、「Add note」をクリックします。

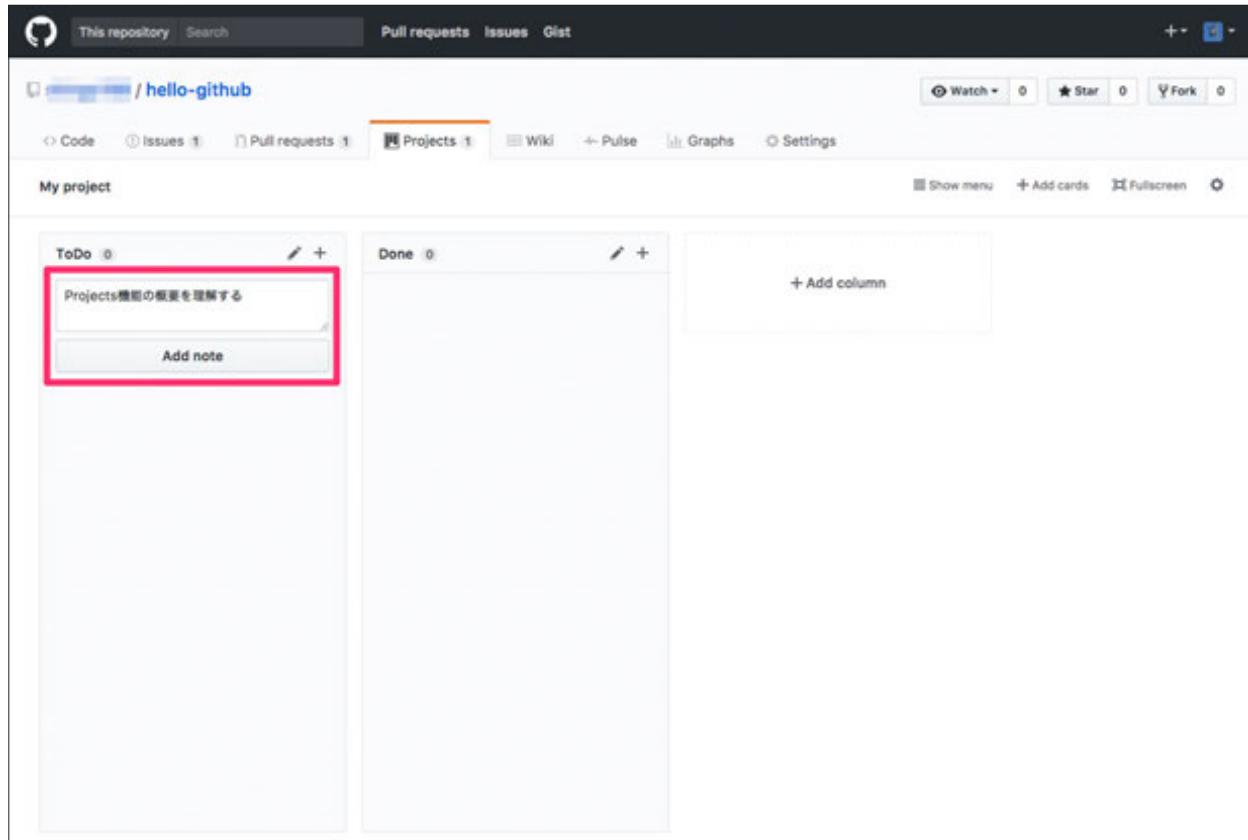


図 17 カード追加

カードが追加されました。

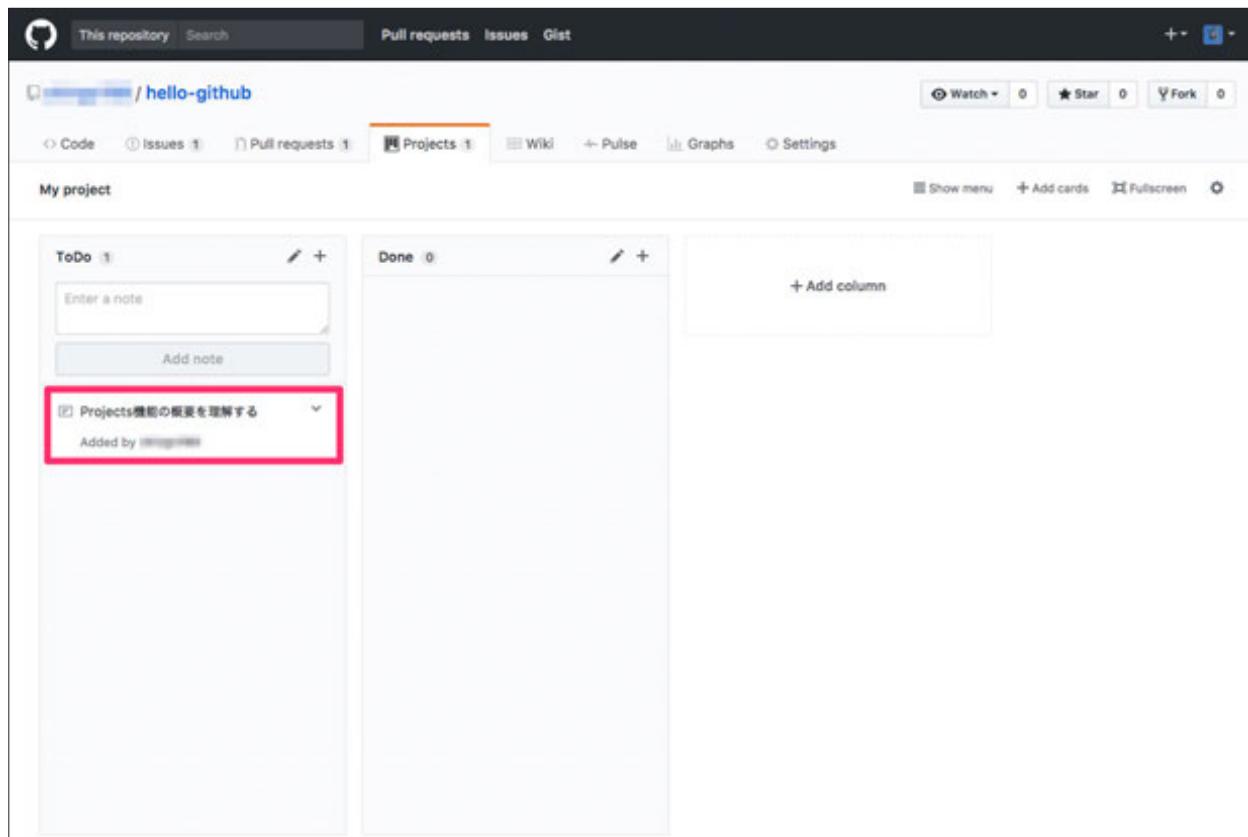


図 18 カード追加後のプロジェクトのページ

カードを移動する

カラムに追加したカードは別のカラムへ移動できます。

今回のようにカードのステータスごとにカラムを作成した場合、カードを移動することでカードのステータスを更新できます。

カードを移動するには、移動したいカードを移動先のカラムへドラッグ&ドロップします。

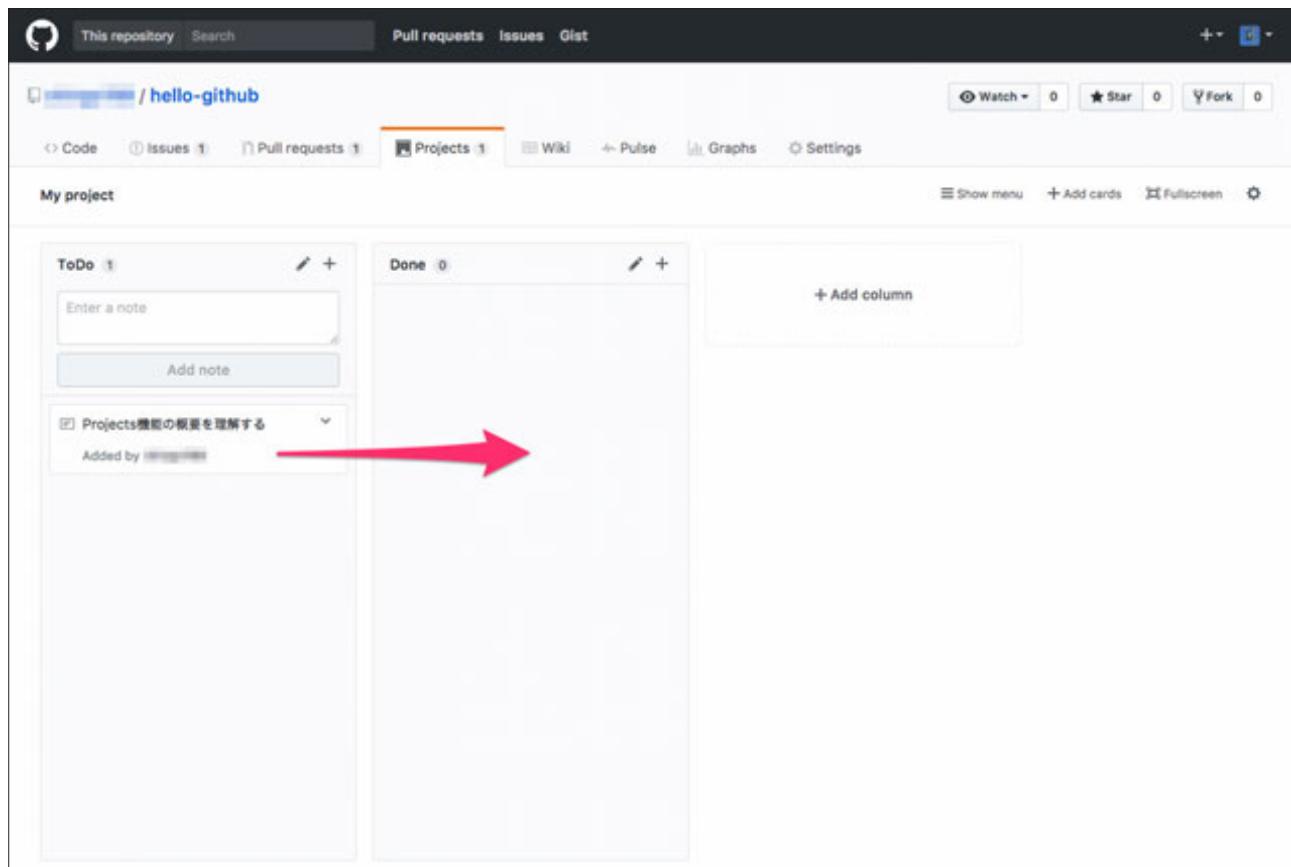


図 19 カードの移動

カードがカラム「Done」へ移動しました。

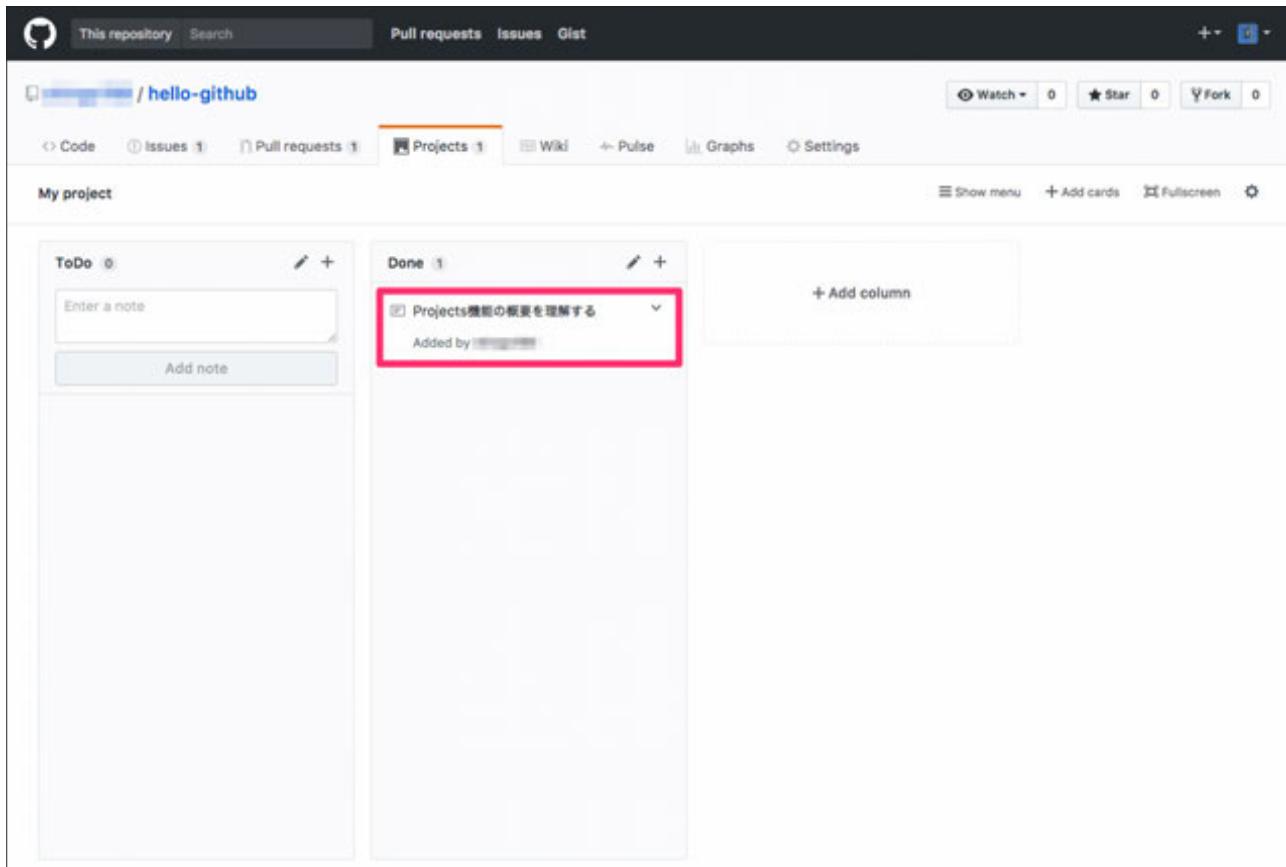


図 20 カード移動後のプロジェクトのページ

1 つのカラム内にカードが複数ある場合、カラム内でドラッグ&ドロップすると、カードの順序を入れ替えることができます。

イシュー（プルリクエスト）を追加する

イシュー（プルリクエスト）をカラムに追加することもできます。追加するには「Add cards」をクリックします。

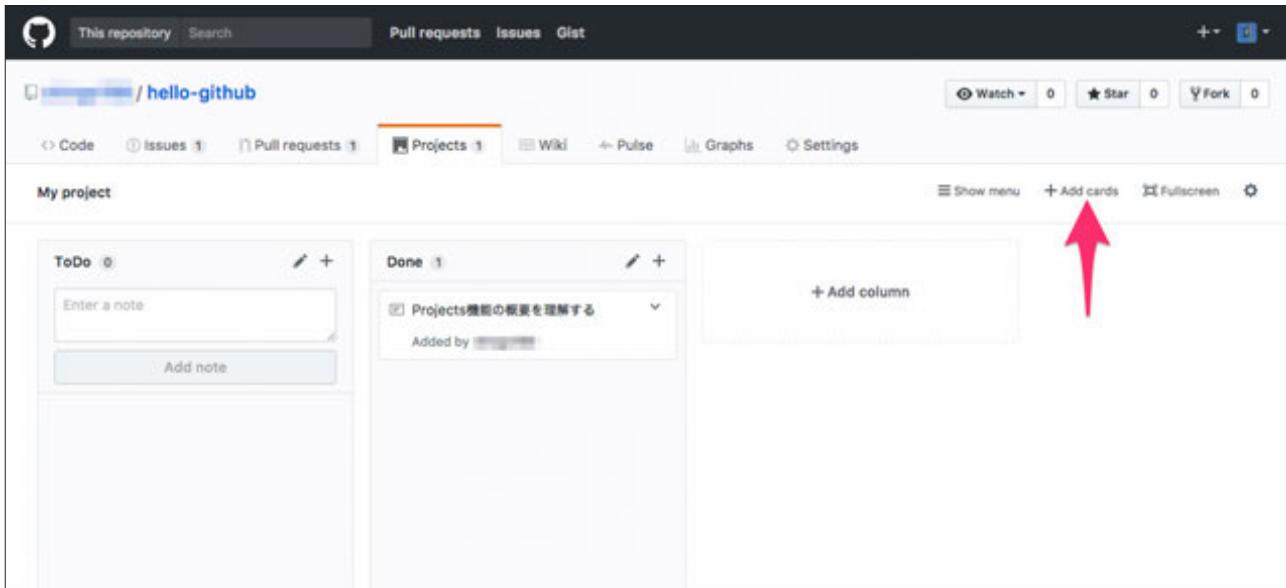


図 21 「Add cards」をクリック

イシュー（プルリクエスト）の一覧が右側に表示されるので、追加したいイシュー（プルリクエスト）を追加先のカラムへドラッグ&ドロップします。

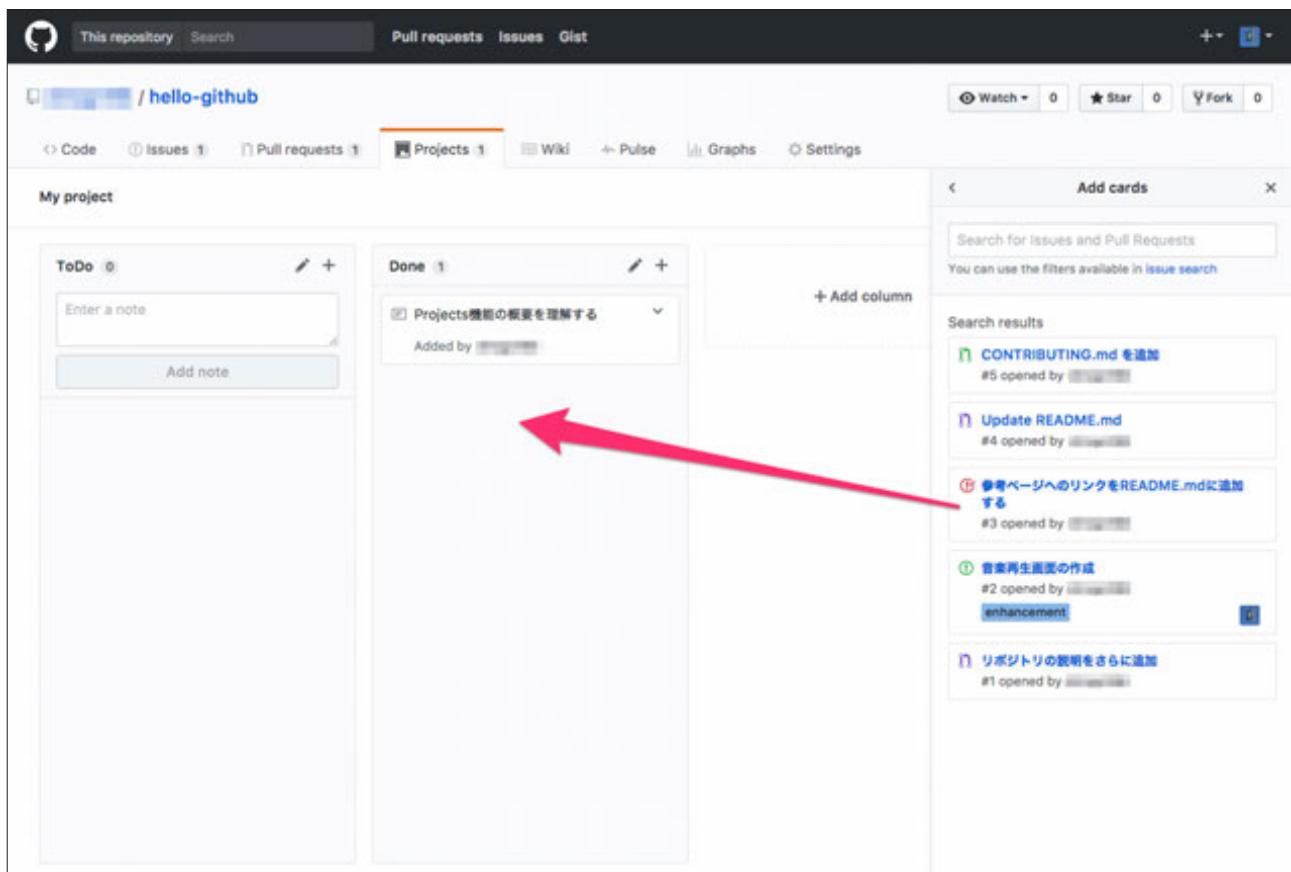


図 22 イシュー（プルリクエスト）を追加

イシュー（プルリクエスト）がカラム「Done」に追加されました。追加したイシューとプルリクエストは、通常のカードと同様に、カラムの移動などの操作ができます。

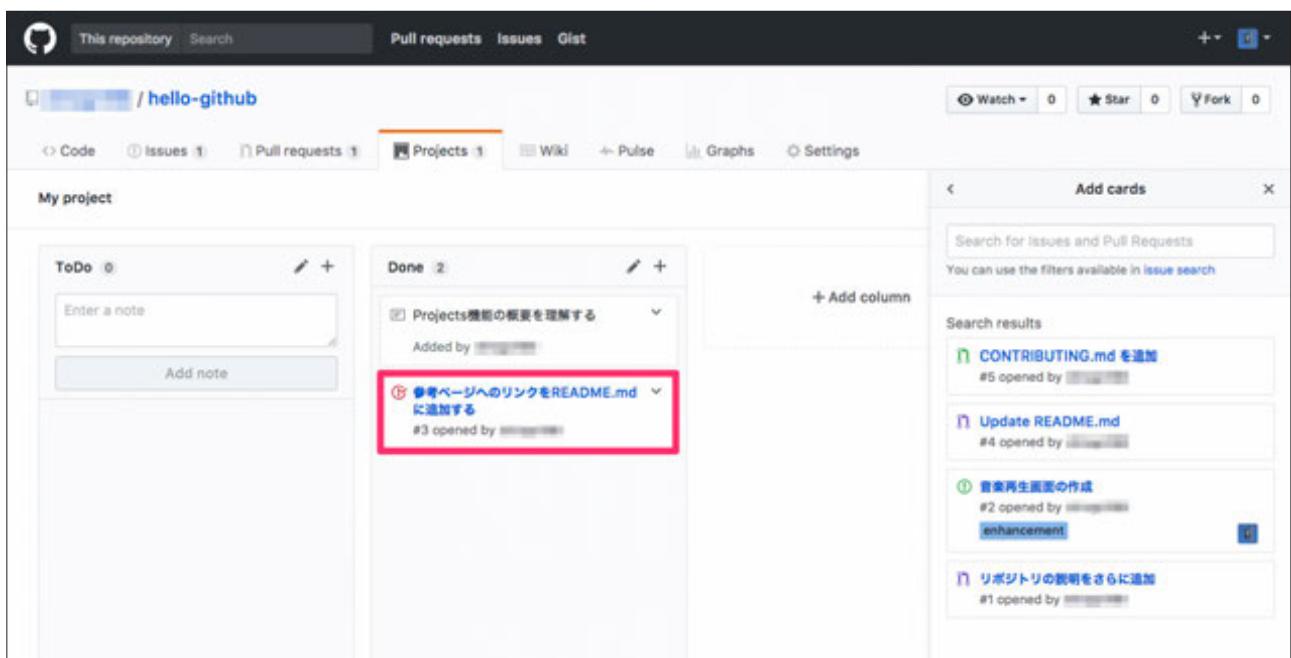


図 23 イシュー（プルリクエスト）追加後のプロジェクトのページ

ラベルについて

イシュー（プルリクエスト）に「ラベル」（Labels）を設定する方法については、前回の記事で解説しました。

デフォルトで用意されているラベルは 7 種類です。これ以外のラベルが必要であれば、新たに追加ができます。また、不要なラベルがあれば削除もできます。

ラベル管理ページを表示する

ラベルの追加、編集、削除は、ラベル管理ページから実行できます。

イシュー（プルリクエスト）の一覧ページを表示し、「Labels」をクリックします。

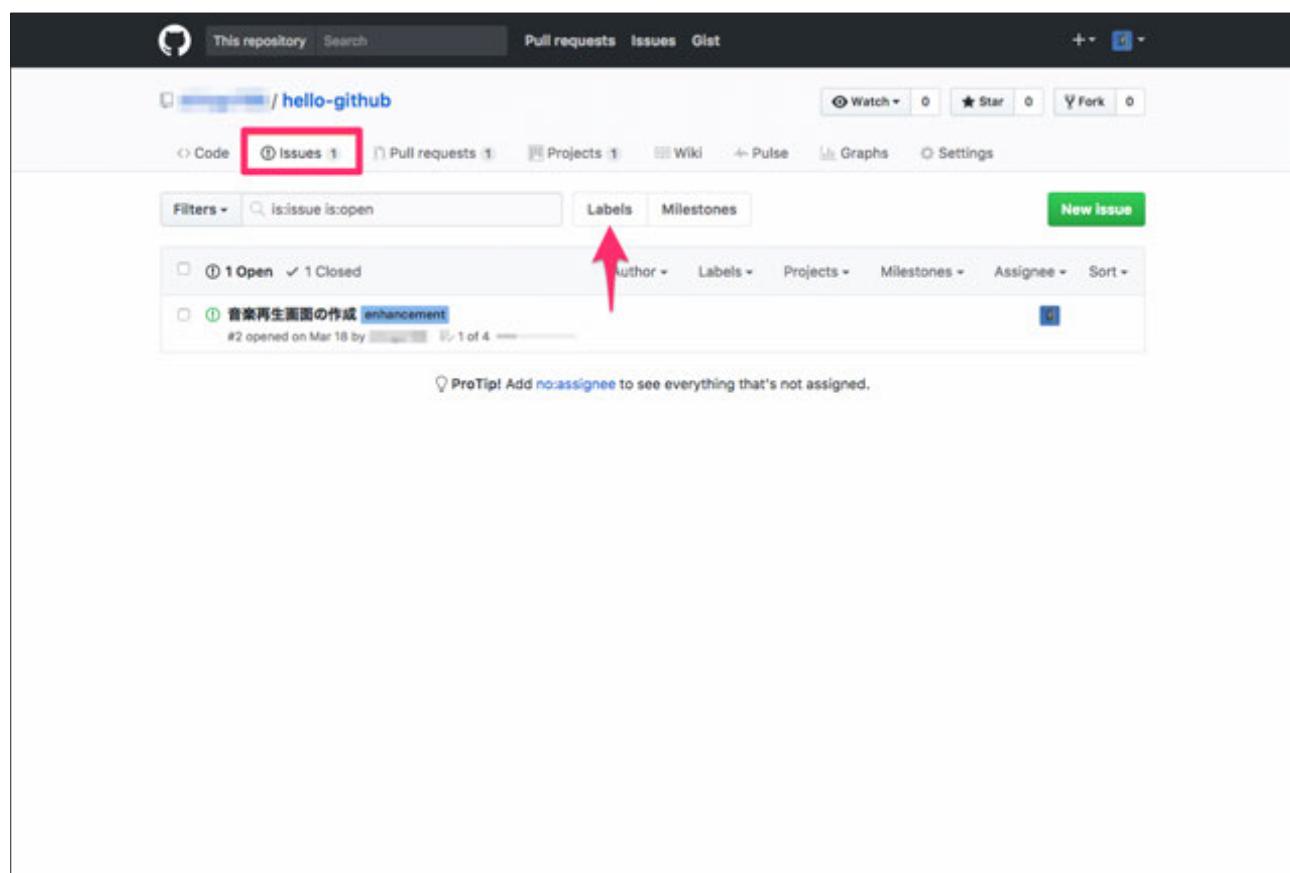


図 24 イシューの一覧ページ

ラベル管理ページを表示できました。

The screenshot shows the 'Labels' section of a GitHub repository named 'hello-github'. It displays a list of seven labels: 'bug', 'duplicate', 'enhancement', 'help wanted', 'invalid', 'question', and 'wontfix'. Each label entry includes a count of '0 open issues', an 'Edit' button, and a 'Delete' button.

Label	Open Issues	Action
bug	0	Edit Delete
duplicate	0	Edit Delete
enhancement	1	Edit Delete
help wanted	0	Edit Delete
invalid	0	Edit Delete
question	0	Edit Delete
wontfix	0	Edit Delete

図 25 ラベル管理ページ

ラベルを追加する

ラベルを追加するには「New label」をクリックします。

The screenshot shows the same 'Labels' section as the previous image, but with a red arrow pointing to the 'New label' button located in the top right corner of the interface.

図 26 「New label」をクリック

テキストボックスやボタンなどが表示されます。

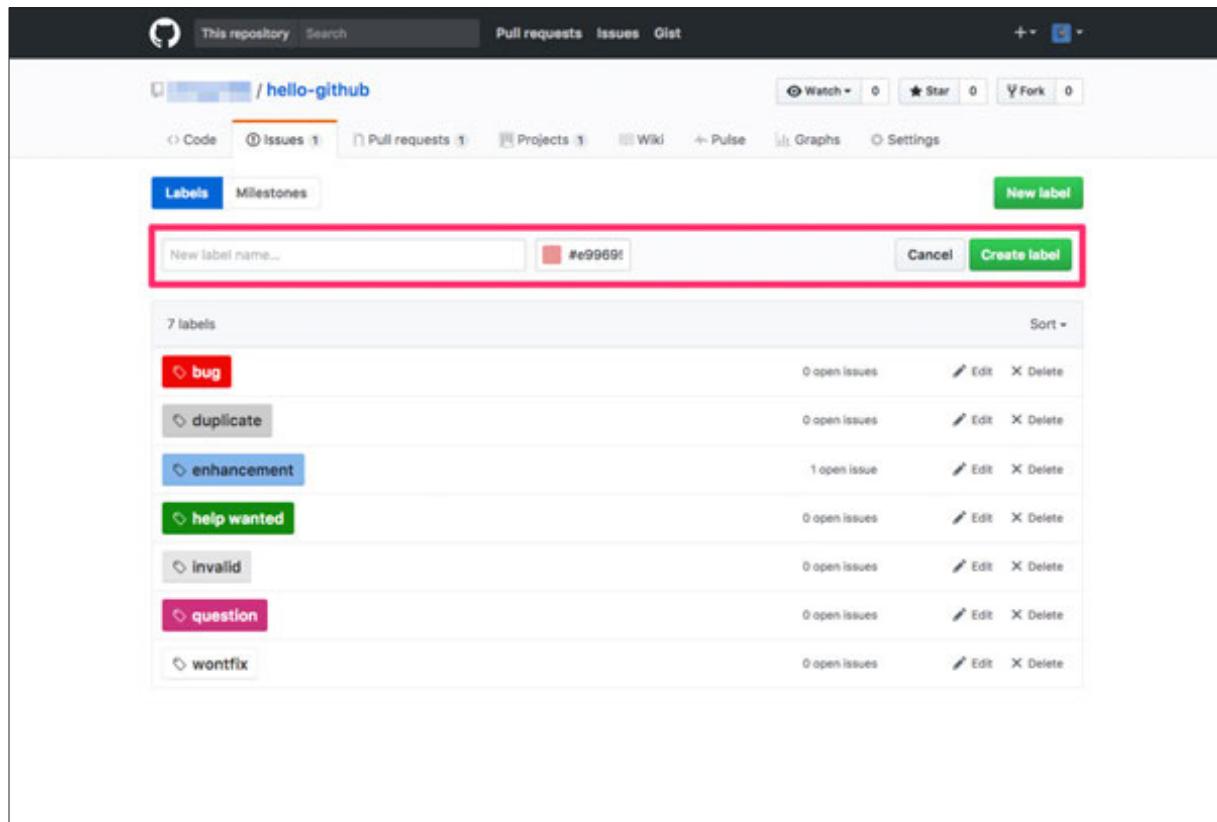


図 27 ラベル追加工アリ

テキストボックスにラベル名を入力します。ここでは「イシュー周りの機能調査」と入力しました。

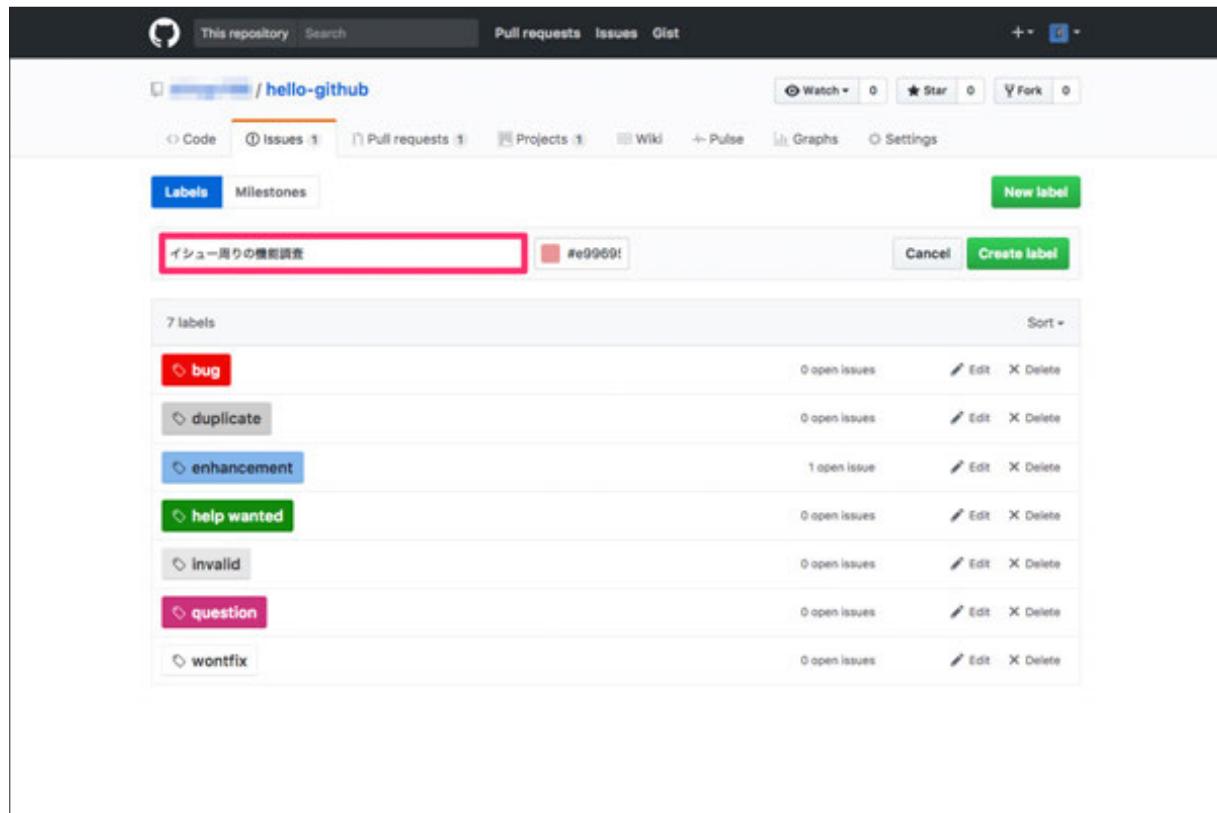


図 28 ラベル名入力

テキストボックスの右のコンポーネントをクリックすると、ラベルの色を指定できます。

色の初期値はランダムに選択された色です。色を変更するには、16進数のカラーコードを入力するか、コンポーネントの下に表示されるメニュー内の任意の色を選択します。

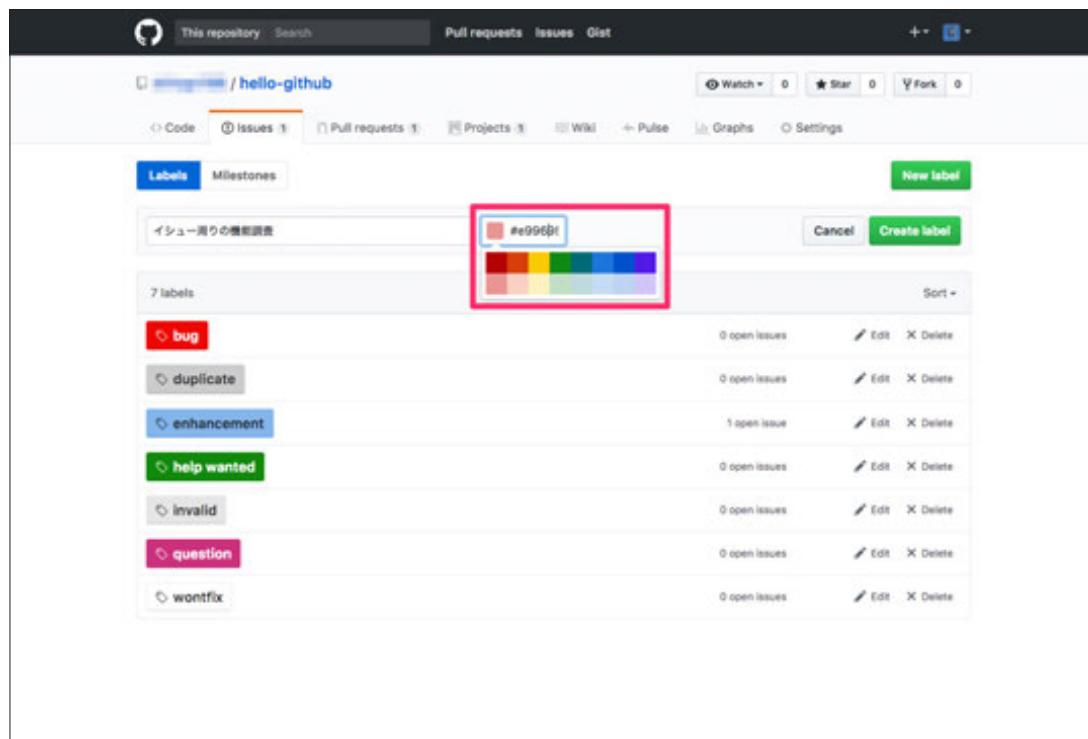


図 29 ラベルの色の設定

ここでは、メニューの中から青を選択しました。

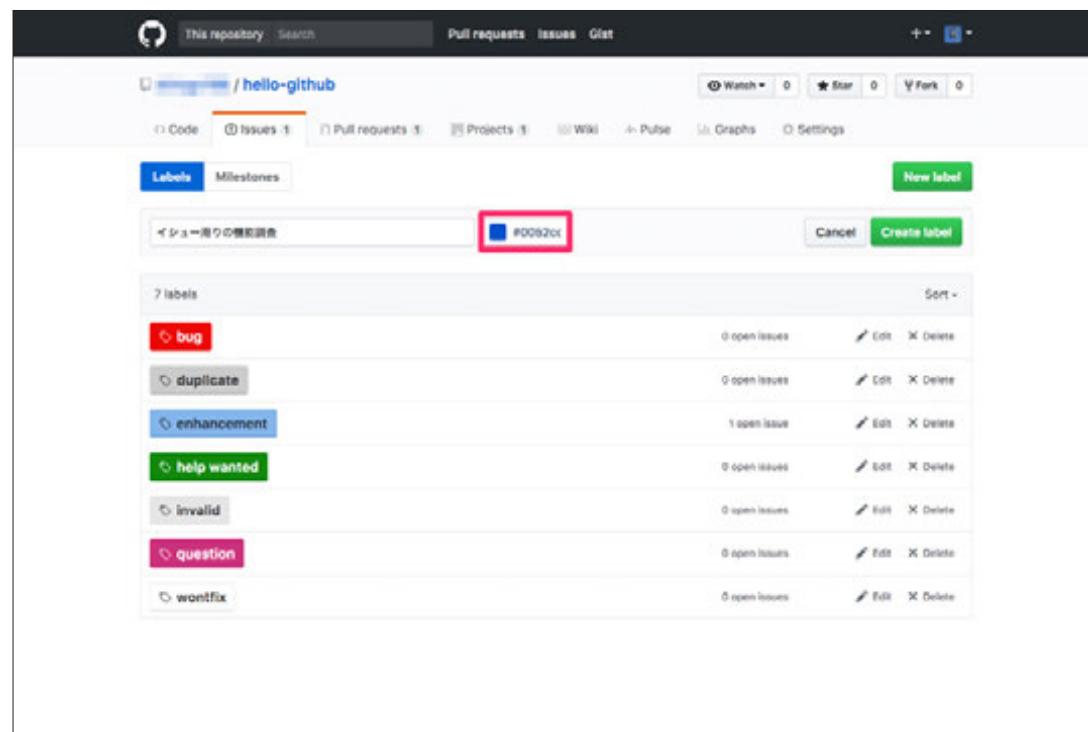


図 30 ラベル名と色を設定した後のラベル追加工アリア

「Create label」をクリックすると、ラベルの追加を実行できます。

The screenshot shows the GitHub interface for managing labels in a repository named 'hello-github'. At the top, there are tabs for 'Code', 'Issues 1', 'Pull requests', 'Projects', 'Wiki', 'Pulse', 'Graphs', and 'Settings'. Below these, there are two tabs: 'Labels' (which is selected) and 'Milestones'. A search bar contains the text '#issue周りの機能調査'. A green button labeled 'New label' is visible. A red arrow points to a green button labeled 'Create label' located at the bottom right of the label list. The label list itself shows seven existing labels: 'bug', 'duplicate', 'enhancement', 'help wanted' (highlighted in green), 'invalid', 'question', and 'wontfix'. Each label entry includes a color swatch, the label name, the count of open issues (0 or 1), and edit/delete buttons.

図 31 ラベル追加を実行

ラベルを追加できました。

This screenshot shows the same GitHub interface as the previous one, but now with a new label added. The 'Labels' tab is still selected. The label list now shows eight entries: 'bug', 'duplicate', 'enhancement', 'help wanted' (still highlighted in green), 'invalid', 'issue周りの機能調査' (the newly added label, highlighted with a red box), 'question', and 'wontfix'. The 'New label name...' input field is empty, and the 'Create label' button is visible again. The rest of the interface remains the same, with the repository name 'hello-github' and various navigation tabs at the top.

図 32 ラベル追加後のラベル管理ページ

ラベルを編集する

ラベルの設定を編集するには、対象ラベルの「Edit」をクリックします。

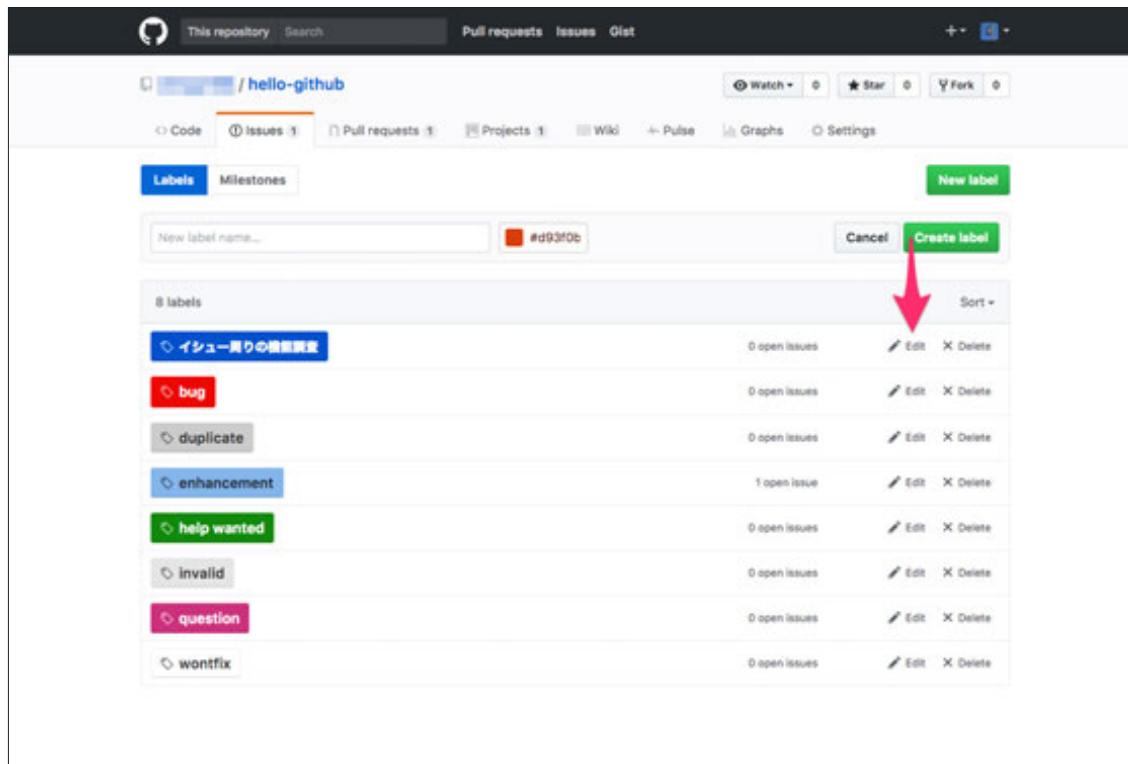


図 33 「Edit」をクリック

テキストボックスやボタンなどが表示されるので、ラベルを新規に追加したときと同じ要領で、ラベル名の入力や色の選択を行い「Save change」をクリックします。

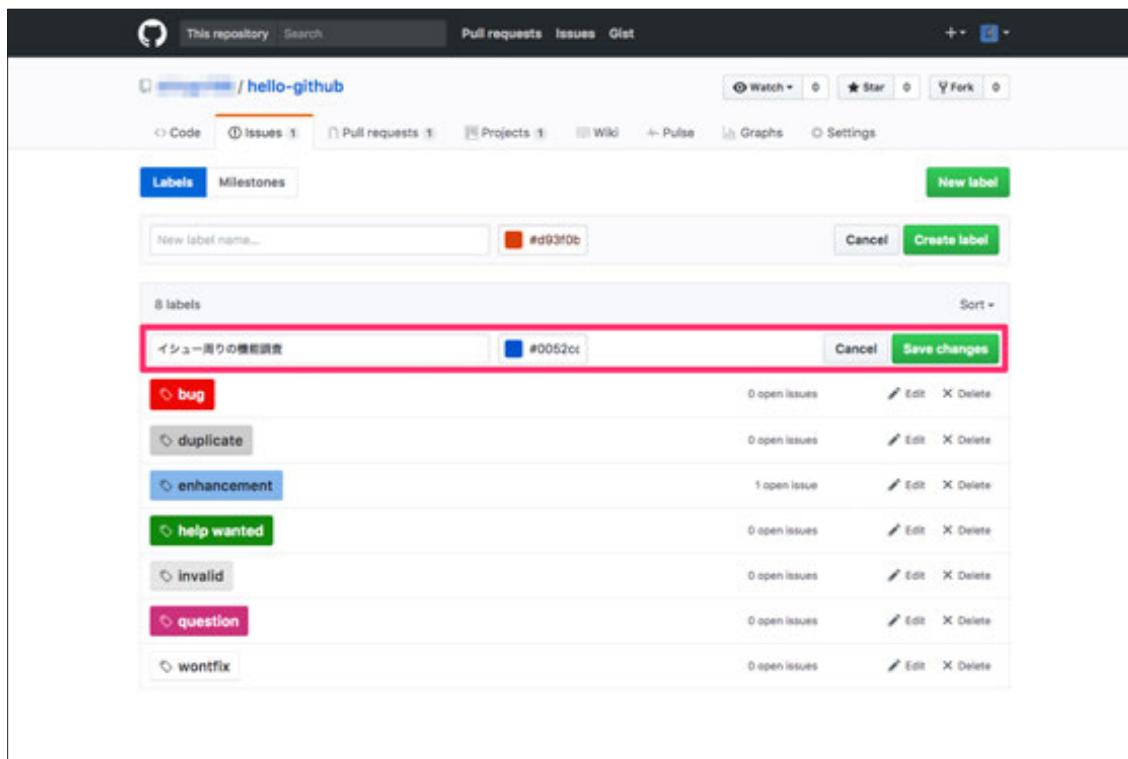


図 34 ラベル編集モード

ラベルを削除する

ラベルを削除するには、対象ラベルの「Delete」をクリックします。ここではラベル「イシュー周りの機能調査」を削除してみます。

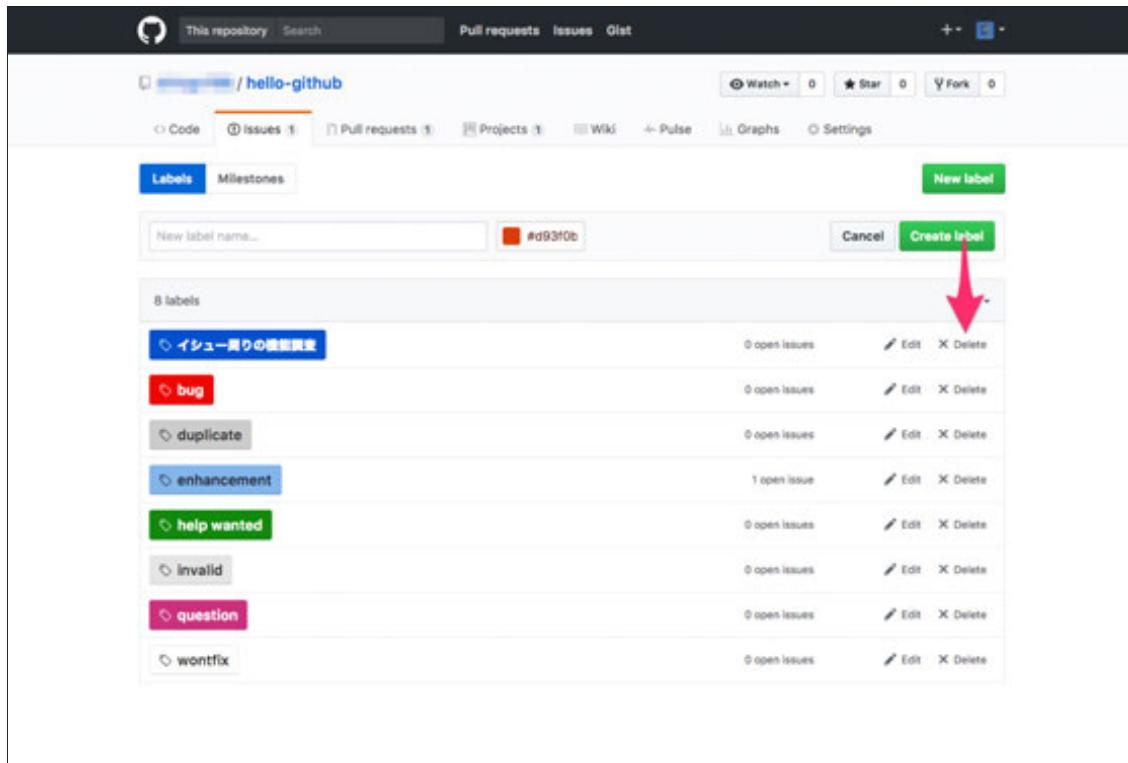


図 35 「Delete」をクリック

確認メッセージが表示されます。「Delete label」をクリックすると、ラベルの削除を実行できます。

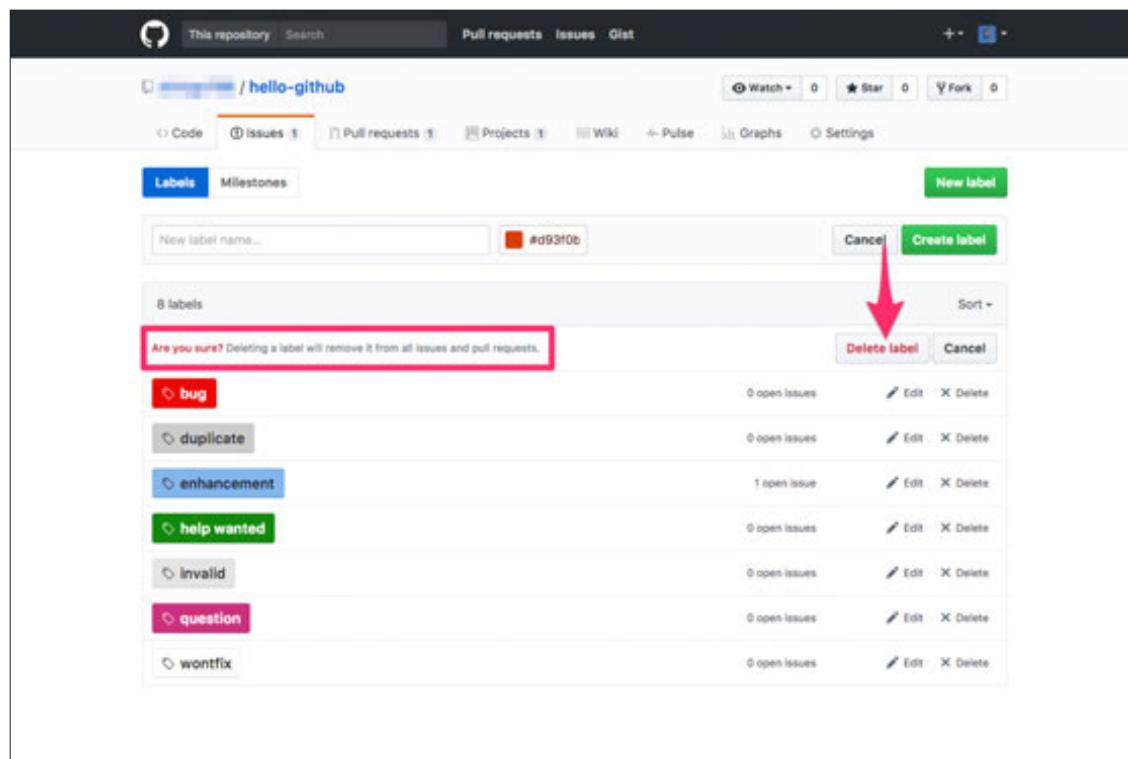


図 36 ラベル削除の確認メッセージ

ラベル「イシュー周りの機能調査」が削除されました。

Label	Open Issues	Action
bug	0	Edit Delete
duplicate	0	Edit Delete
enhancement	1	Edit Delete
help wanted	0	Edit Delete
invalid	0	Edit Delete
question	0	Edit Delete
wontfix	0	Edit Delete

図 37 ラベル削除後のラベル管理ページ

マイルストーンについて

「マイルストーン」(Milestones) は複数のイシュー（プルリクエスト）の進捗（しんちょく）を追跡するための機能です。

例えば、以下のように「バージョン 1.0 リリース」というプロジェクトの節目に対してマイルストーンを作成します。

- マイルストーン「バージョン 1.0 リリース」
 - ・ 期限
 - ・ 2017 年 6 月 1 日
 - ・ 関連イシュー
 - ・ # 10
 - ・ # 12
 - ・ 関連プルリクエスト
 - ・ # 9
 - ・ # 11
 - ・ # 13

マイルストーン名と期限を設定したマイルストーンを作成し、関連するイシュー（プルリクエスト）にマイルストーンを割り当てることによって、特定の期限までに完了する必要があるタスクの進捗状況を把握したり、スケジュールを再検討したりできるようになります。

マイルストーン管理ページを表示する

マイルストーンの追加、編集、削除は、マイルストーン管理ページから実行できます。

イシュー（プルリクエスト）の一覧ページを表示し、「Milestones」をクリックします。

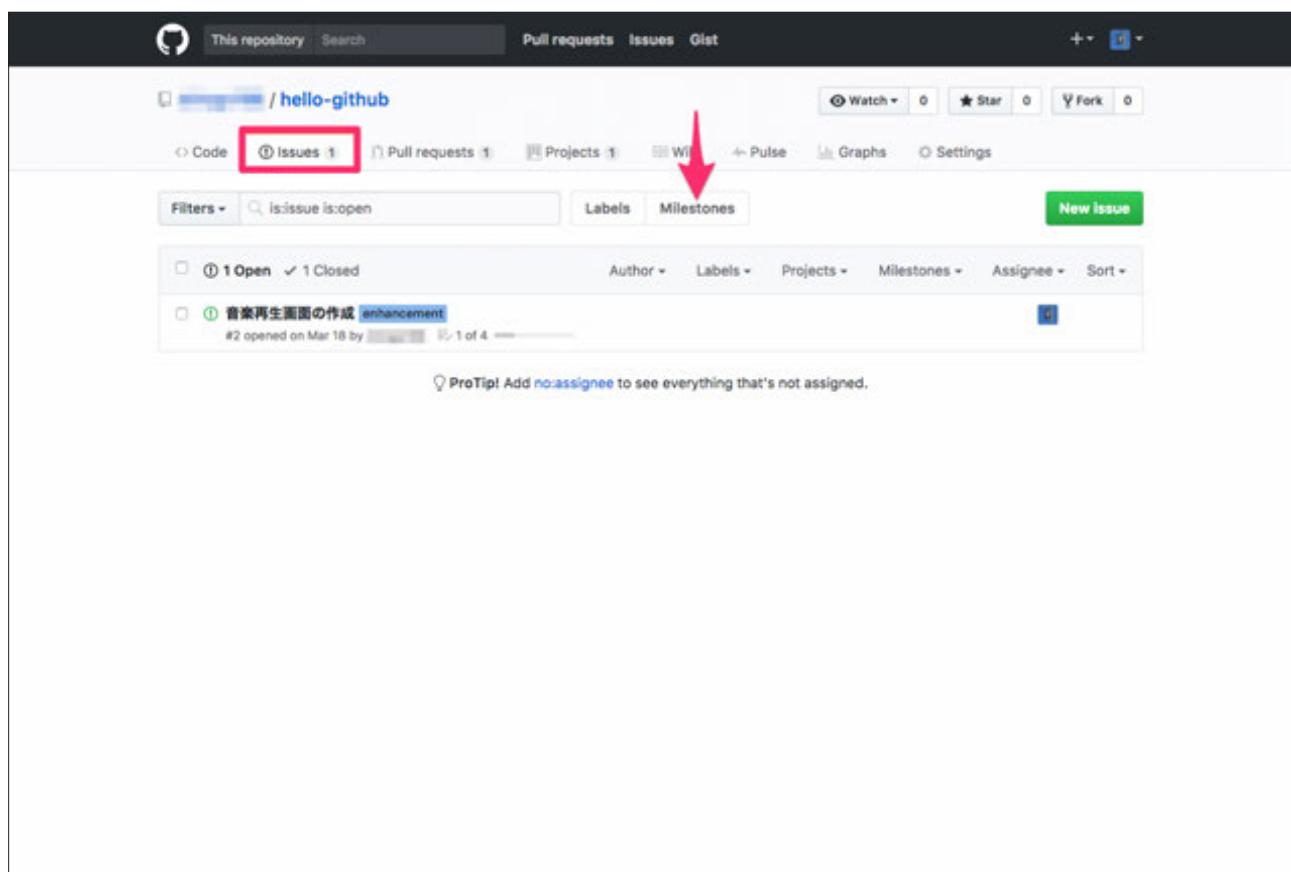


図 38 イシューの一覧ページ

マイルストーン管理ページを表示できました。マイルストーンを1つも作成していない場合、以下のような表示になります。

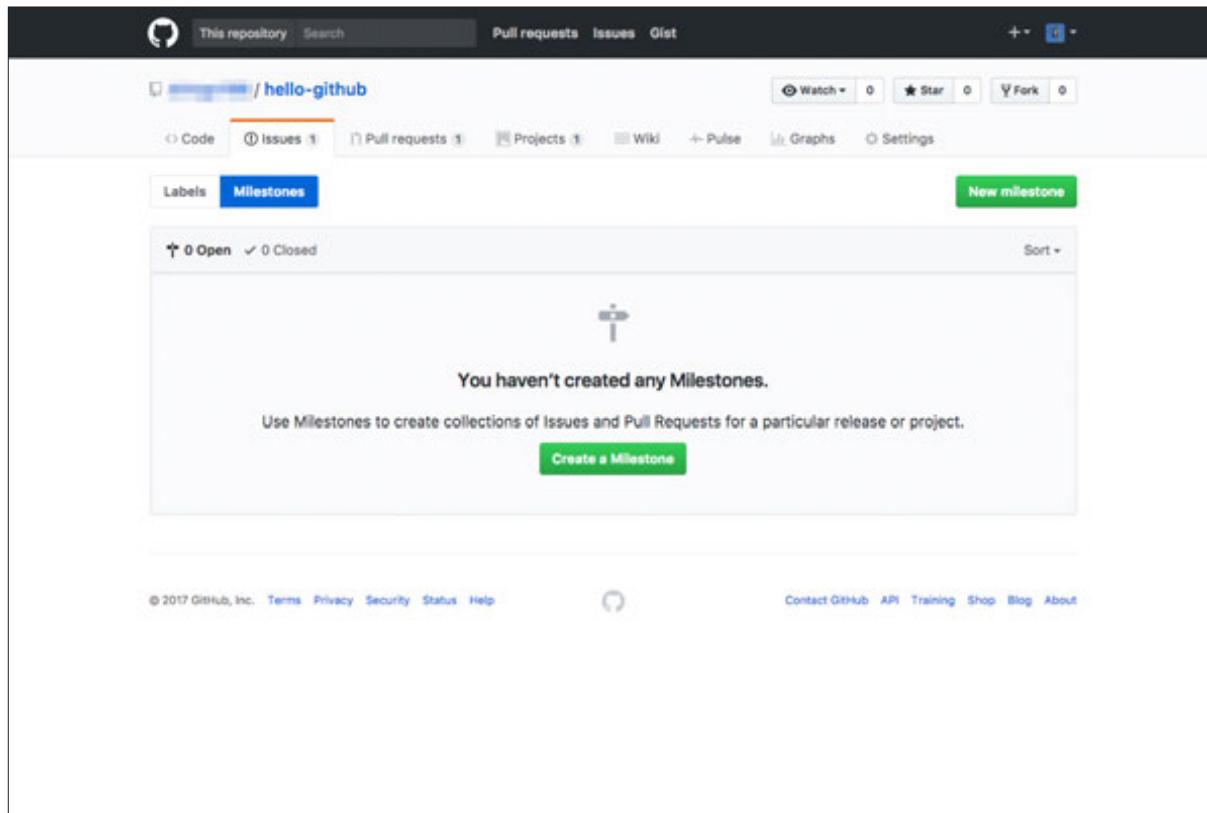


図 39 マイルストーン管理ページ

マイルストーンを作成する

マイルストーンを追加するには「New milestone」をクリックします。

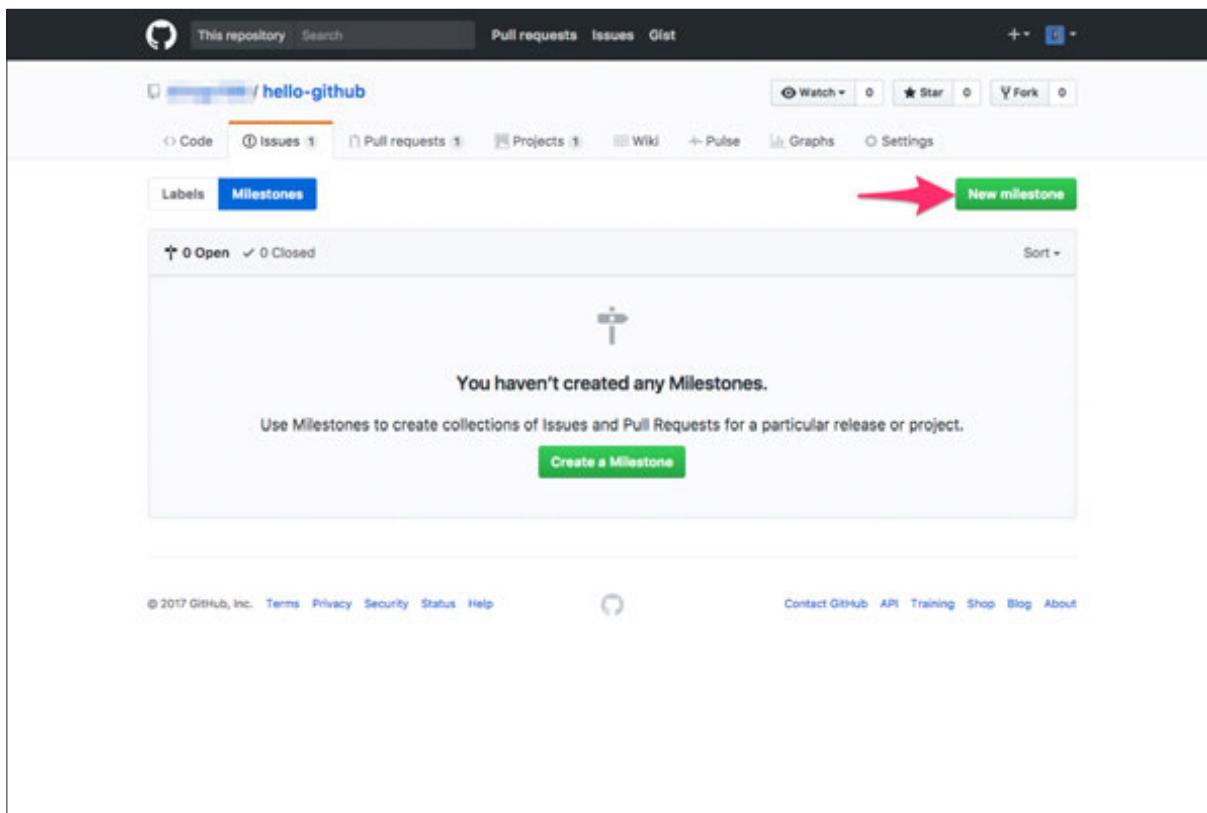


図 40 「New milestone」をクリック

マイルストーン作成ページが表示されます。「Title」にマイルストーンの名前を、「Description」にマイルストーンの説明を記入します。期限は右側のカレンダーを使用して選択できます。

ここでは、名前を「バージョン 1.0 リリース」、期限を「2017 年 6 月 1 日」に設定します。

「Create milestone」をクリックすると、マイルストーンの作成できます。

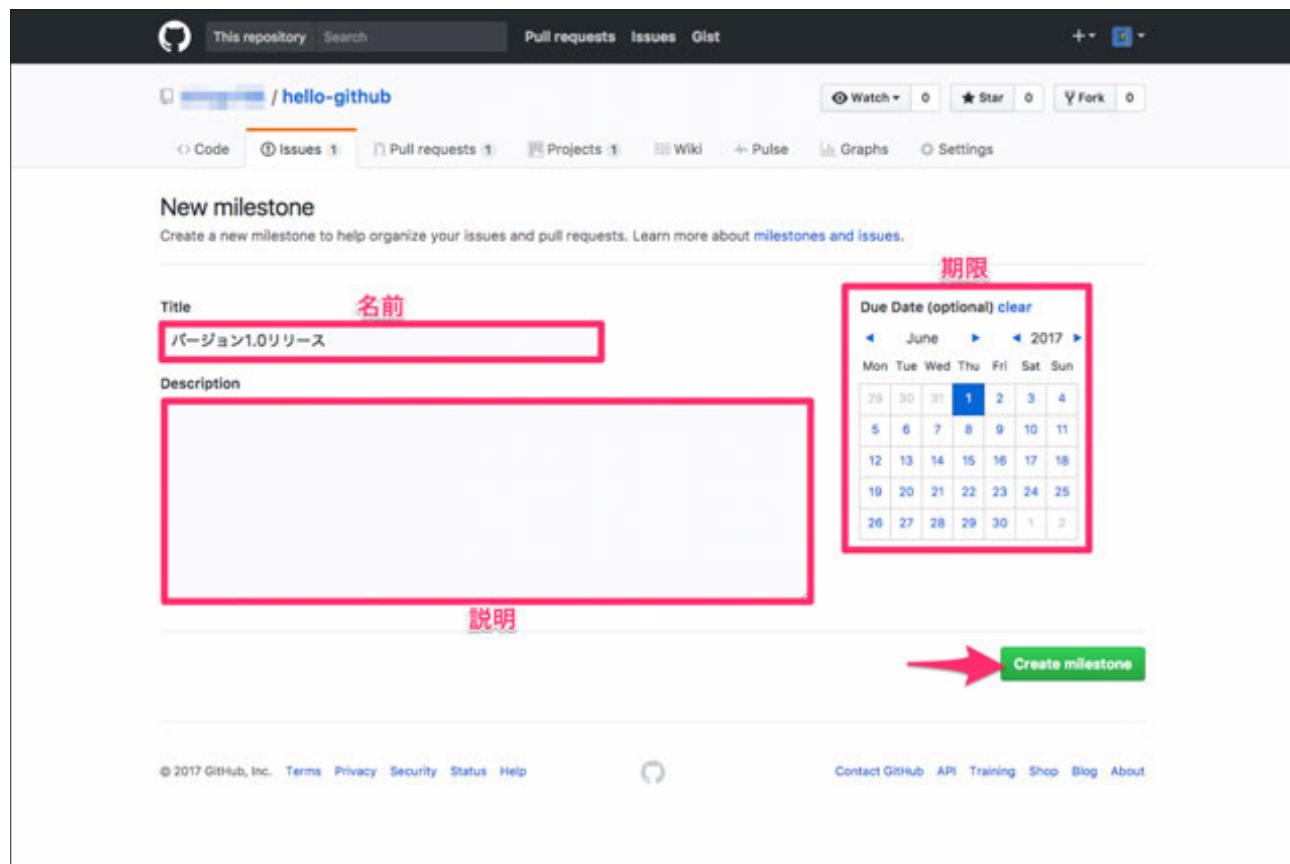


図 41 マイルストーン作成ページ

マイルストーン作成すると、マイルストーン管理ページが表示されます。

マイルストーン一覧の中に、さきほど作成した「バージョン 1.0 リリース」を確認できます。

現在このマイルストーンを割り当てているイシュー（プルリクエスト）は 0 件です。

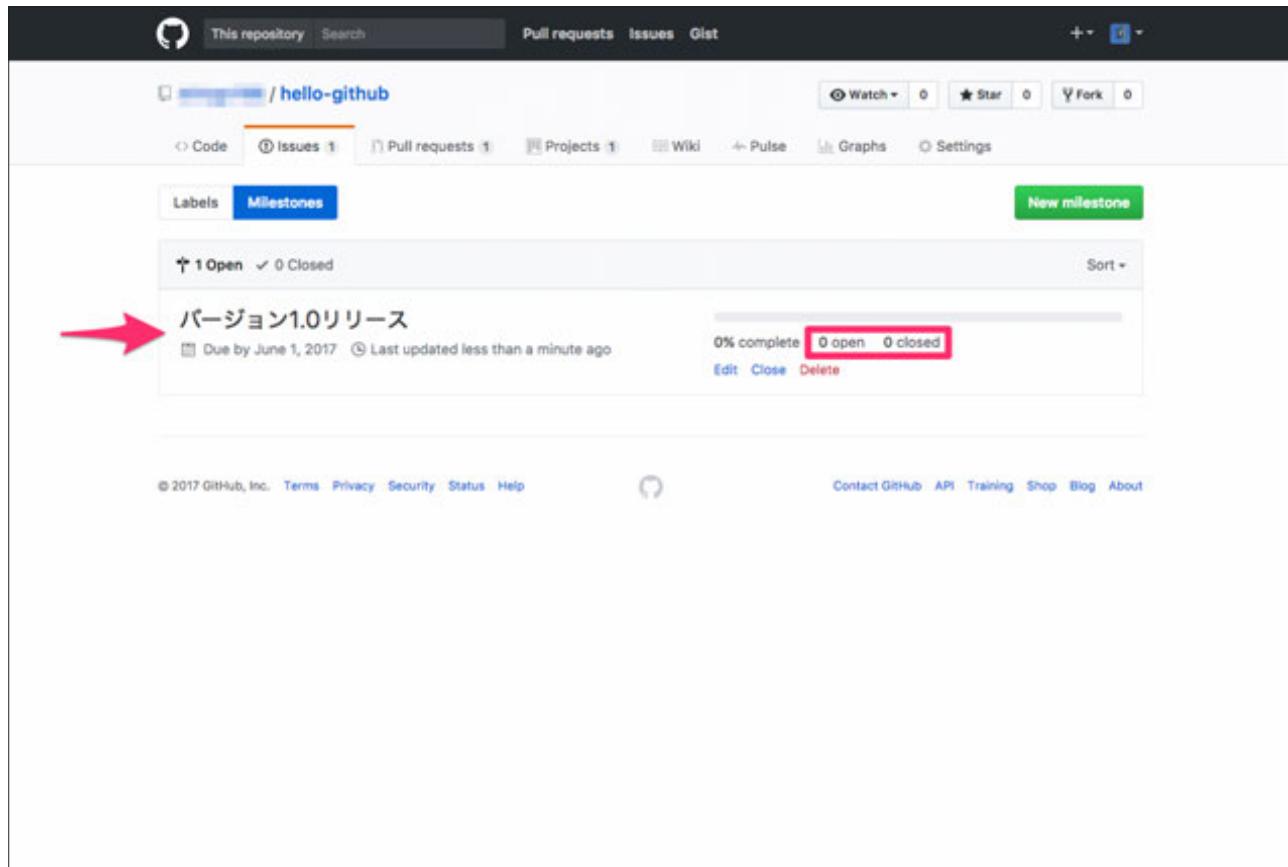


図 42 マイルストーン追加後のマイルストーン管理ページ

マイルストーンを割り当てる

既存のイシュー（プルリクエスト）にマイルストーンを割り当てるには、対象のイシュー（プルリクエスト）のページを開きます。

サイドバーの「Milestones」またはその横の歯車アイコンをクリックし、割り当てたいマイルストーンをクリックします。

The screenshot shows a GitHub repository page for 'hello-github'. The main navigation bar includes 'Pull requests', 'Issues', and 'Gist'. Below the navigation, there are tabs for 'Code', 'Issues 2', 'Pull requests 1', 'Projects 1', 'Wiki', 'Pulse', 'Graphs', and 'Settings'. The 'Issues' tab is active, showing an open issue titled 'プレイリスト画面の作成 #6'. The issue details include a summary: 'プレイリストの一覧を表示する画面を作成する', a task list with three items, and a comment from a user. On the right side, there are sections for 'Assignees', 'Labels', 'Projects', and 'Milestone'. A modal window titled 'Set milestone' is open, showing a dropdown menu with 'バージョン1.0リリース' selected. An arrow points to this selection. Other options in the dropdown include 'Due by June 1, 2017', 'Open', 'Closed', and 'Filter milestones'.

図 43 イssueのページ

マイルストーンの割り当てが実行されました。

This screenshot shows the same GitHub issue page after the milestone has been assigned. The 'Milestone' dropdown menu is open, and the previously selected 'バージョン1.0リリース' is highlighted with a red box. The rest of the interface remains the same, including the issue title, summary, task list, and the 'Leave a comment' section.

図 44 マイルストーン割り当て後のイissueのページ

マイルストーン管理ページに戻ると、「バージョン 1.0 リリース」を割り当てているイシュー（プルリクエスト）が 1 件になっていることを確認できます。

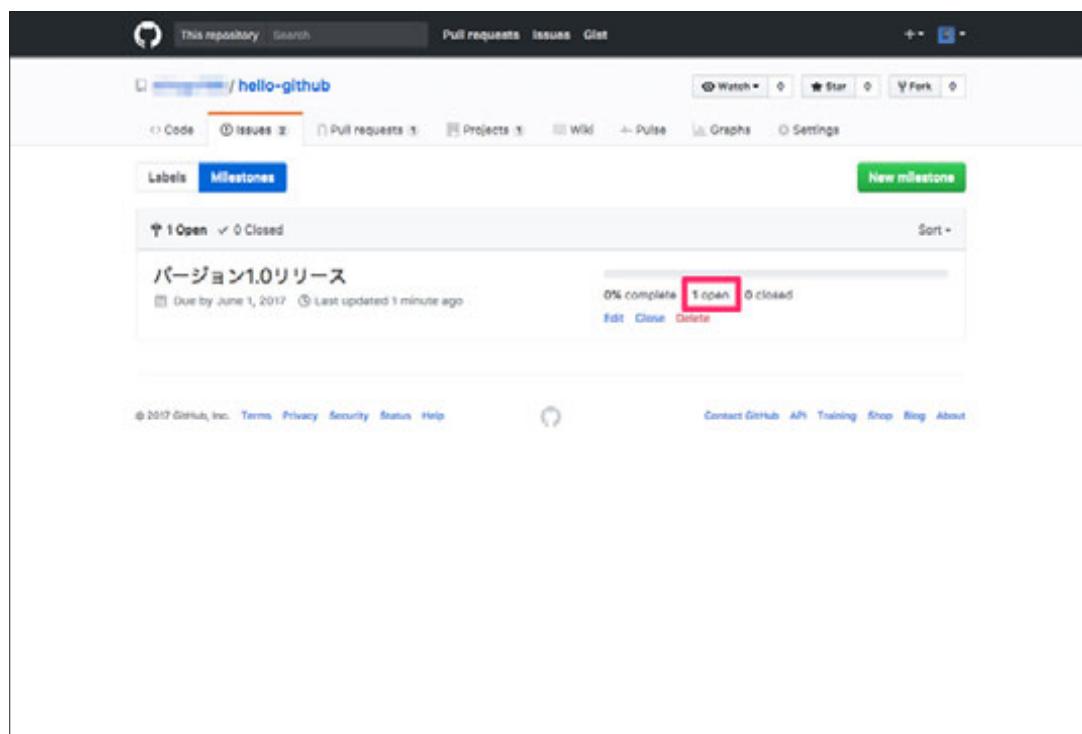


図 45 マイルストーン割り当て後のイシュー（プルリクエスト）の件数

また、このページでは、進捗率を確認することもできます。現在、進行中のイシュー（プルリクエスト）が 1 件、クローズしたイシュー（プルリクエスト）が 0 件なので、進捗率は 0% です。進行中のイシュー（プルリクエスト） 1 件を閉じると、進捗率は 100% になります。

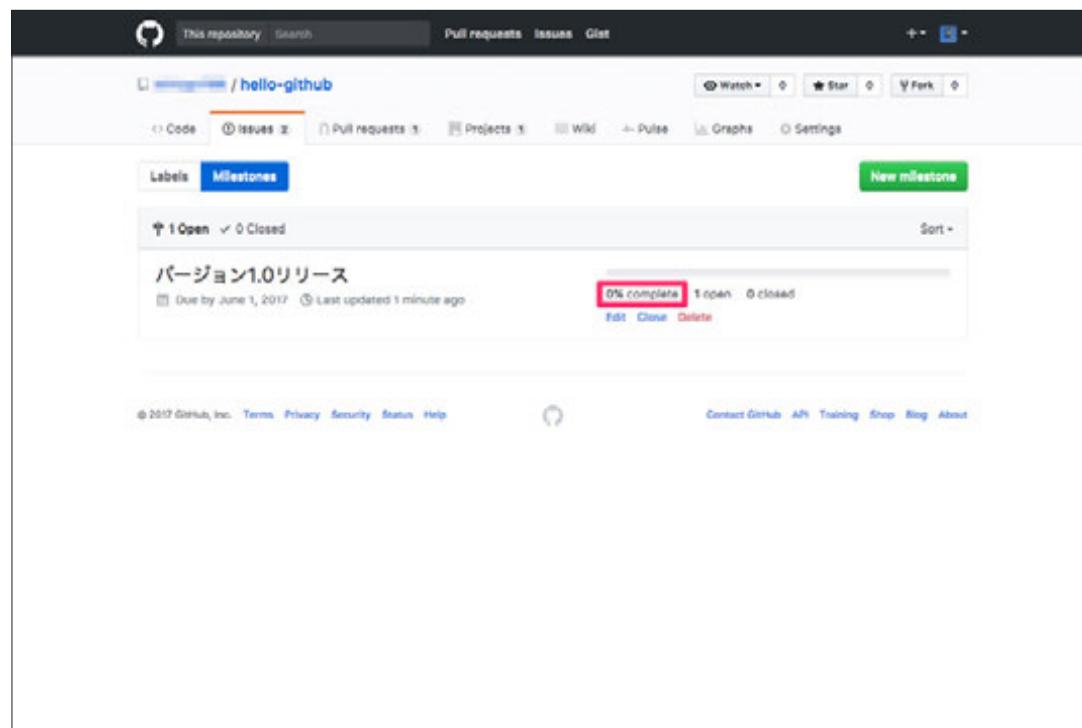


図 46 マイルストーンの進捗率

マイルストーン管理ページでマイルストーンのタイトルをクリックすると、以下のようなマイルストーンの個別ページを表示できます。マイルストーンを割り当てているイシュー（プルリクエスト）の一覧を確認できます。

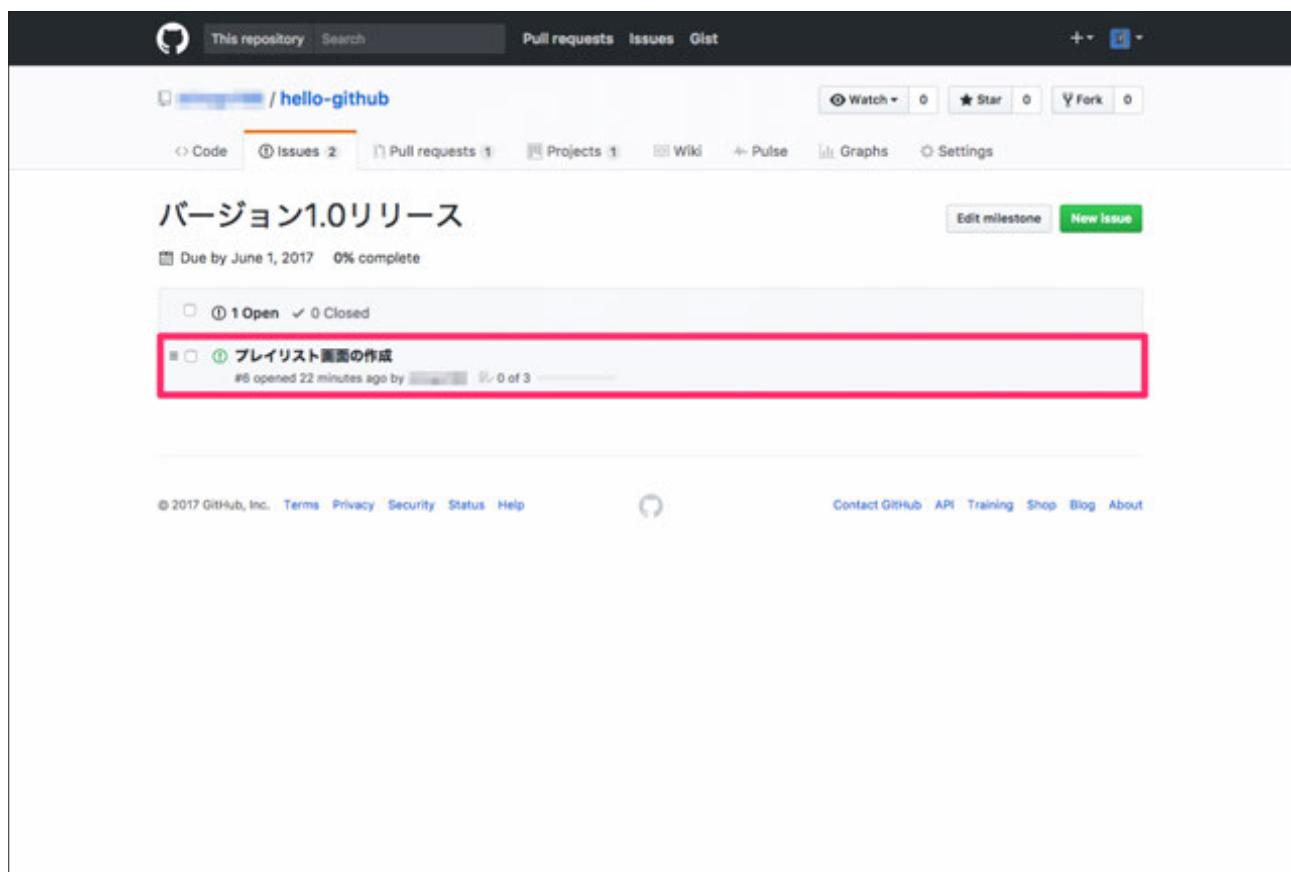


図 47 マイルストーンの個別のページ

マイルストーンを編集する

再びマイルストーン管理ページでの操作に戻ります。

マイルストーンの設定を編集するには、対象マイルストーンの「Edit」をクリックします。

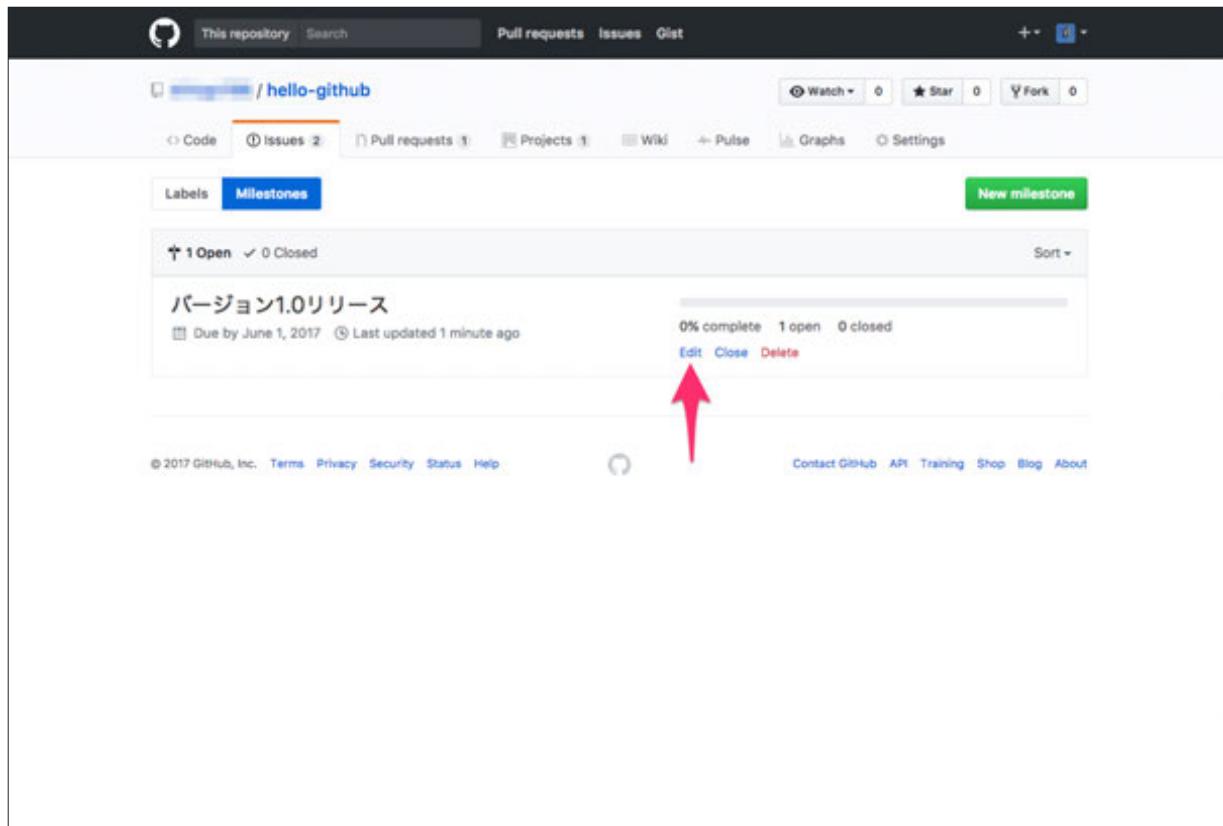


図 48 マイルストーン管理ページ

マイルストーンを新規に追加したときと同じ要領で、名前の入力や期限の選択を行い「Save change」をクリックします。

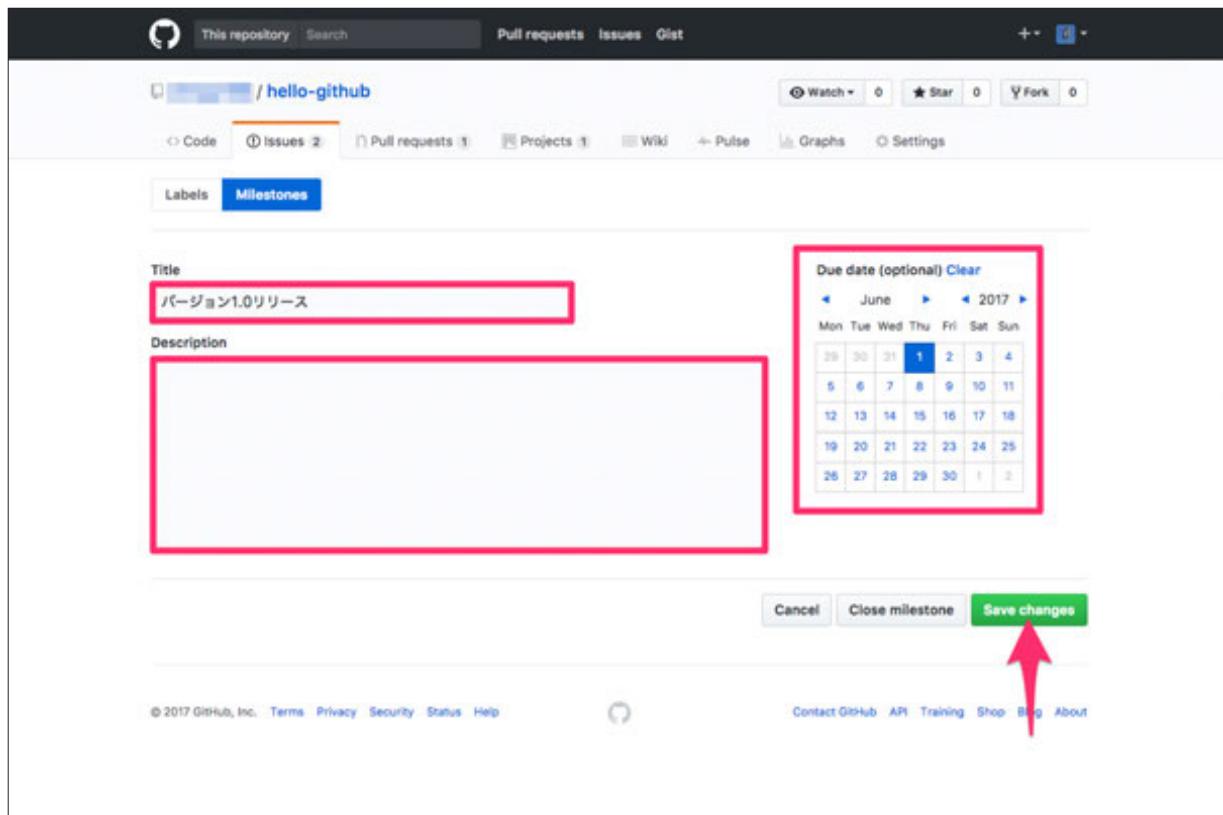


図 49 マイルストーン編集ページ

マイルストーンを削除する

マイルストーンを削除するには、対象マイルストーンの「Delete」をクリックします。

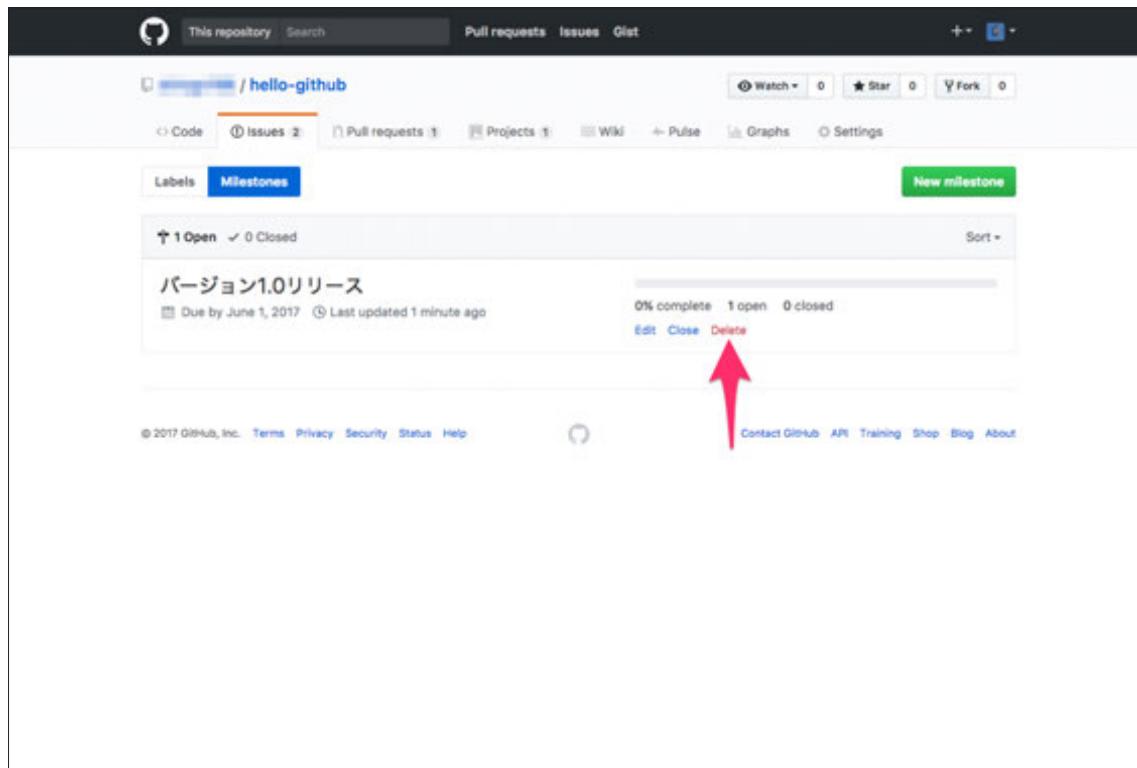


図 50 マイルストーン管理ページ

確認メッセージが表示されます。「Delete this milestone」をクリックすると、マイルストーンの削除を実行できます。

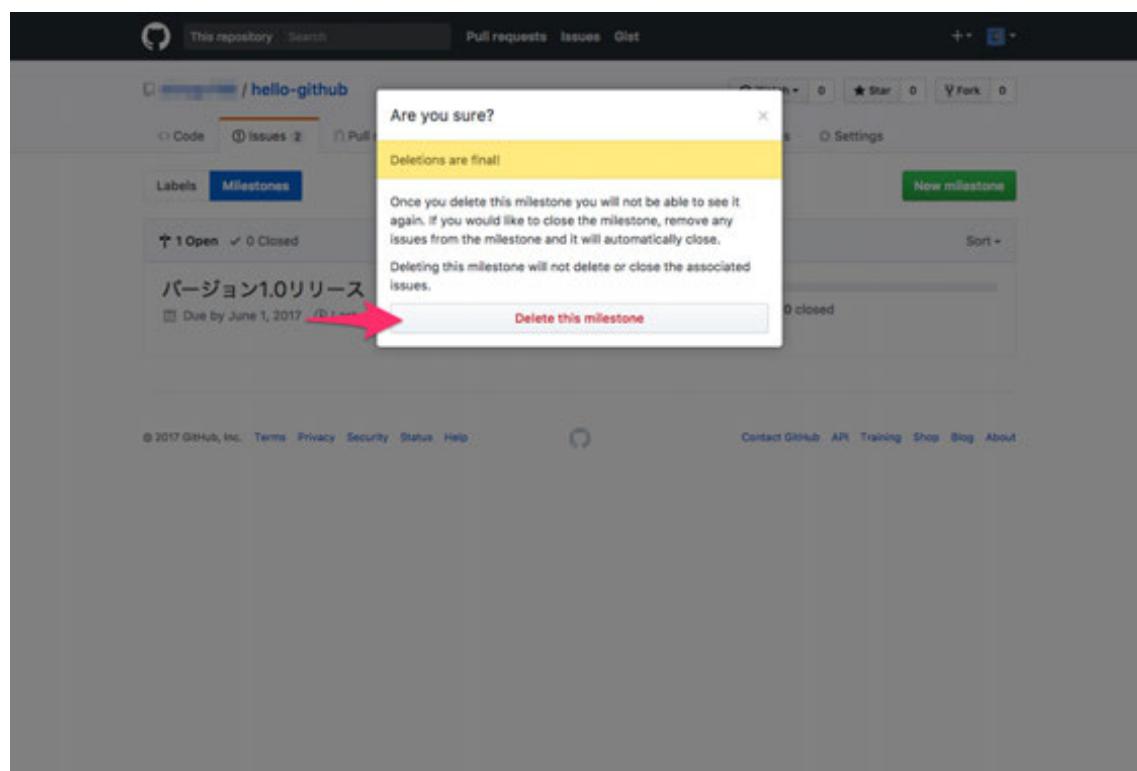


図 51 マイルストーン削除確認メッセージ

マイルストーンが割り当てられているイシュー（プルリクエスト）の一覧を表示する

最後に、特定のマイルストーンが割り当てられているイシュー（プルリクエスト）の一覧を表示する方法を紹介します。

前回の記事で、イシューの一覧ページの「フィルタ機能」を紹介しました。イシューの一覧ページでは「マイルストーン」でイシュー（プルリクエスト）を絞り込むこともできます。

イシュー（プルリクエスト）をマイルストーンで絞り込むには、イシューの一覧ページで、以下のように「Milestones」をクリックし、リストのマイルストーンをクリックします。

The screenshot shows the GitHub Issues page for the repository 'hello-github'. At the top, there are tabs for 'Issues' (2), 'Pull requests' (1), 'Projects' (1), 'Wiki', 'Pulse', 'Graphs', and 'Settings'. Below the tabs are buttons for 'Watch' (0), 'Star' (0), 'Fork' (0), and a green 'New issue' button. A red arrow labeled '①' points from the 'Issues' tab to the 'Milestones' dropdown menu. Another red arrow labeled '②' points to the 'Milestone' dropdown menu, which is open, showing options like 'Filter by milestone', 'Filter milestones', and 'Issues with no milestone'. The main list of issues includes two items: '#6 opened 33 minutes ago by [redacted]' and '#2 opened on Mar 18 by [redacted]'. The second item is highlighted with a blue background and the text 'バージョン1.0リリース'.

図 52 マイルストーンで絞り込む

対象のマイルストーンが割り当てられているイシュー（プルリクエスト）だけ表示されるようになりました。

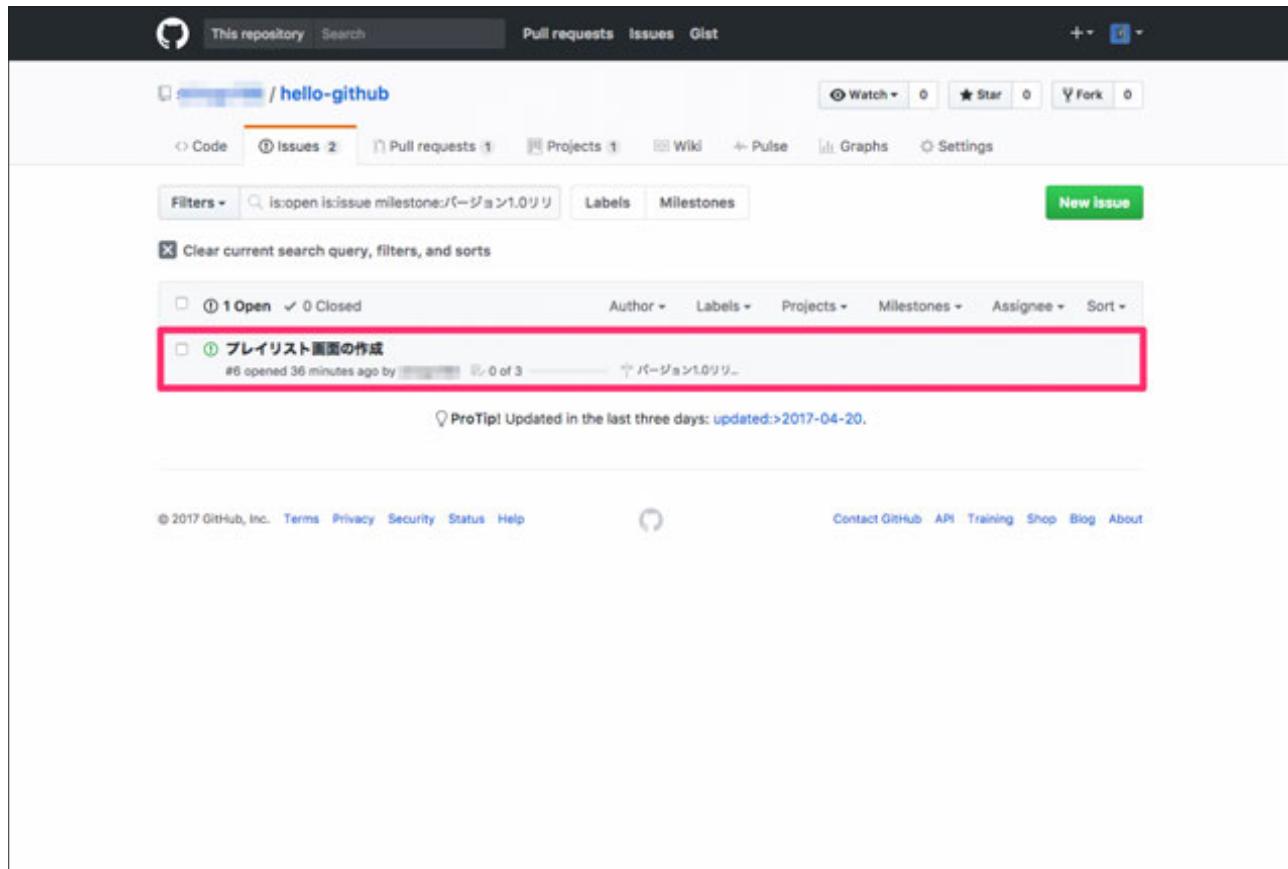


図 53 特定のマイルストーンが割り当てられているイシューの一覧

次回は「外部サービス連携や Wiki などの機能」について

本稿ではイシュー や プルリクエスト の周辺機能を解説しました。

今回解説した機能は必ず使用しなければいけない機能ではありません。しかし、これらの機能をうまく活用できれば、開発にまつわるタスクやスケジュールの管理を改善できるかもしれません。

次回は「外部サービス連携や Wiki などの機能」を解説する予定です。お楽しみに!

13.GitHub と Slack の連携の基本&知られざる便利機能 Wiki、Releases、Graphs、Pulse

(2017年05月29日)

本連載では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説しています。今回は、Pulse、Graphs、Releases、Wiki、Slackを例にした外部サービスとの連携について解説します。

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終える頃には、GitやGitHubの基本的な操作が身に付いた状態になっていると思います。

前回の記事「開発者のスケジュール管理に超便利、GitHub Issues、Label、Milestone、Projects 使いこなし術」ではイシュー・プルリクエストの周辺機能を解説しました。

連載第13回目の本稿では、Pulse、Graphs、Releases、Wiki、外部サービスとの連携について解説します。

リポジトリ上での直近のアクティビティーを表示する「Pulse」

Pulseとは、リポジトリ上での直近のアクティビティーを表示する機能です。下記の操作がアクティビティーとして扱われます。

- プルリクエストの作成
- プルリクエストのマージ
- イシューの作成
- イシューのクローズ

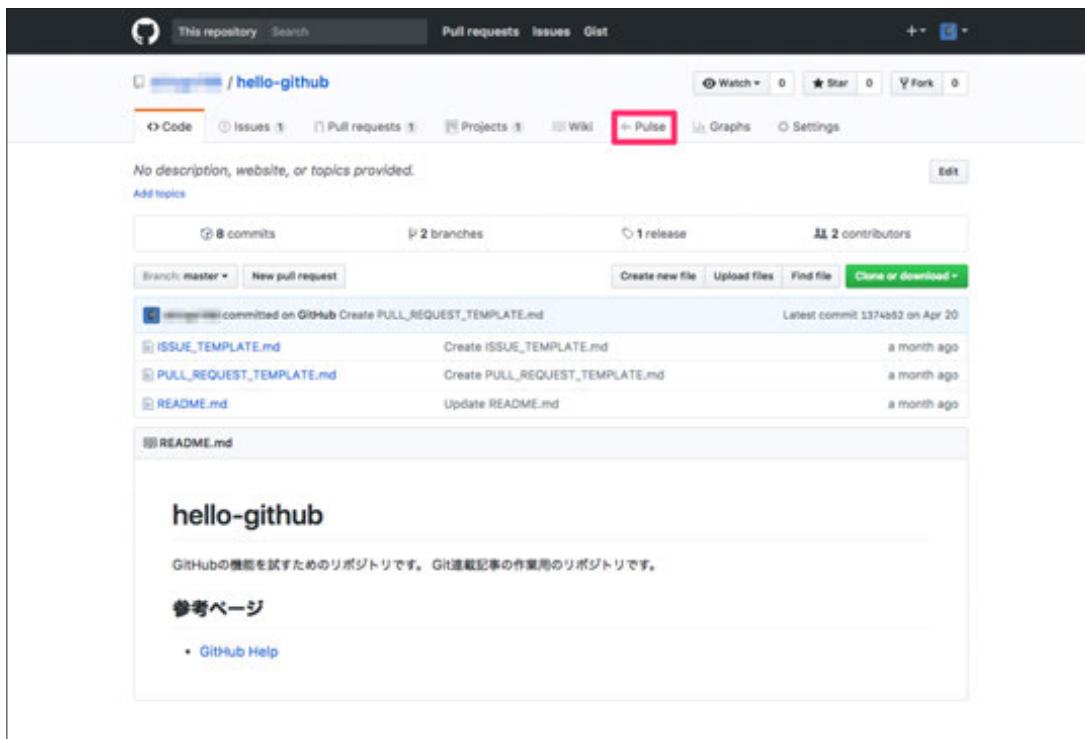
Pulseのページでは、下記のいずれかの期間内に発生したアクティビティーを表示できます。

- 24時間以内
- 3日以内
- 1週間以内
- 1ヶ月以内

直近で行われた操作が列挙されるので、リリースノートを書く際などに役に立つと思います。

Pulse ページを表示する

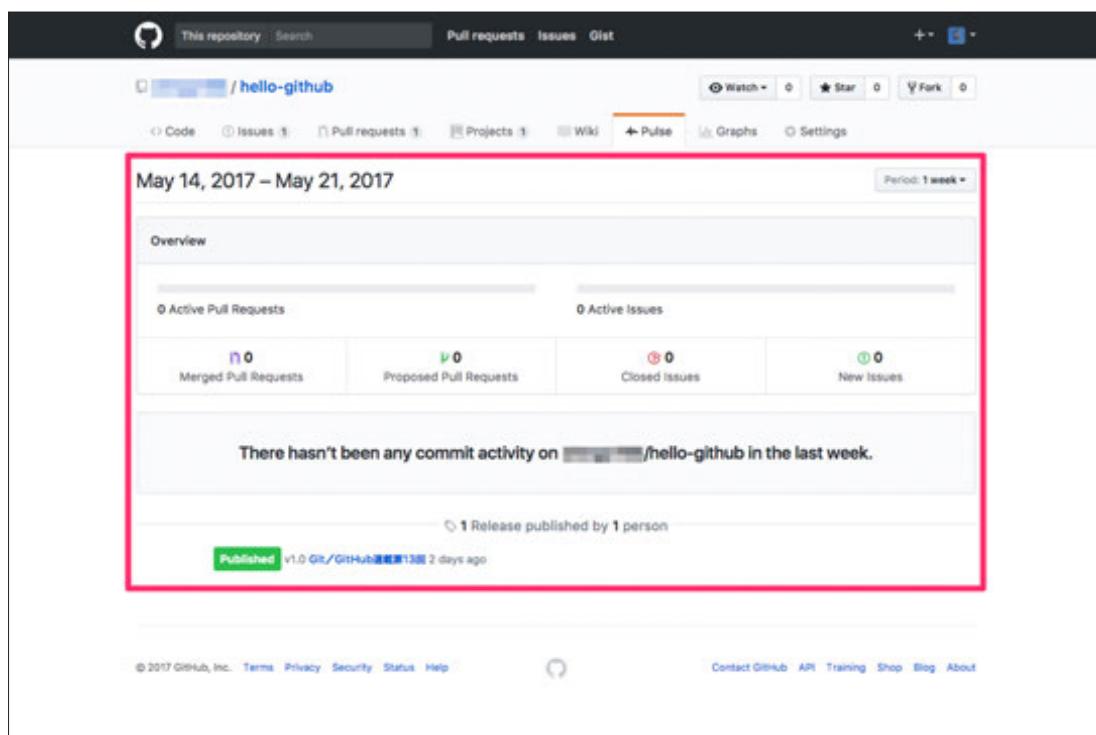
Pulse ページを表示するには、リポジトリのメインタブの「Pulse」を選択します。



The screenshot shows the main repository page for 'hello-github'. The 'Pulse' tab is highlighted with a red box. The page displays basic repository statistics: 8 commits, 2 branches, 1 release, and 2 contributors. Below this, a list of recent activities is shown, including committing files like 'ISSUE_TEMPLATE.md', 'PULL_REQUEST_TEMPLATE.md', and 'README.md'. A large section titled 'hello-github' provides a brief description of the repository as a place to try GitHub features. A '参考ページ' (Reference Page) section includes a link to 'GitHub Help'.

図 1 リポジトリトップページ

Pulse ページが表示されました。デフォルトでは、1週間以内に発生したアクティビティーが表示されます。



The screenshot shows the Pulse page for the same repository, with the 'Pulse' tab selected. A red box highlights the date range 'May 14, 2017 – May 21, 2017'. The page displays an 'Overview' section with metrics: 0 Active Pull Requests, 0 Active Issues, 0 Merged Pull Requests, 0 Proposed Pull Requests, 0 Closed Issues, and 0 New Issues. A message states 'There hasn't been any commit activity on [REDACTED]/hello-github in the last week.' Below this, it shows 1 Release published by 1 person. The bottom of the page includes standard GitHub footer links and a 'Published' note.

図 2 Pulse ページ (1週間以内のアクティビティー表示)

対象期間を変更する

期間を変更するには、右上の「Period 1 week」ボタンをクリックして、表示されるメニューのいずれかの期間を選択します。

ここでは「1 month」を選択してみます。

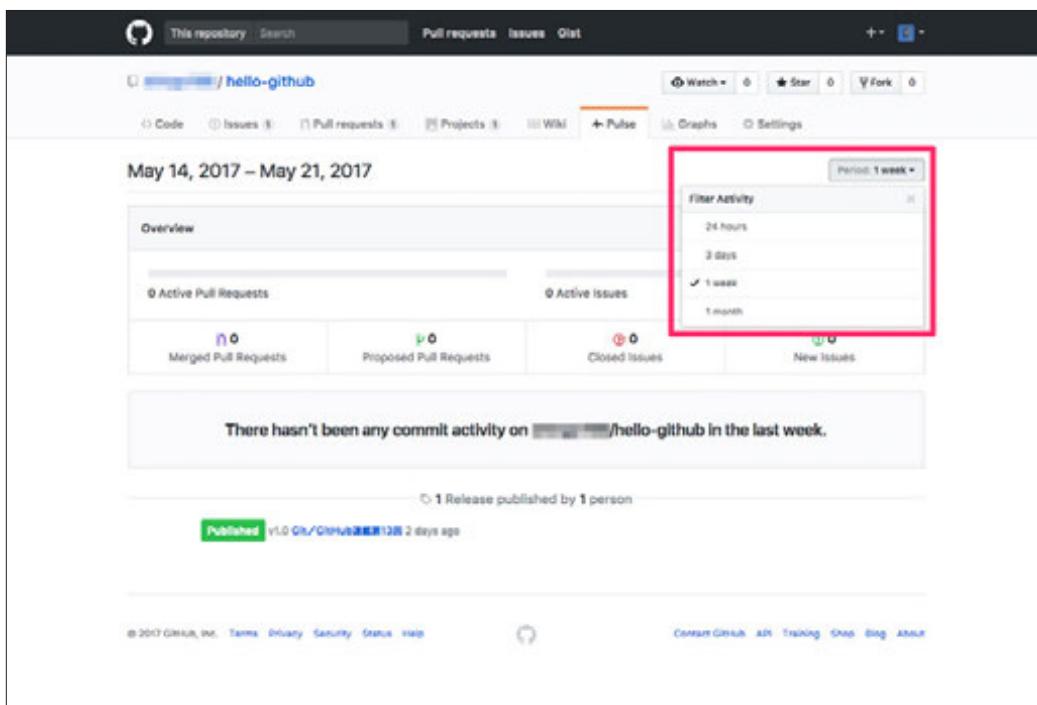


図 3 Pulse の対象期間変更

1 ル月以内に発生したアクティビティが表示されるようになりました。

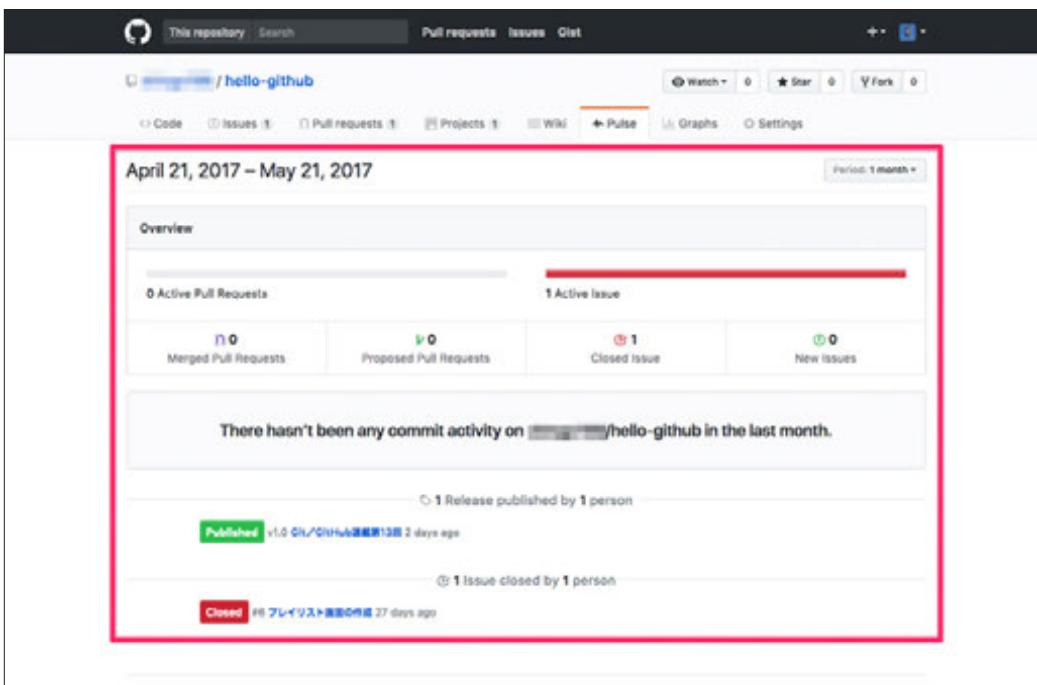


図 4 Pulse ページ (1 ル月以内のアクティビティ表示)

リポジトリに関する各種データの確認や分析を行う「Graphs」

Graphs とは、リポジトリに関する各種データの確認や分析を行うための機能です。

Graphs ページを表示するには、リポジトリのメインタブの「Graphs」を選択します。

The screenshot shows the GitHub repository interface for 'hello-github'. At the top, there are tabs for 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Pulse', and 'Graphs'. The 'Graphs' tab is highlighted with a red box. Below the tabs, there's a summary section with counts for commits, branches, releases, and contributors. A list of recent commits is shown, each with a small profile picture, commit message, creation date, and a 'View file' link. Below this is a section titled 'hello-github' with a brief description and a '参考ページ' (Reference Page) section containing a single link to 'GitHub Help'.

図 5 リポジトリトップページ

Graphs ページが表示されました。上部のタブを選択することによって、表示されるデータを切り替えることができます。デフォルトでは「Contributors」タブが選択されます。

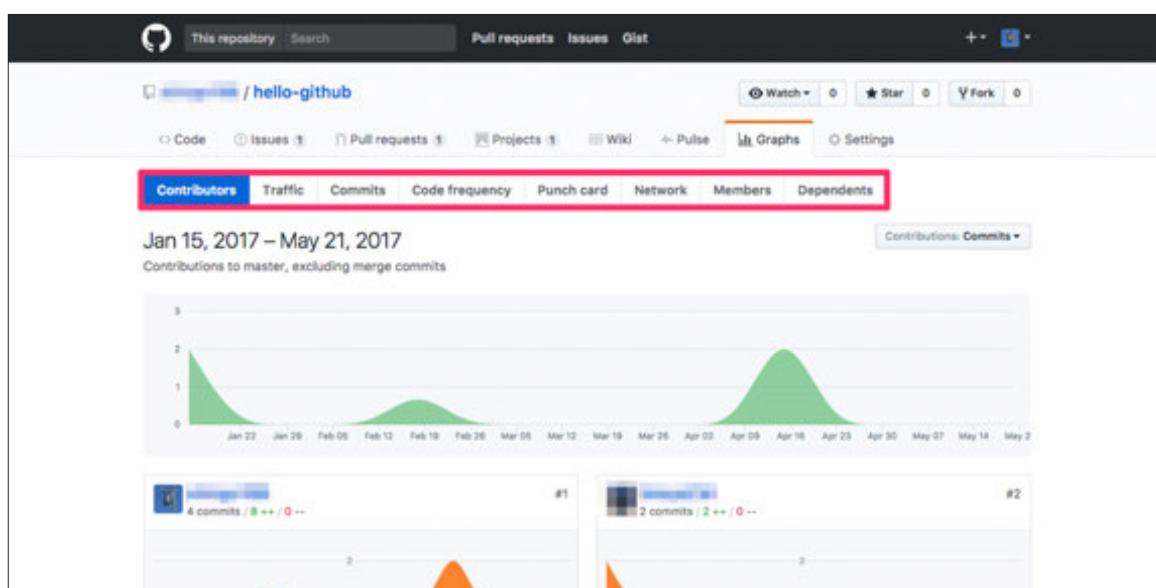


図 6 Graphs ページ

表示されるデータをタブごとに見ていきましょう。

Contributors

このタブを選択すると、リポジトリに貢献した回数のグラフを確認できます。一番上にリポジトリ全体のグラフが表示され、その下にユーザーごとのグラフが表示されます。

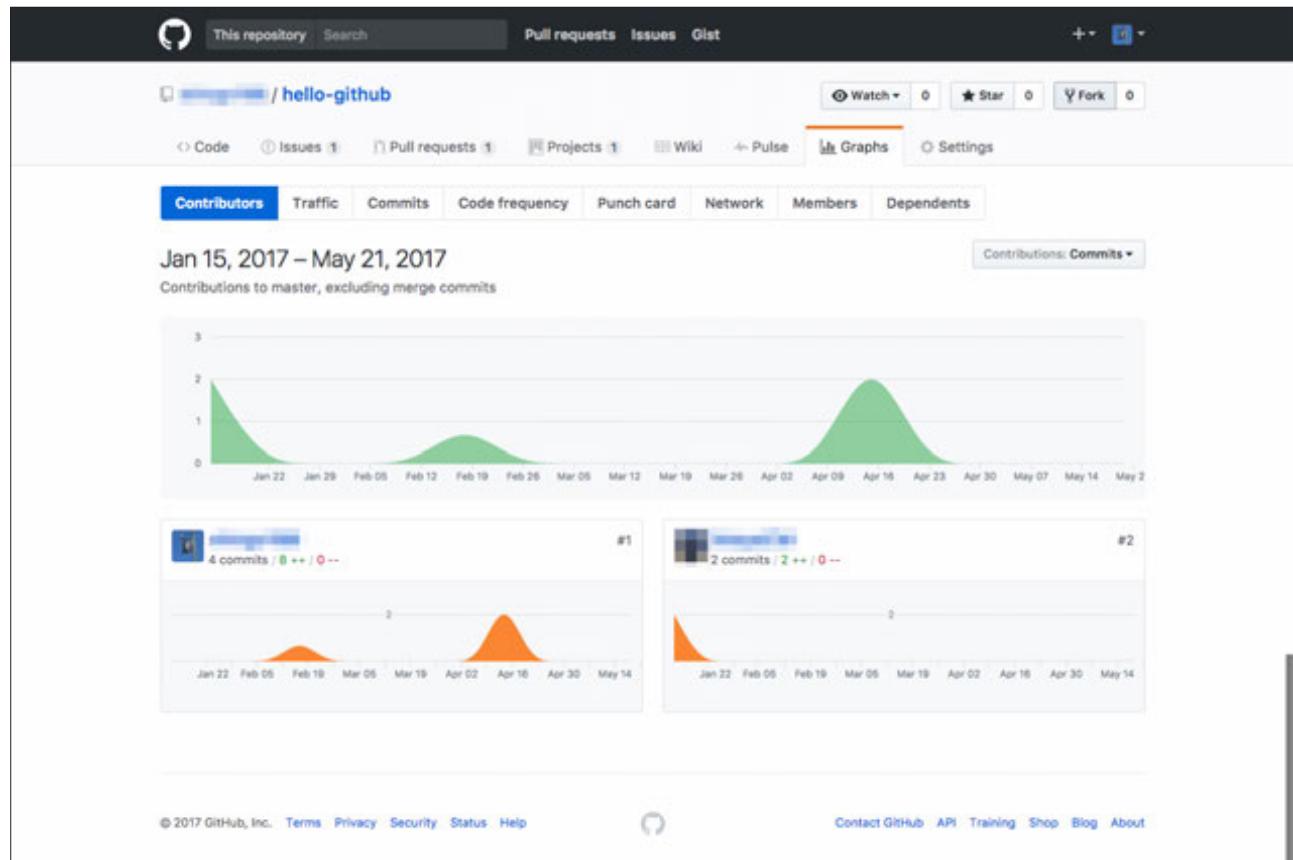


図 7 Graphs ページ (Contributors)

Traffic

このタブを選択すると、リポジトリへのアクセスに関する下記のデータを確認できます。

- クローンされた数
- 訪問者の数
- 流入元のリスト
- よく見られているコンテンツのリスト

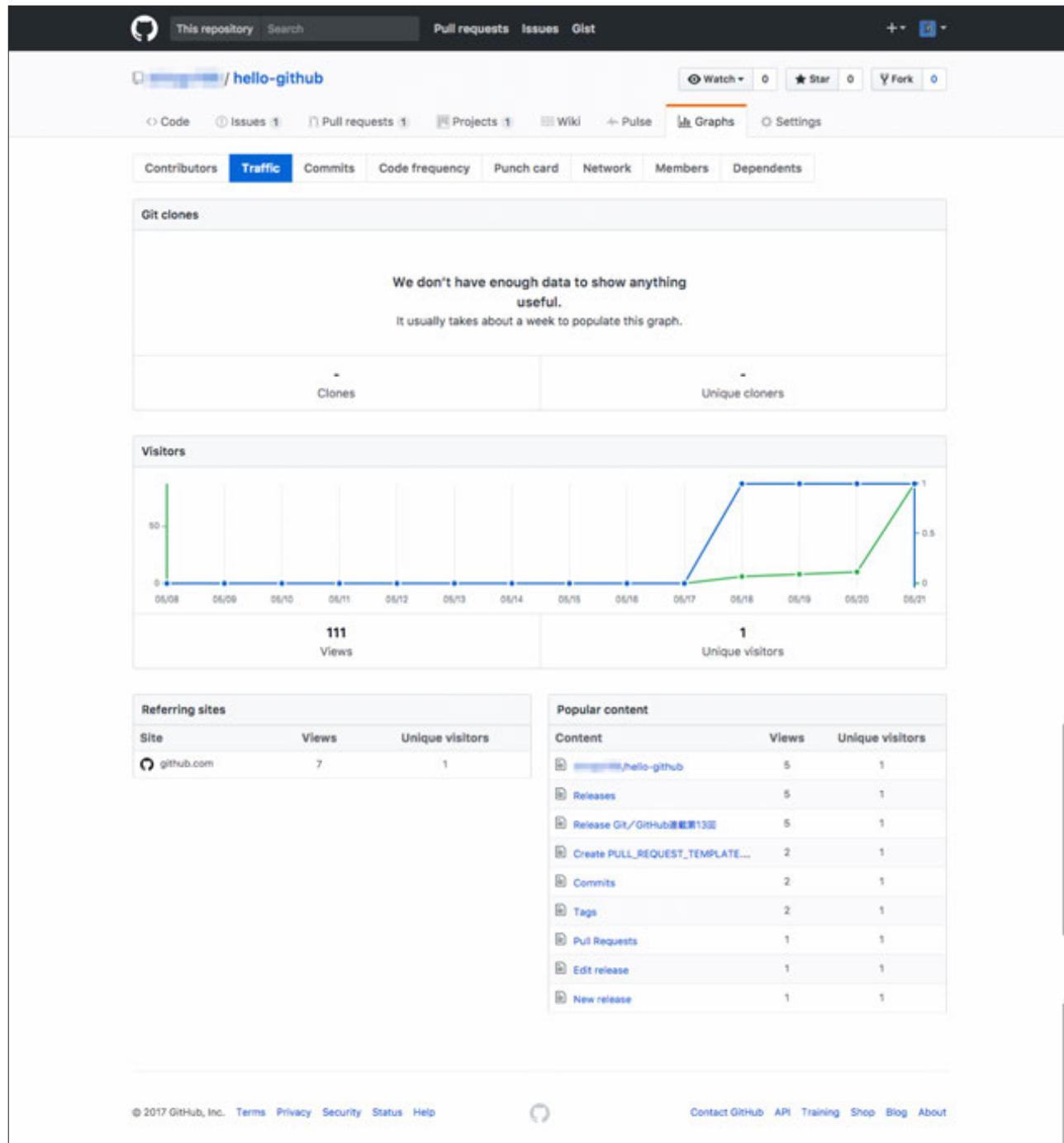


図 8 Graphs ページ (Traffic)

Commits

このタブを選択すると、コミット回数のグラフを確認できます。

上部の棒グラフは、1週間ごと、1年分のコミット回数を表します。

下部の折れ線グラフは、「選択中の週」の曜日ごとにおけるコミット回数の分布を表します。上部グラフのうち下に点が付いている週が「選択中の週」であり、左右の十字キーを使用すると「選択中の週」を他の週に変更できます。

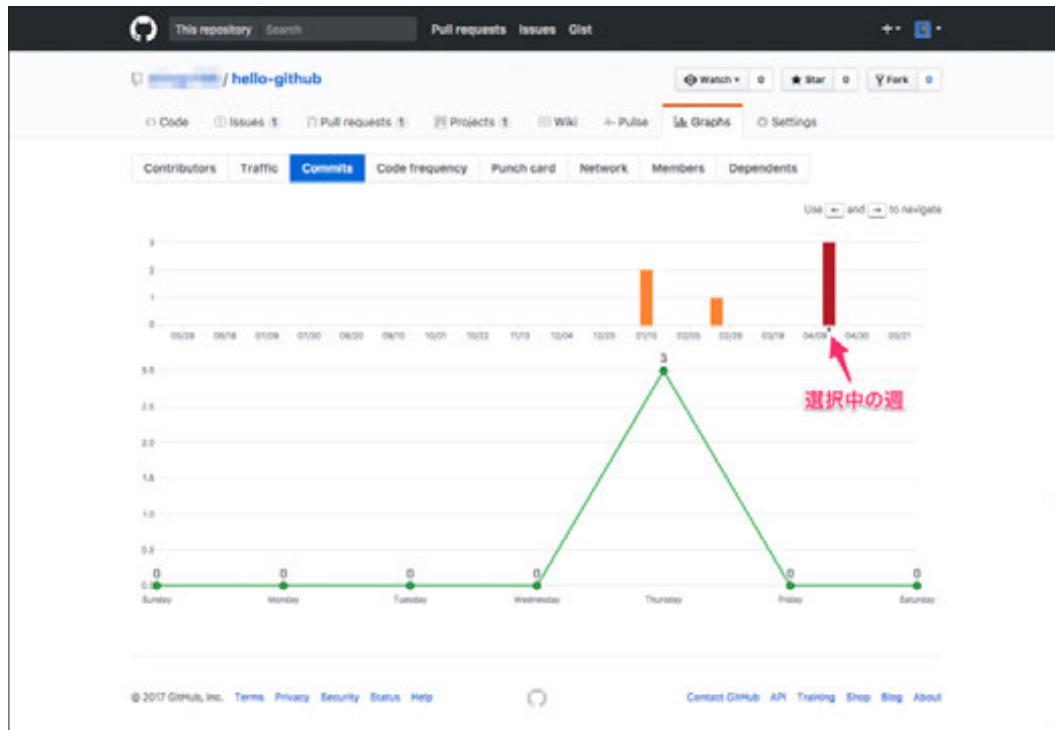


図 9 Graphs ページ (Commits)

Code frequency

このタブを選択すると、下記の数を表すグラフを確認できます。

- 1 週の間に追加した行の数
- 1 週の間に削除した行の数

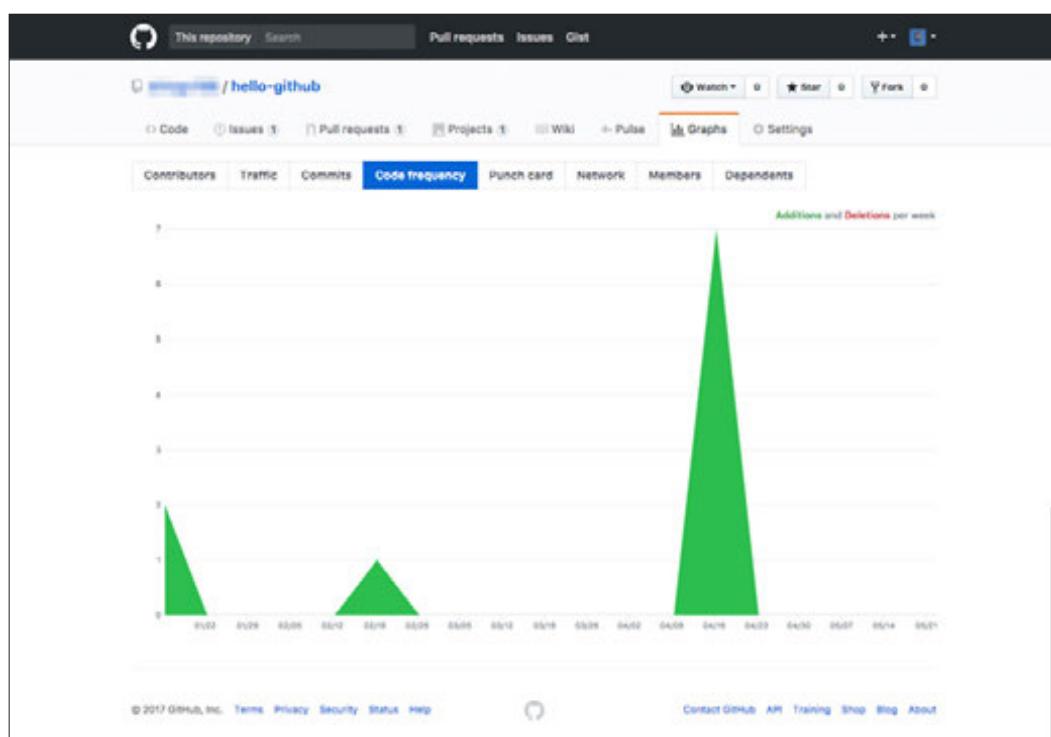


図 10 Graphs ページ (Code frequency)

Punch card

このタブを選択すると、曜日／時間帯ごとの、リポジトリの更新頻度を表すグラフを確認できます。円のサイズにはコミット頻度が反映されます。

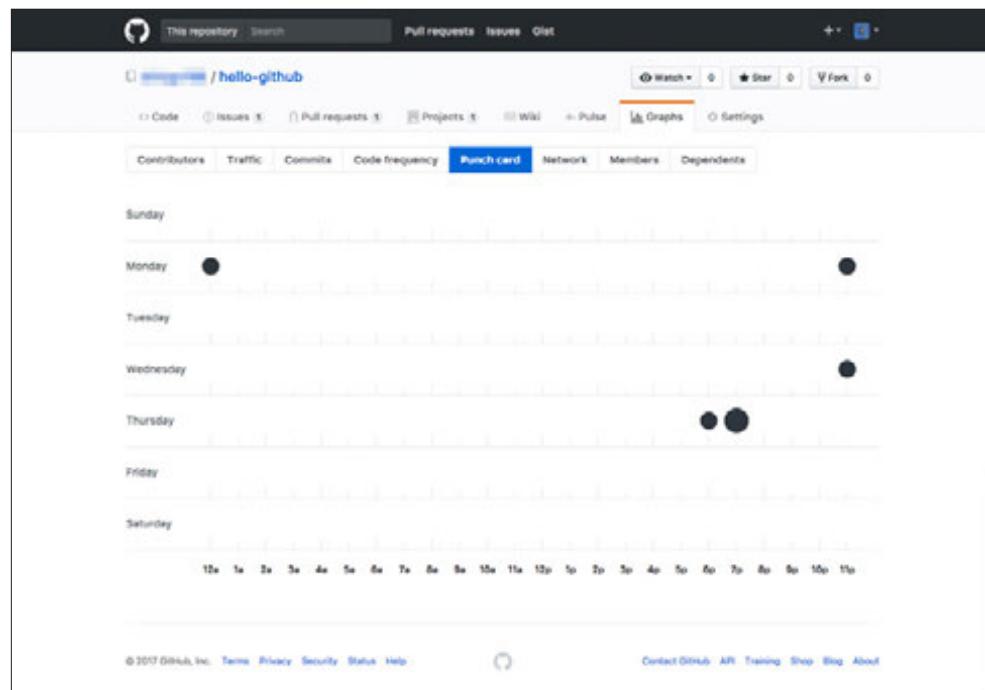


図 11 Graphs ページ (Punch card)

Network

このタブを選択すると、リポジトリのブランチの歴史のグラフを確認できます。

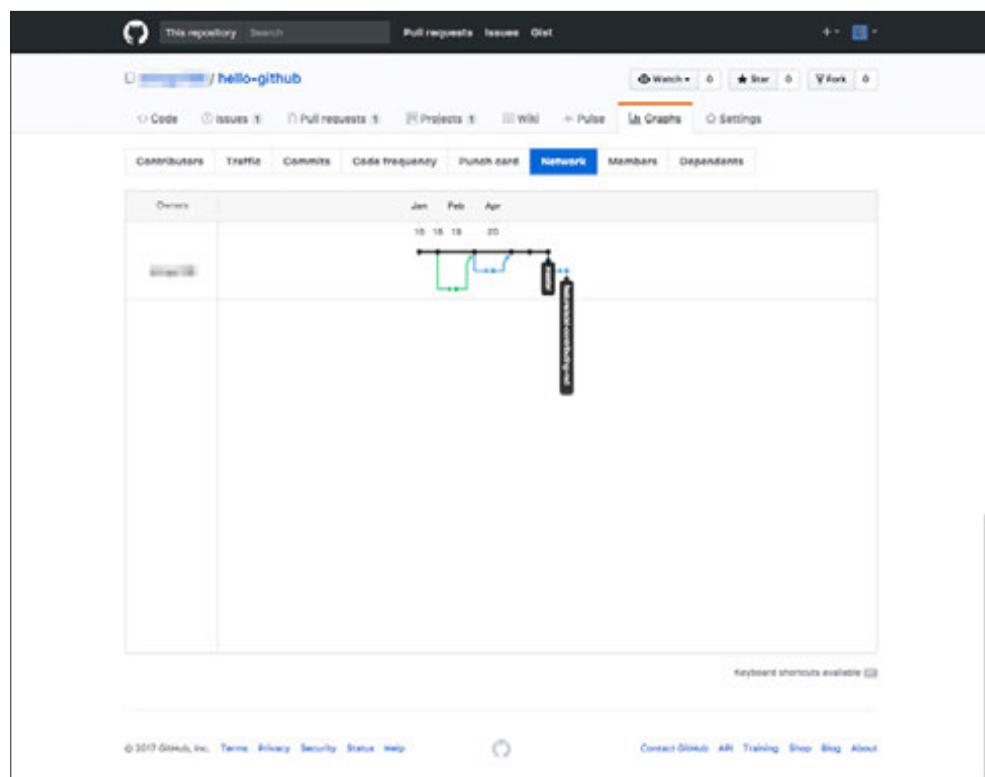


図 12 Graphs ページ (Network)

Members

このタブを選択すると、リポジトリをフォークしたユーザーのリストを確認できます。フォークとは、リポジトリの複製を作る操作のことです。

フォークしたユーザーがいなければ、図 13 のような表示になります。

The screenshot shows a GitHub repository page for 'hello-github'. At the top, there are tabs for 'Pull requests', 'Issues', 'Gist', 'Code', 'Issues', 'Pull requests', 'Projects', 'Wiki', 'Pulse', 'Graphs' (which is highlighted in orange), and 'Settings'. Below these, there are tabs for 'Contributors', 'Traffic', 'Commits', 'Code frequency', 'Punch card', 'Network', 'Members' (which is highlighted in blue), and 'Dependents'. The main content area contains a message: 'No one has forked this repository yet.' followed by a note: 'Forks are a great way to contribute to a repository. After forking a repository, you can send the original author a pull request.' At the bottom, there are links for 'Contact GitHub', 'API', 'Training', 'Shop', 'Blog', and 'About'.

図 13 Graphs ページ (Members)

Dependents

このタブを選択すると、ソフトウェアパッケージの依存に関するデータを表示できます。

他のリポジトリから依存されている場合は、ここにリポジトリのリストが表示されます。Ruby gems でパッケージ管理されている場合のみが対象のようです。

他のリポジトリからの依存がない場合は、図 14 のような表示になります。

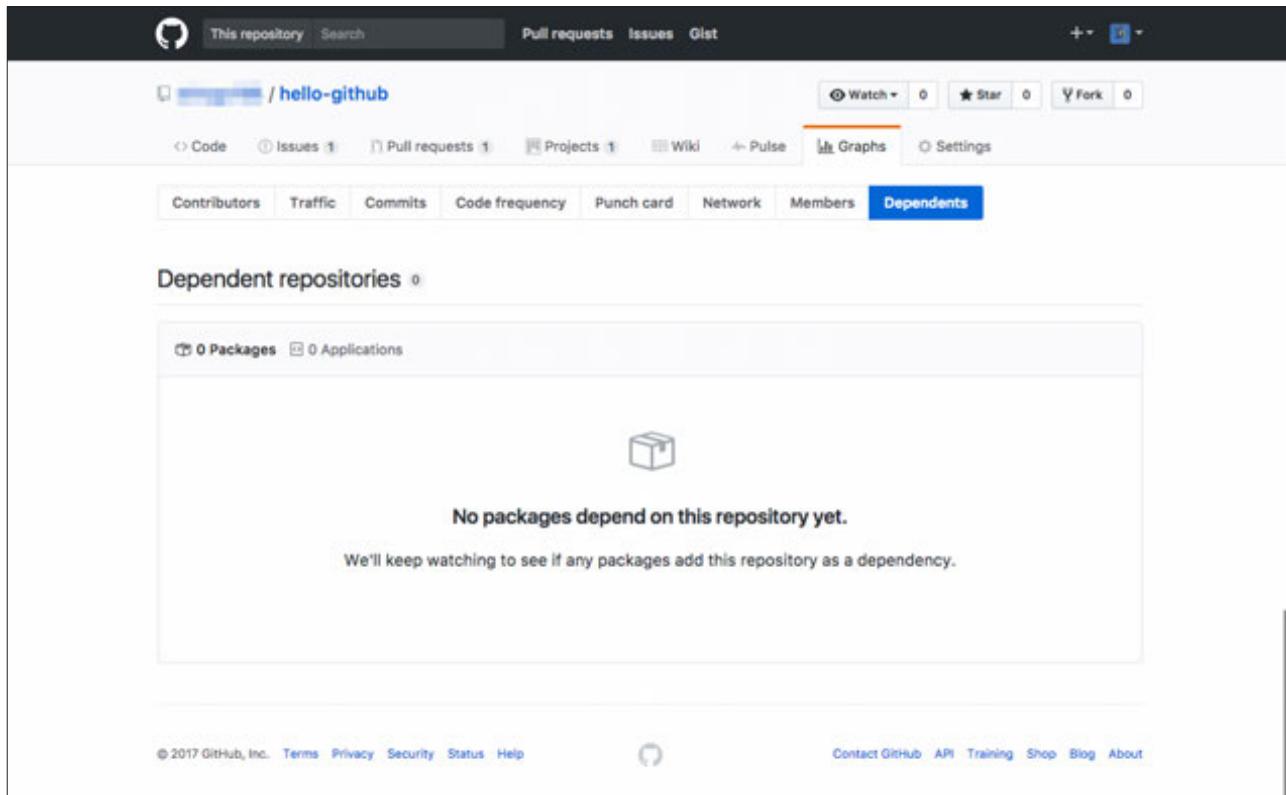


図 14 Graphs ページ (Dependents)

特定のコミットに名前を付ける「タグ」とリリース情報を作成、編集、削除する「Releases」

Releases とは、ある時点のソフトウェアを配布する際のリリース情報を作るための GitHub の機能です。Git の「タグ」という機能がベースになっています。

Git のタグは特定のコミットに名前を付ける機能です。タグを使えば、特定のコミットを参照しやすくなったり、後から特定のコミットへ戻しやすくなったりします。

例えば、コミット「1374b52」に「v1.0」というタグを付けたとします。すると、後からタグ「v1.0」を指定して、リポジトリをコミット「1374b52」の状態に戻すことができます。

GitHub の Releases 機能を使用してリリースを作成すると、「リリースノート」や「添付ファイル」をタグにひも付けることができます。

例えば、下記リストの内容のリリースを作成すると、バージョン 1.0 のソフトウェアを配布するための準備が整います。

- ・タグ: 「v1.0」
- ・リリースのタイトル: 「バージョン 1.0 リリース」
- ・リリースの説明: バージョン 1.0 の変更点（リリースノート）を記載
- ・添付ファイル: バージョン 1.0 のソフトウェア一式を格納した zip ファイル

GitHub 上でソフトウェアを配布する必要がない場合でも、重要なタイミングでリリースを作成しておけば、後から特定のリリースの内容を簡単に確認できるようになります。

リリースを作成する

リリースを作成してみましょう。GitHub 上のリポジトリページを表示し「Code」タブの中の「releases」をクリックします。

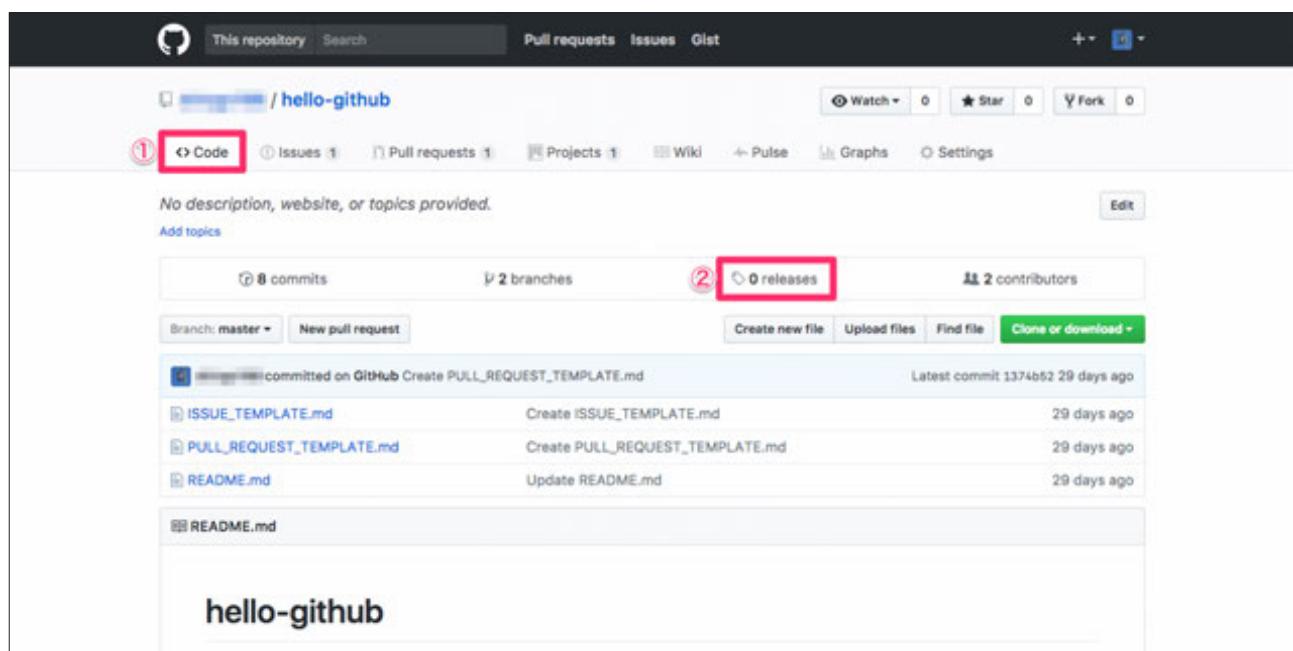


図 15 リポジトリトップページ

リリースを 1 つも作成していない場合、図 16 のような表示になります。「Create a new release」をクリックします。

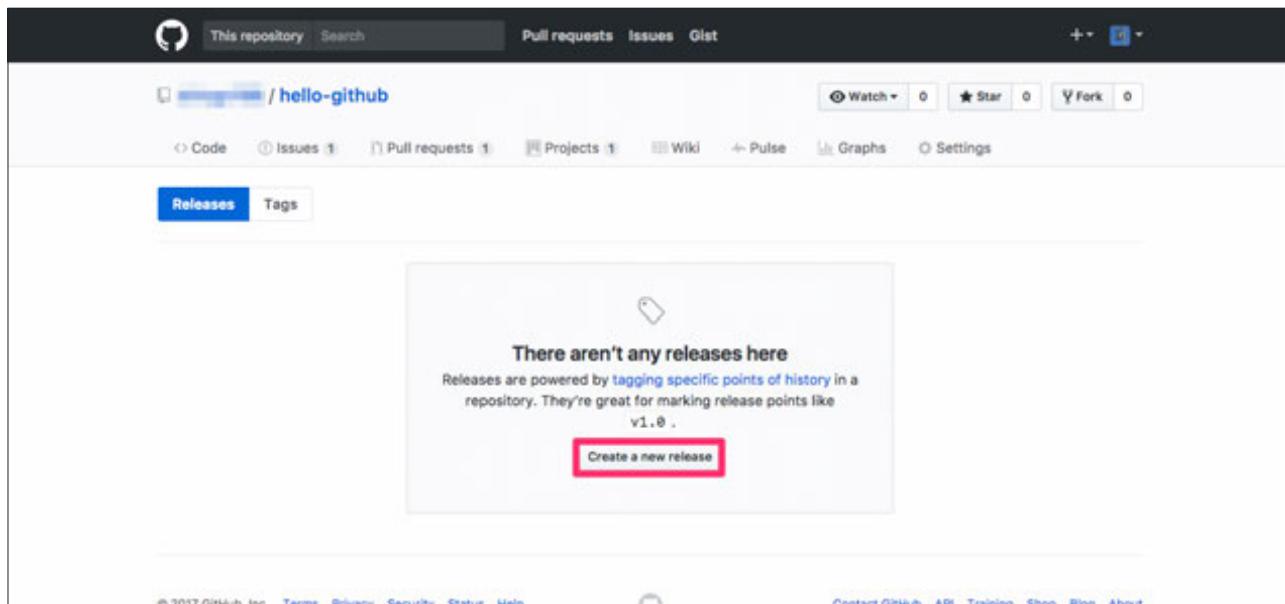


図 16 Release ページ

リリース作成ページが表示されます。「Tag version」にタグの名前を、「Release title」にリリースのタイトルを、「Describe this release」にリリースの説明を記入します。

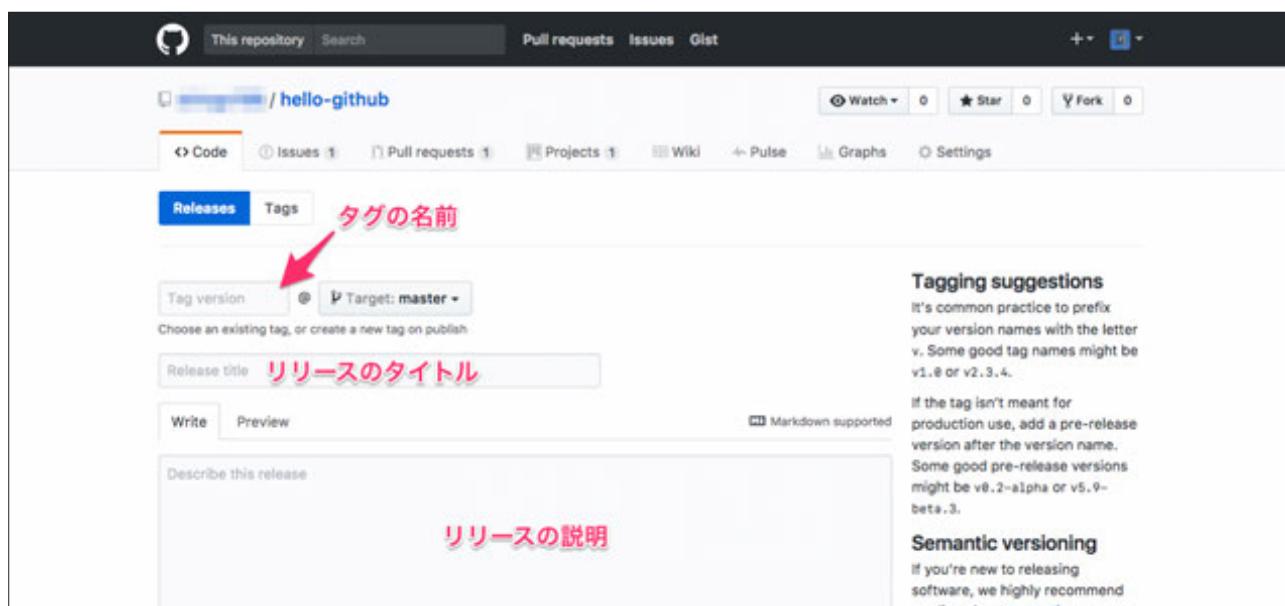


図 17 リリース作成ページ

今回は図 18 のように、それぞれ「v1.0」「Git / GitHub 連載第 13 回」「- Git / GitHub 連載第 13 回までの内容を反映」と入力しました。

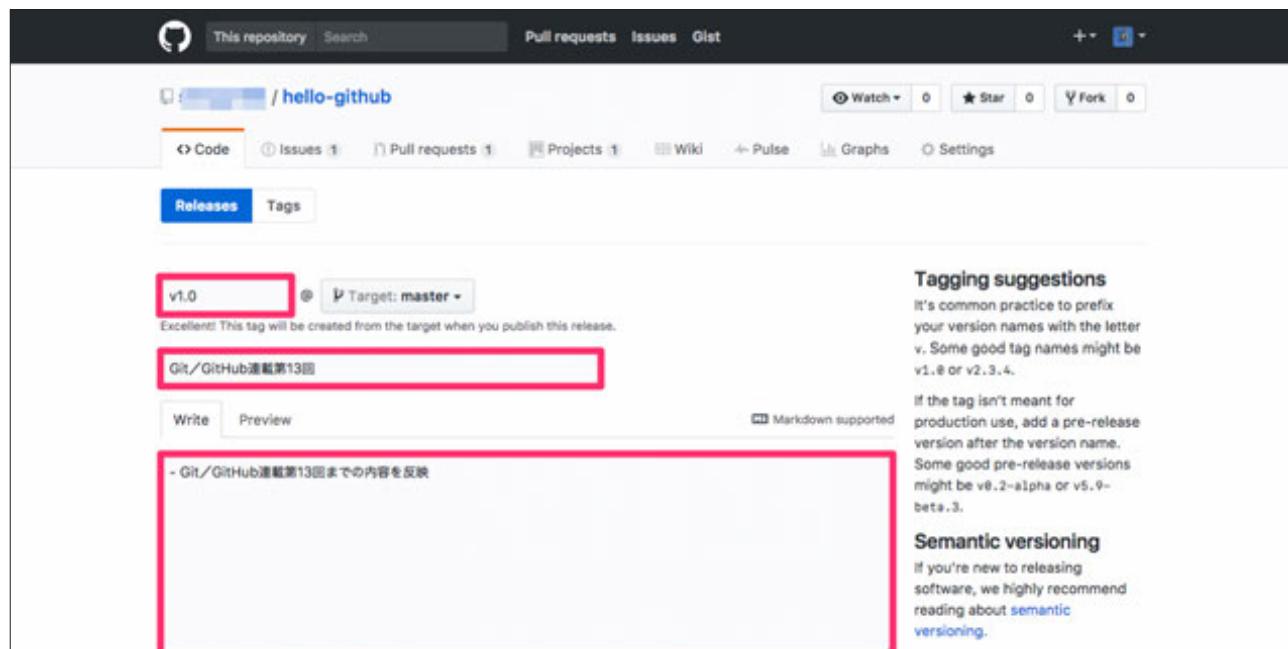


図 18 各値入力後のリリース作成ページ

「Describe this release」の下に、添付ファイル設定に関するエリアがあります。ここにファイルをドロップすると、リリースにファイルを添付できます。

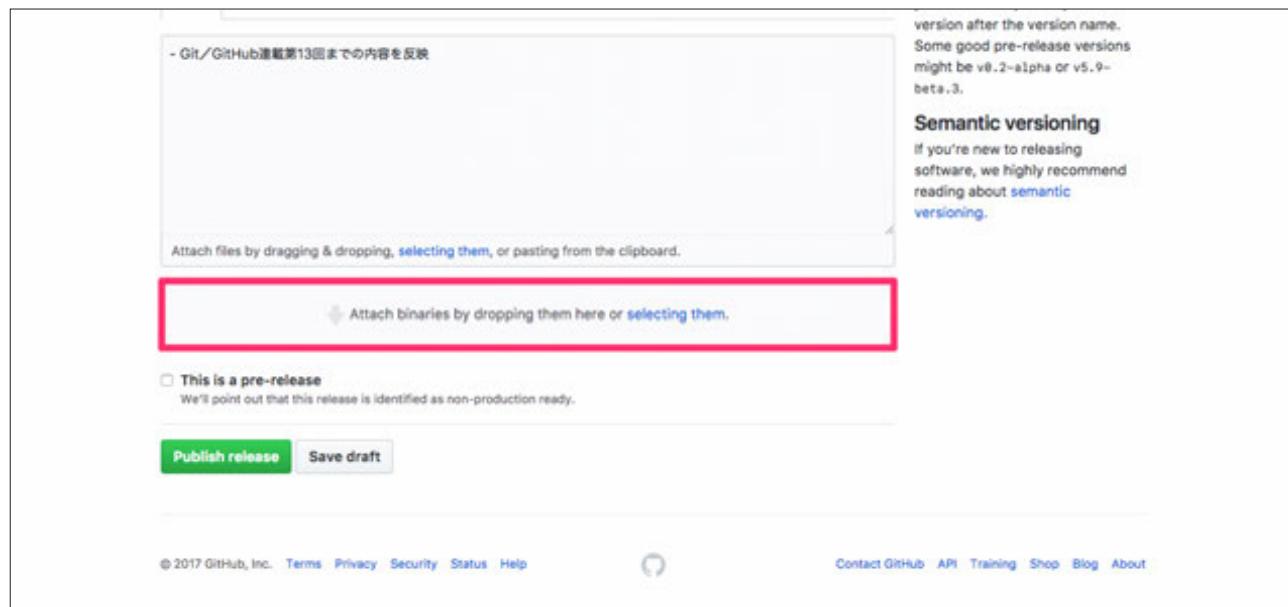


図 19 添付ファイル設定に関するエリア

今回はこのままリリース作成を実行します。「Publish release」をクリックします。

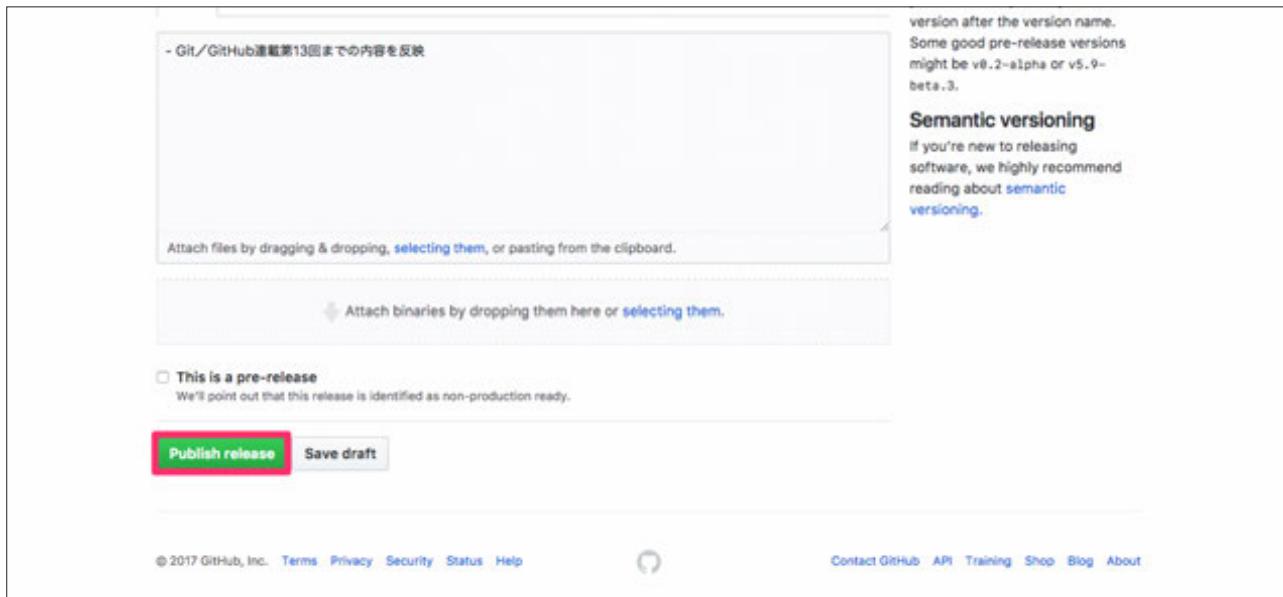


図 20 リリース作成を実行

作成が完了すると、作成したリリースのページが表示されます。作成時に設定した各値が適用されていることを確認できます。また、リポジトリ内のファイルをまとめた圧縮ファイル（拡張子 zip と tar.gz）が自動で作成されます。



図 21 リリースのページ

リリースを編集する

リリースの内容を編集するには、対象リリースのページで「Edit release」をクリックします。



図 22 リリースのページの「Edit release」ボタン

リリース編集ページが表示されます。リリースを新規追加したときと同じ要領で各値を編集し、「Update release」をクリックすると編集を実行できます。

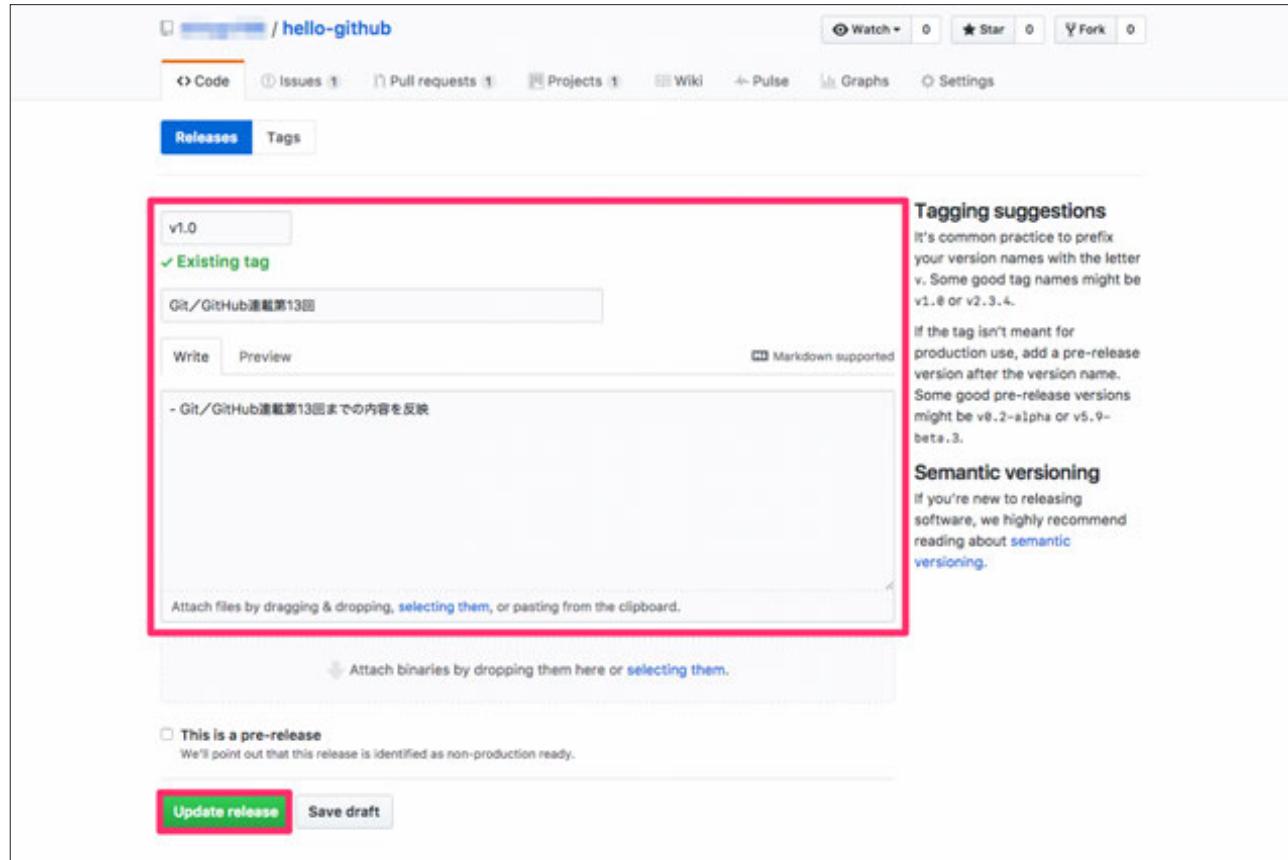


図 23 リリース編集ページ

リリースを削除する

リリースを削除するには、対象リリースのページで「Delete」をクリックします。



図 24 リリースのページの「Delete」ボタン

確認メッセージが表示されます。「Delete this release」をクリックすると、リリースの削除を実行できます。

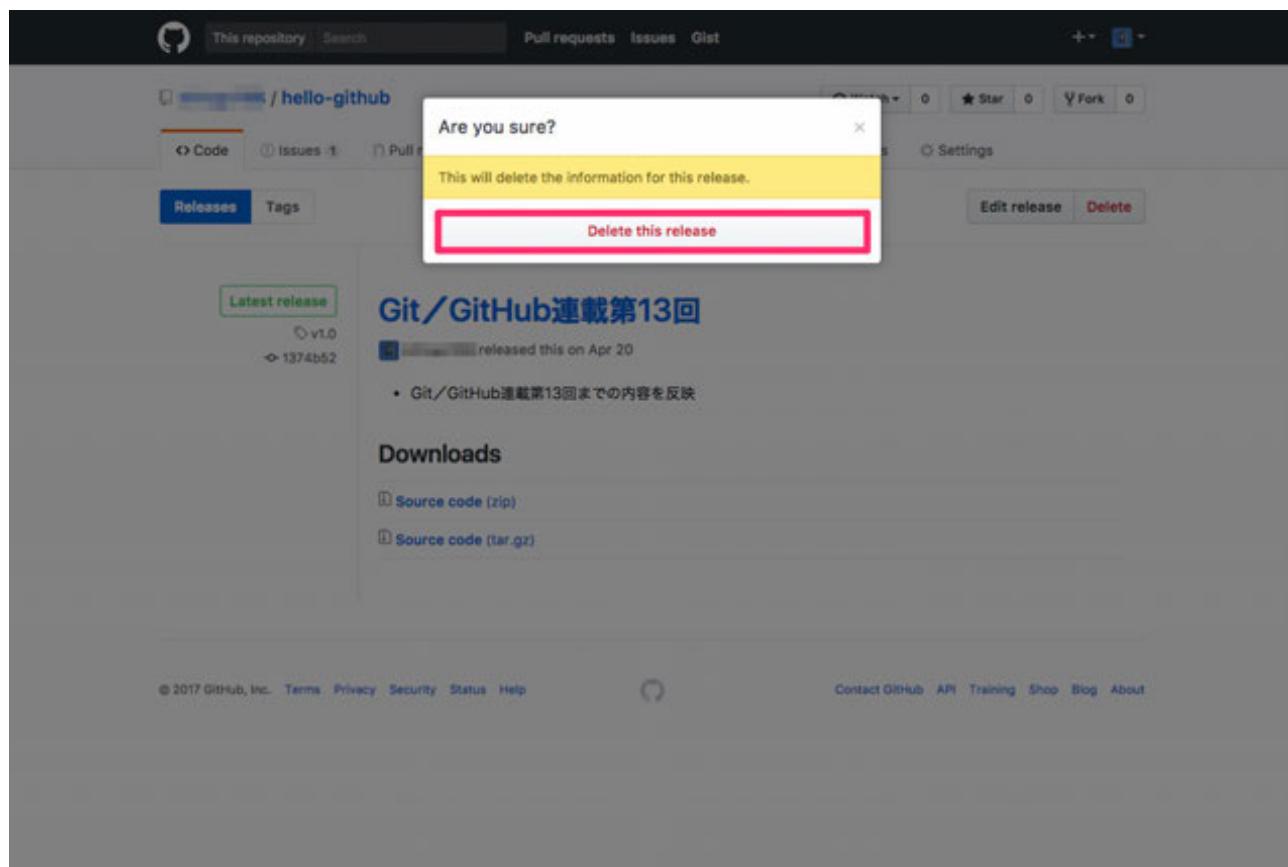


図 25 リリース削除確認メッセージ

ソフトウェアに関するドキュメントをホスティングする「Wiki」

Wiki とは、開発中のソフトウェアに関するドキュメントをホスティングするための機能です。例えば、下記のようなドキュメントをホスティングできます。

- ・ソフトウェアの使い方
- ・ソフトウェアの設計や意図などの説明

ソフトウェアに関するドキュメントには、本連載第 9 回で解説した「[README ファイル](#)」があります。 README ファイルは、ソフトウェアに関する情報を読者に素早く伝えるためのドキュメントです。一方、Wiki は長い文章を提供するのに適した機能です。

長めのドキュメントを提供したい場合は、概要を「README ファイル」に書き、詳細は Wiki で提供するといいでしょう。

Wiki のトップを表示する

Wiki に関する各種操作は、Wiki のトップから実行できます。

GitHub 上のリポジトリページを表示し「Wiki」タブを選択します。

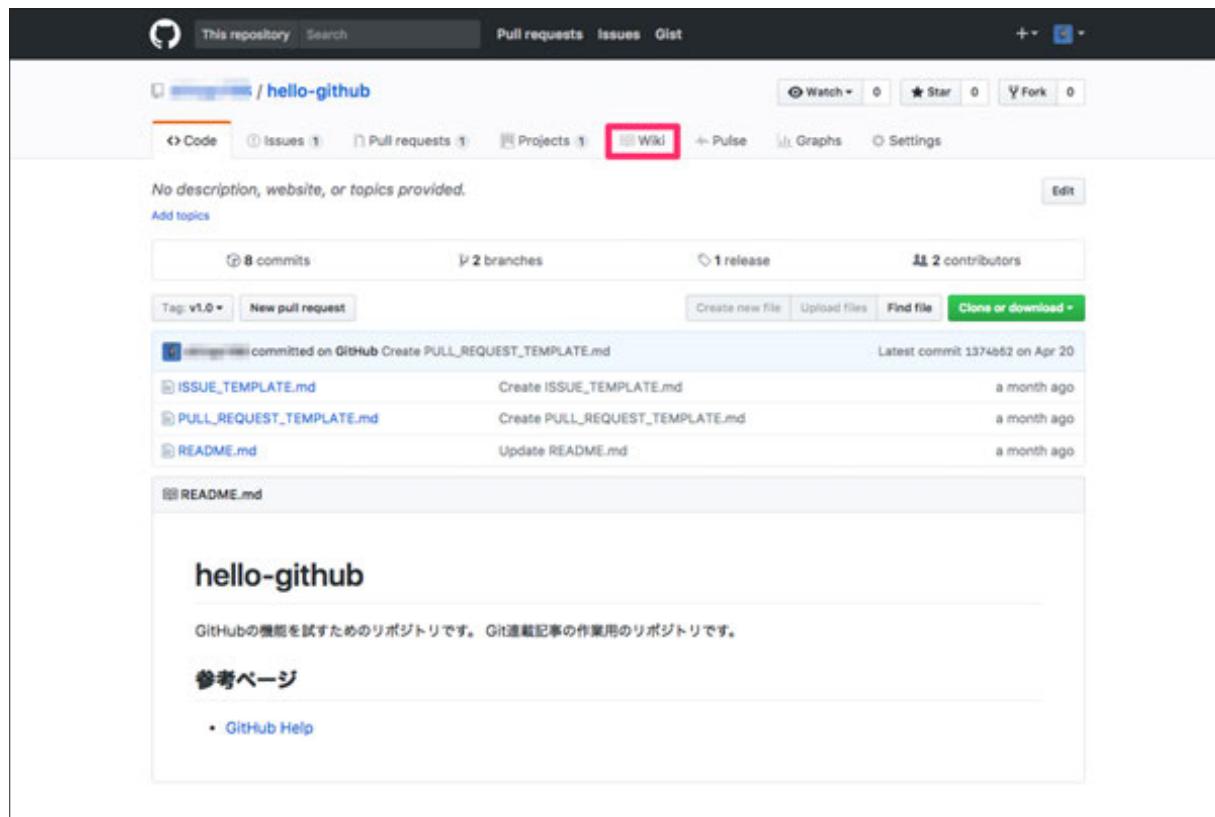


図 26 リポジトリトップページ

Wiki のトップを表示できました。Wiki を 1 つも作成していない場合、図 13 のような表示になります。

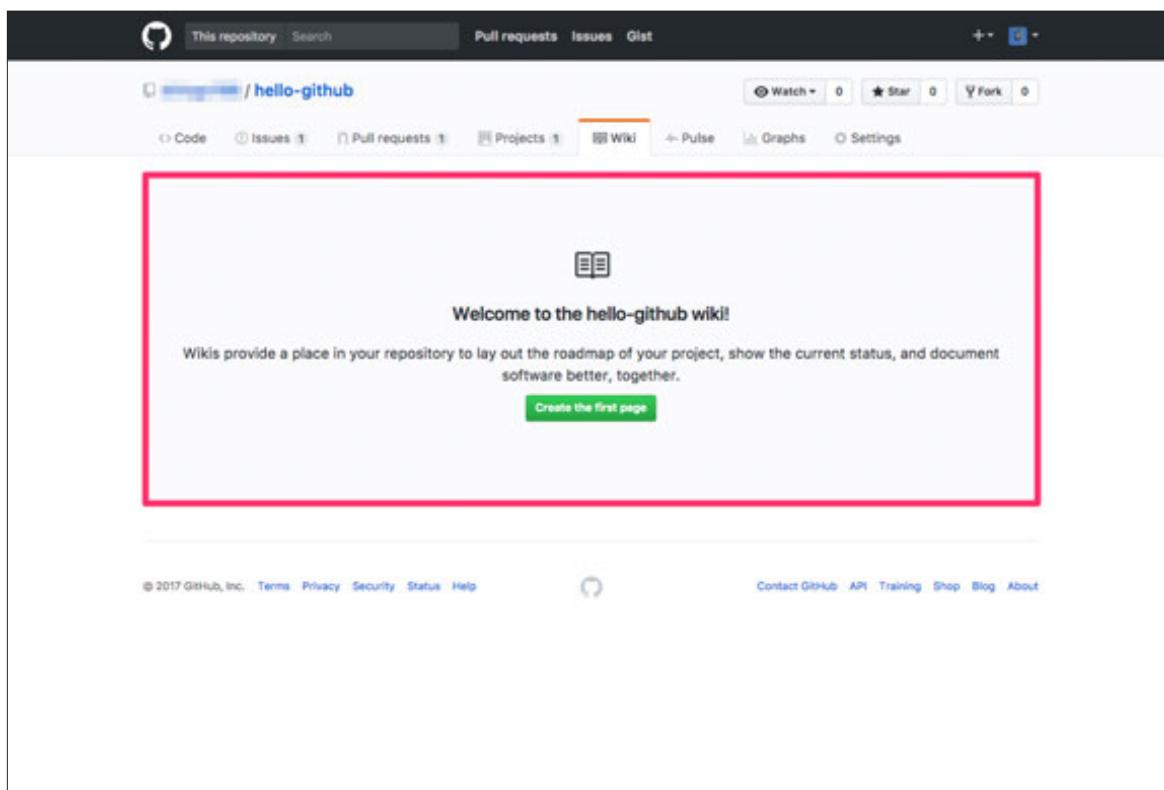


図 27 Wiki のトップ

1 つ目の Wiki ページを作成する

Wiki を作成するには「Create the first page」をクリックします。

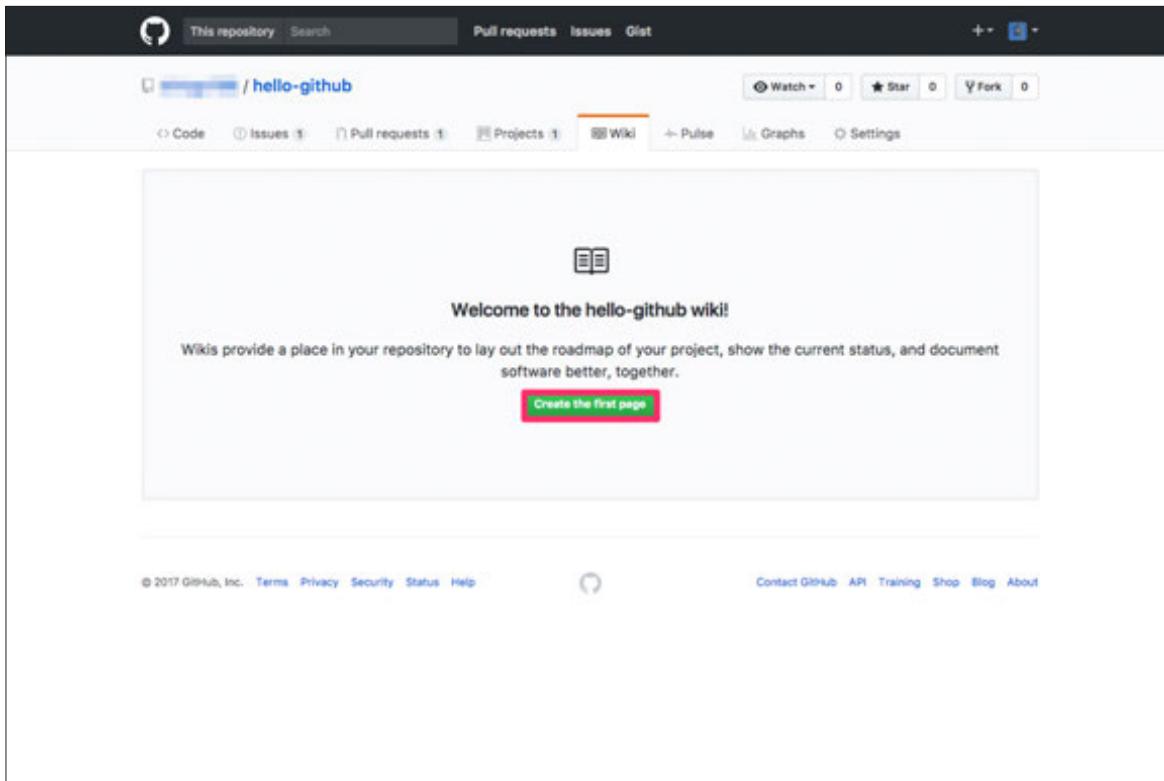


図 28 Wiki のトップの「Create the first page」ボタン

Wiki 作成ページが表示されます。一番上のテキストボックスにタイトルを、下半分のテキストボックスに本文を、一番下のテキストボックスに編集内容を記入できます。

イシューなどと同様に、本文には Markdown を使用できます。

Create new page

Home タイトル

Write Preview

h1 h2 h3 B I <> 三 三 “ ” Edit mode: Markdown

Block Elements Paragraphs & Breaks
Span Elements Headers
Miscellaneous Blockquotes
Lists
Code Blocks
Horizontal Rules

Welcome to the hello-github wiki!

本文

Edit Message Initial Home page 編集内容 Save Page

図 29 Wiki 作成ページ

タイトル用テキストボックスと本文用テキストボックスの間のエリアには、本文記入をサポートするためのタブやボタンがあります。これらのコンポーネントを使用すれば、プレビューの確認、Markdown の要素の挿入、文法に関する説明の確認などを行えます。

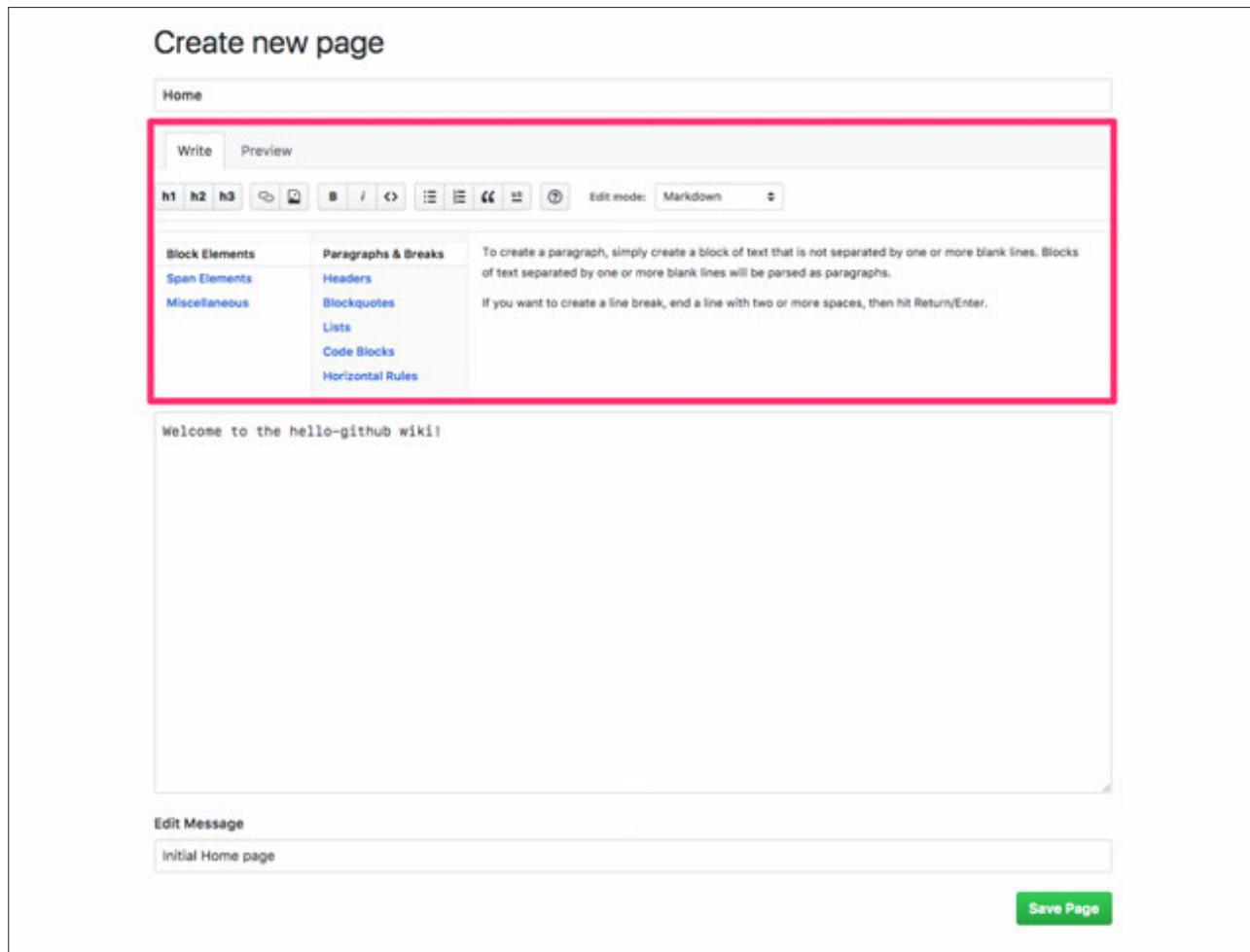


図 30 本文記入をサポートするためのエリア

一番下の「Save Page」をクリックすると、Wiki の作成を実行できます。ここでは、そのまま「Save Page」をクリックします。

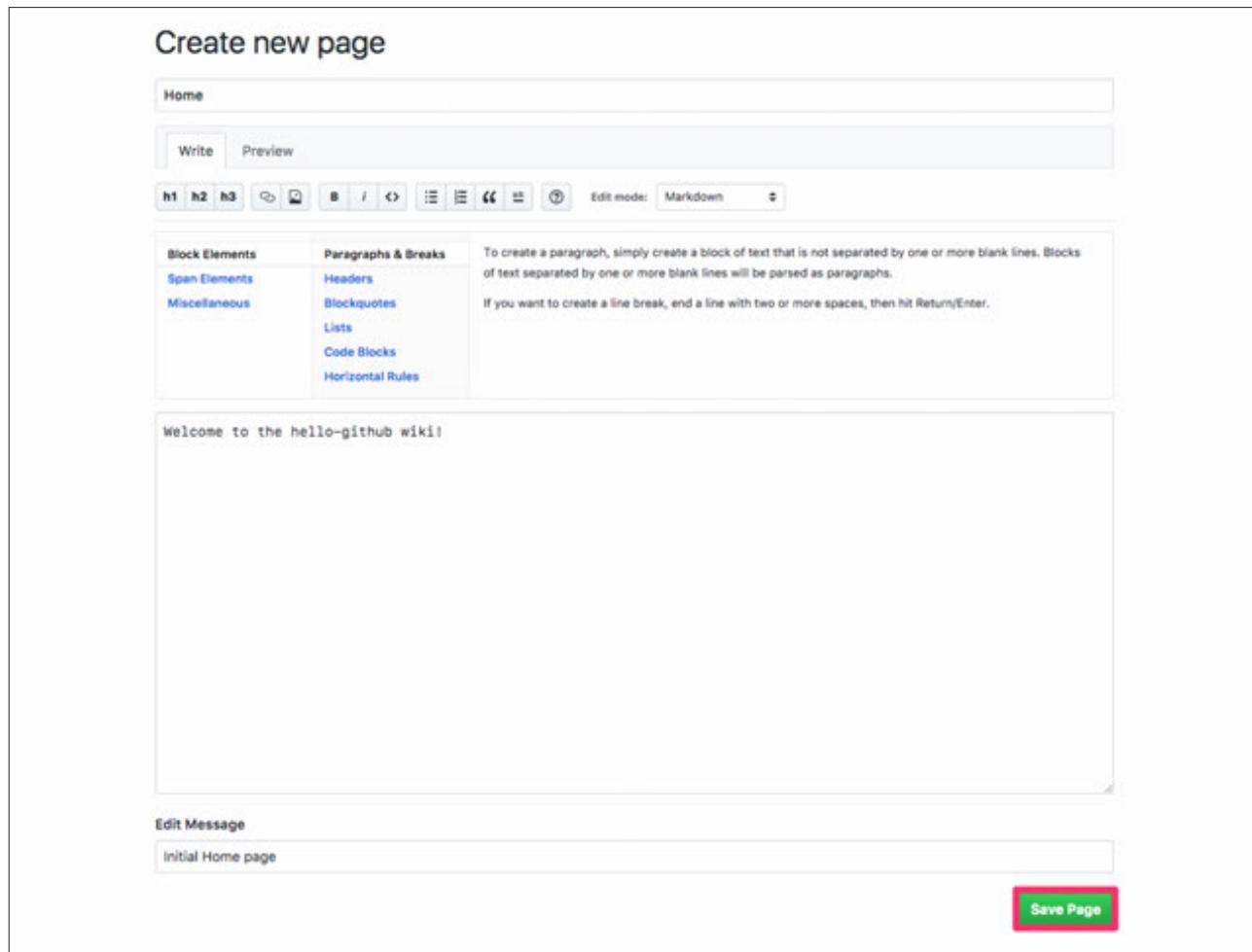


図 31 Wiki の作成を実行

Wiki 作成が完了すると、作成した Wiki のページが表示されます。

タイトルが「Home」のページは、Wiki のホームとなるページになります。リポジトリを見に来たユーザーがメインタブの中の「Wiki」を選択すると、まずこのページが表示されます。

Home ページには、Wiki に関して読者に伝えたいことなどを載せるといいでしよう。

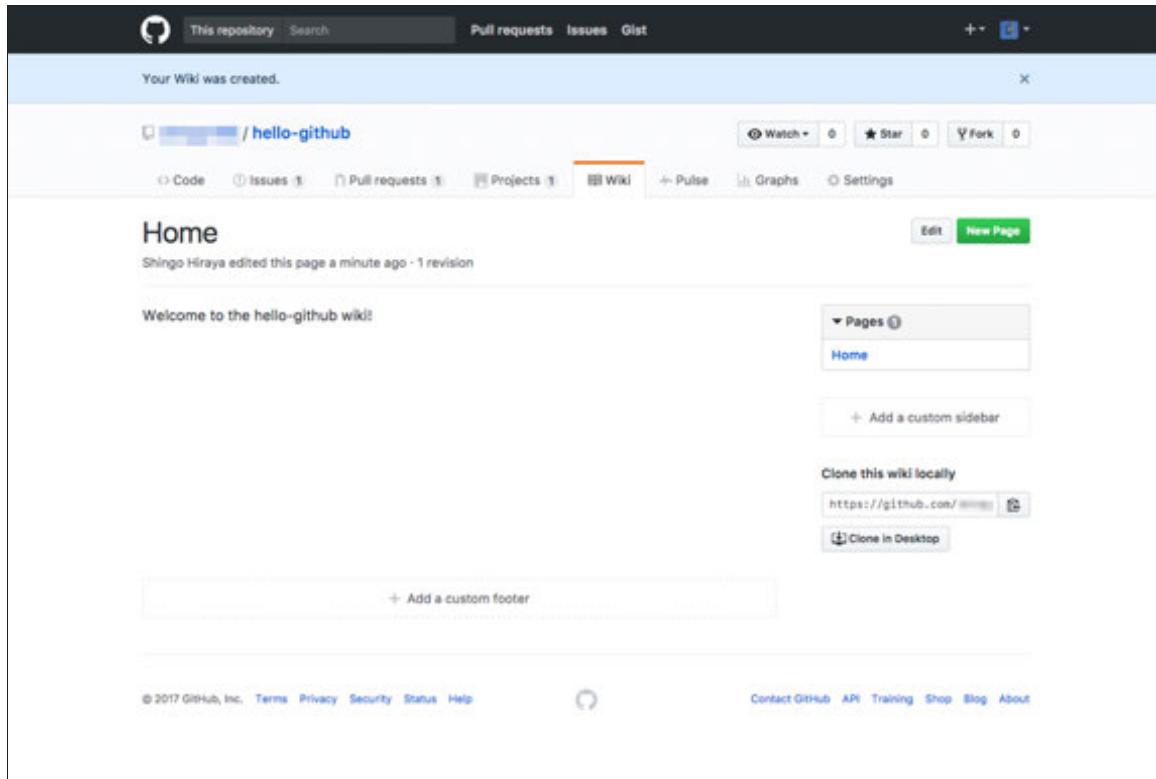


図 32 Home ページ

2つ目の Wiki ページを作成する

Wiki のホームとなるページが作成できたので、2つ目の Wiki ページを作成してみましょう。「New Page」をクリックします。

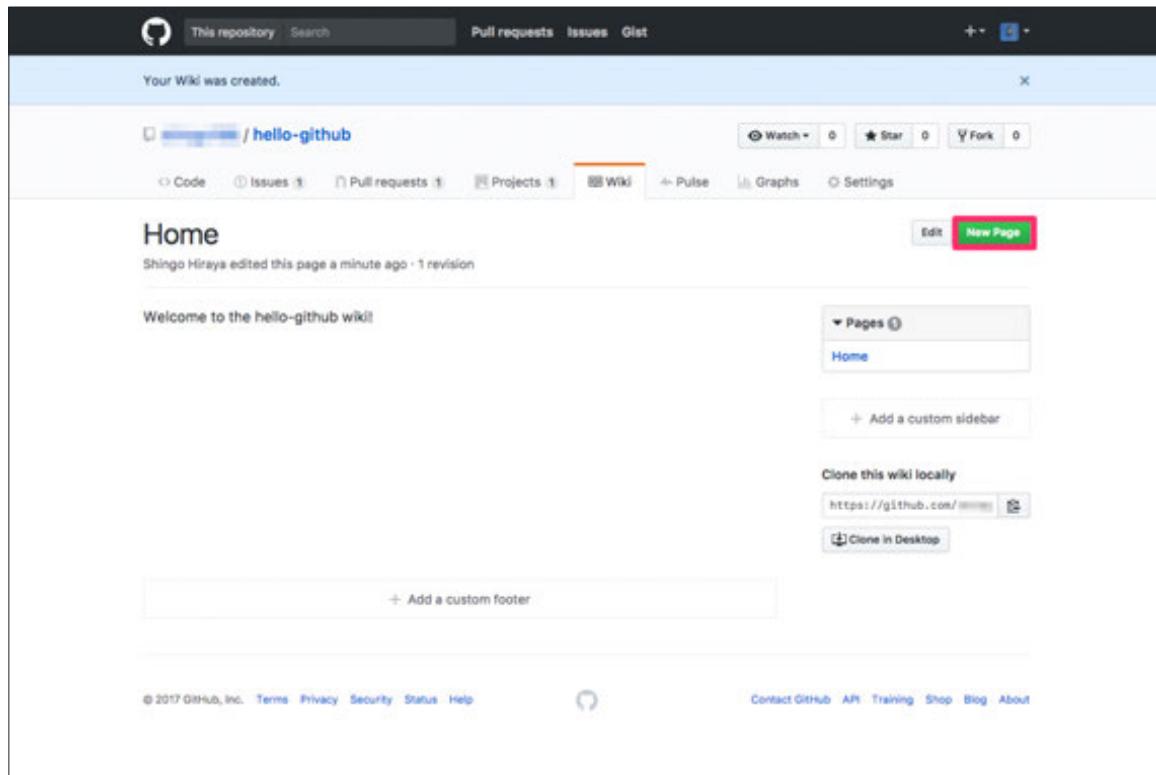


図 33 Home ページ上の「New Page」ボタン

1つ目の Wiki ページを作成したときと同じ要領で各値を記入します。

ここでは、「References」というタイトルで、「Wiki 機能に関するドキュメントのリンク集ページ」を作成してみます。

「Save Page」をクリックして追加を実行します。

The screenshot shows the 'Create new page' interface for GitHub Wikis. The title 'References' is highlighted with a red box. Below it, the 'Write' tab is selected. The main content area contains a list of GitHub documentation links under the heading '# Wiki機能に関するドキュメント'. A second red box highlights this list. At the bottom right of the content area is a green 'Save Page' button.

Wiki機能に関するドキュメント

- [About GitHub Wikis - User Documentation](https://help.github.com/articles/about-github-wikis/)
- [Adding wiki pages via the online interface - User Documentation](https://help.github.com/articles/adding-wiki-pages-via-the-online-interface/)
- [Editing wiki pages via the online interface - User Documentation](https://help.github.com/articles/editing-wiki-pages-via-the-online-interface/)
- [Viewing a wiki's history of changes - User Documentation](https://help.github.com/articles/viewing-a-wiki-s-history-of-changes/)

図 34 Wiki 作成ページ

Wiki 作成が完了すると、作成した Wiki のページが表示されます。

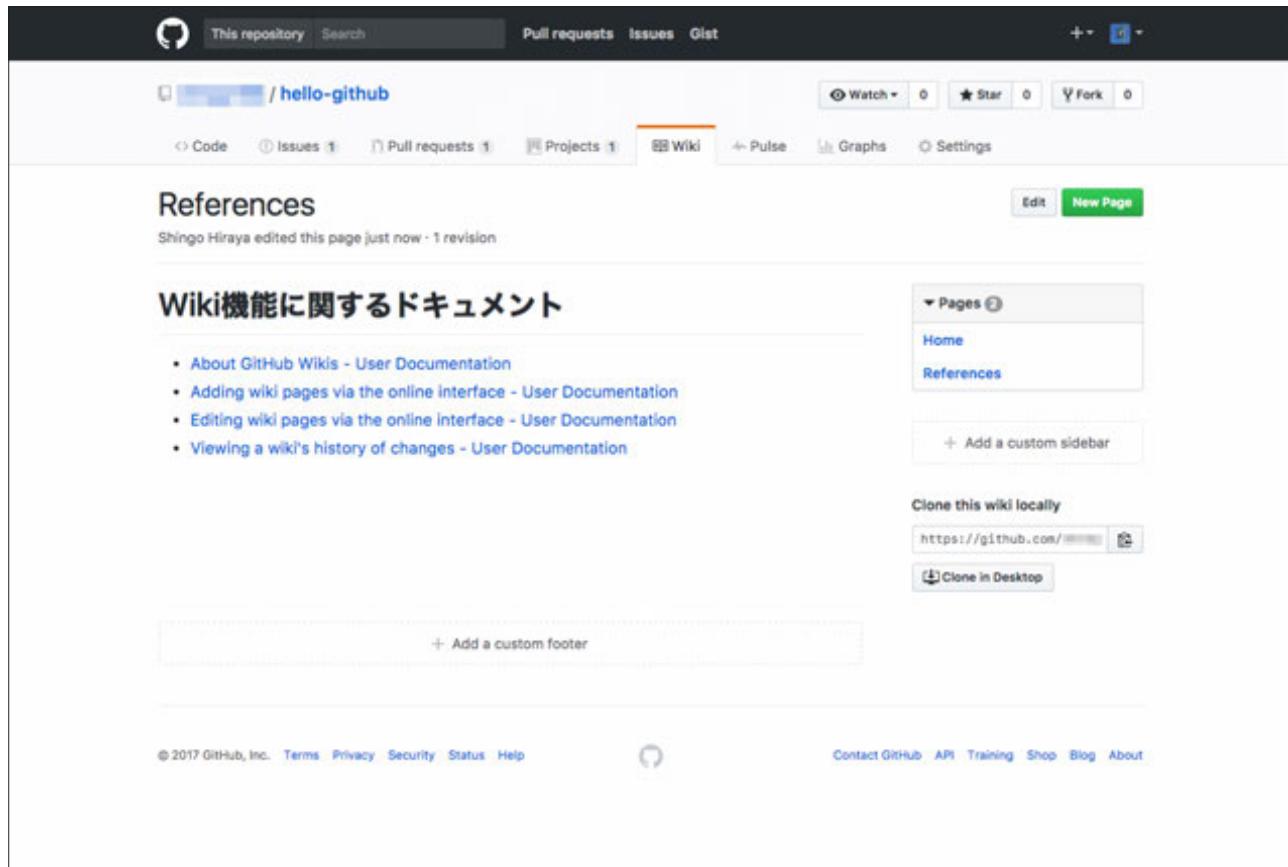


図 35 References ページ

Wiki ページを編集する

2つ目の Wiki ページが作成できましたので、次は「2つ目の Wiki ページへのリンク」を Home に追加してみましょう。

メインタブの中の「Wiki」（または右側の Pages エリア内の「Home」）をクリックして Home に戻ります。

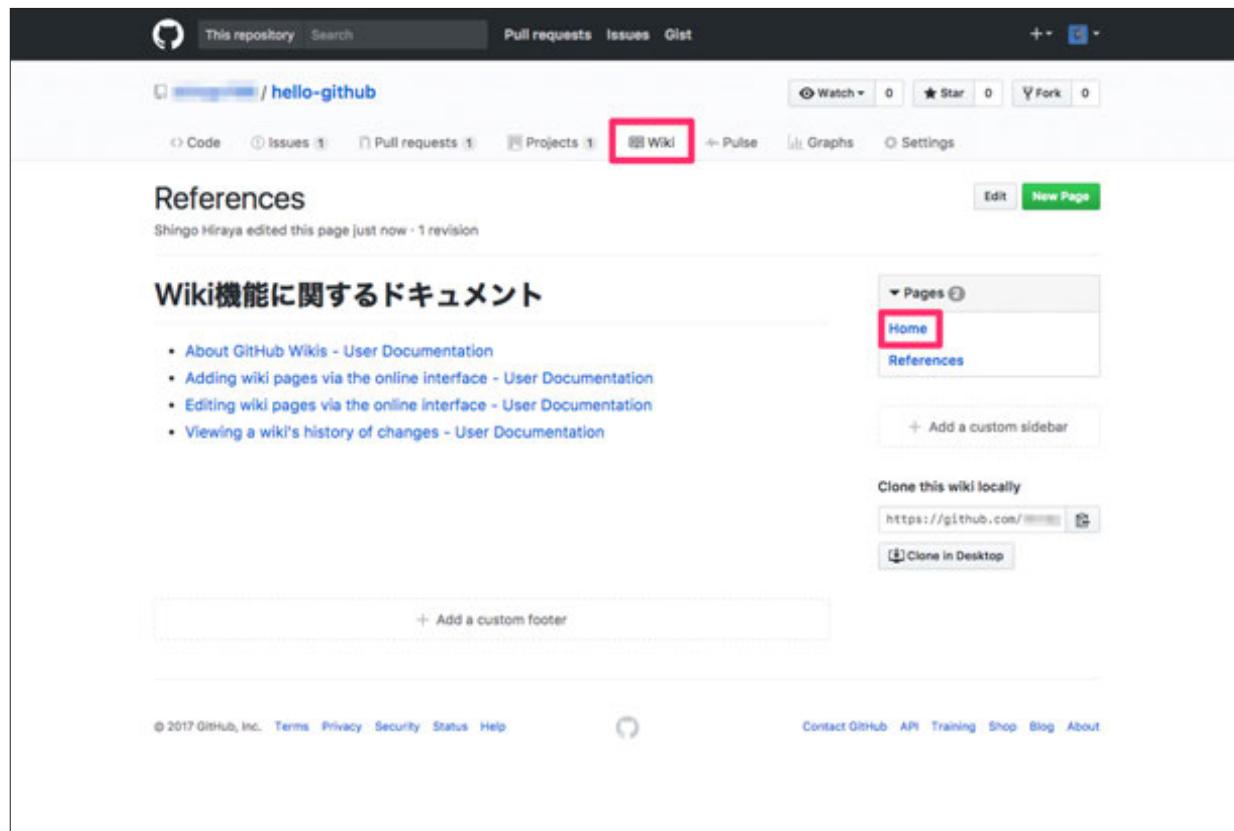


図 36 Home ページへ移動

ページを編集するには「Edit」をクリックします。

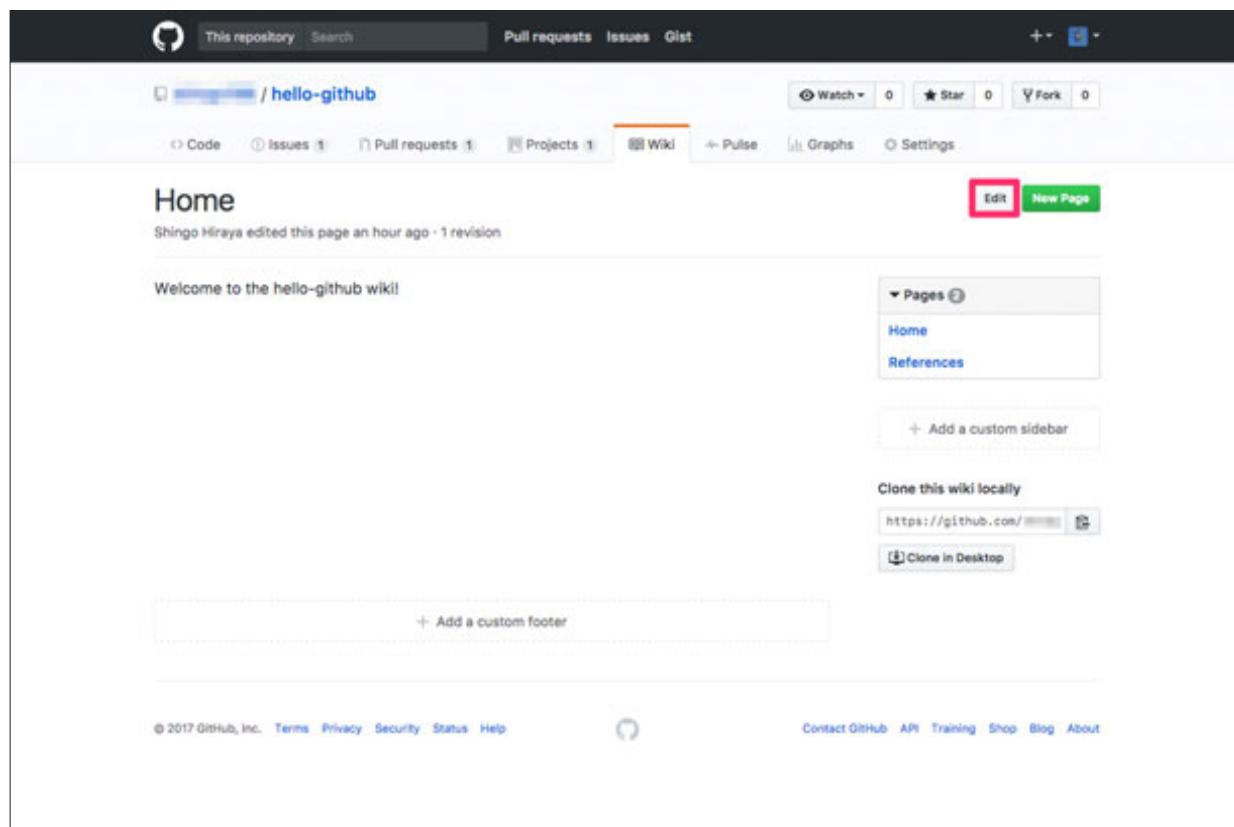


図 37 Home ページ上の「Edit」ボタン

編集ページが表示されます。本文に「2つ目の Wiki ページへのリンク」を追加し、「Save Page」をクリックします。

The screenshot shows the GitHub Wiki interface for the 'Editing Home' page. At the top, there are tabs for 'Page History' and 'New Page'. Below the tabs, there are 'Write' and 'Preview' buttons. The 'Write' button is selected. There is also a toolbar with buttons for h1, h2, h3, bold, italic, etc. The 'Edit mode' is set to 'Markdown'. On the left, there is a sidebar with sections for 'Block Elements', 'Span Elements', 'Miscellaneous', 'Paragraphs & Breaks', 'Headers', 'Blockquotes', 'Lists', 'Code Blocks', and 'Horizontal Rules'. In the main content area, there is a welcome message and a bulleted list. One bullet point has a red arrow labeled ① pointing to it. At the bottom right, there is an 'Edit Message' section and a green 'Save Page' button with a red arrow labeled ② pointing to it.

図 38 Wiki 編集ページ

Wiki 編集が完了すると、編集した Wiki のページが表示されます。編集した内容が反映されていることを確認できます。

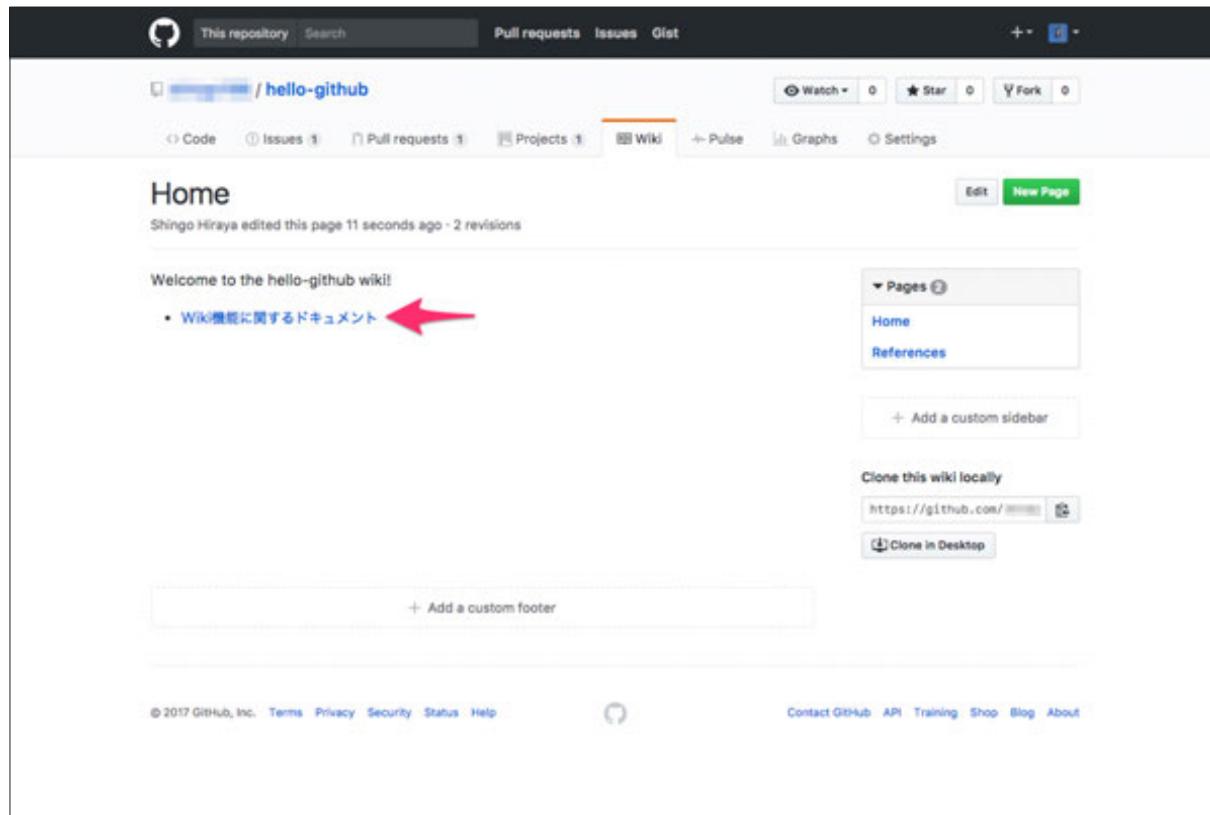


図 39 編集後の Home ページ

Wiki ページの編集履歴を確認する

Wiki ページの編集履歴を確認するには、Wiki ページのタイトルの下にあるリンクをクリックします。

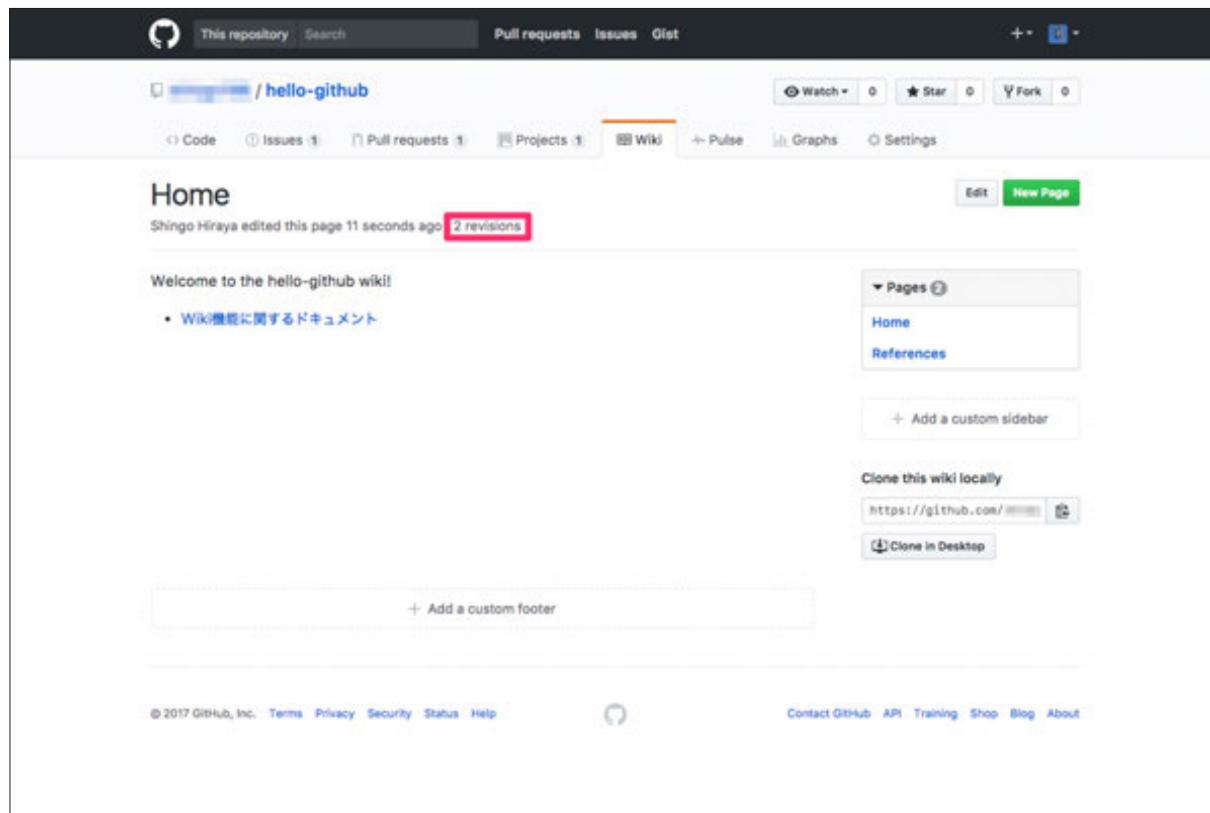


図 40 Home ページ上のリンク

Wiki の編集履歴ページが表示されます。Wiki の各ページのファイルは Git で管理されており、保存を実行するたびにコミットが行われます。

今回 Home を 2 回保存したので、2 つの履歴が表示されています。

Revisions			
<input type="checkbox"/>	Updated Home (markdown)	9 minutes ago	b5bcc68
<input type="checkbox"/>	Initial Home page	an hour ago	841f986

図 41 Wiki の編集履歴ページ

2 つの履歴を選択して「Compare Revisions」をクリックすると、差分を表示できます。

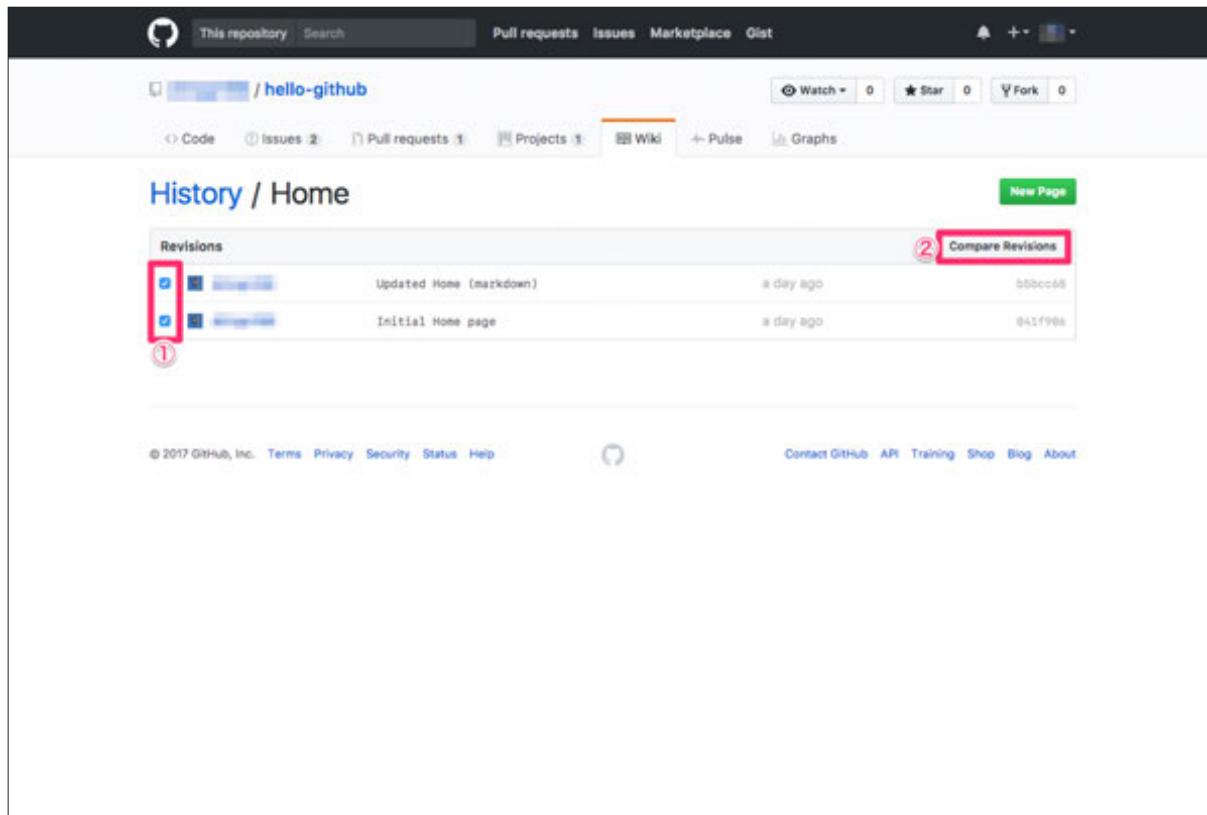


図 42 Wiki の編集履歴ページ上の「Compare Revisions」ボタン

差分は図 43 のように表示されます。本連載第 10 回で紹介した「[プルリクエストのページの Files changed タブ](#)」と同様の表示で差分を確認できます。

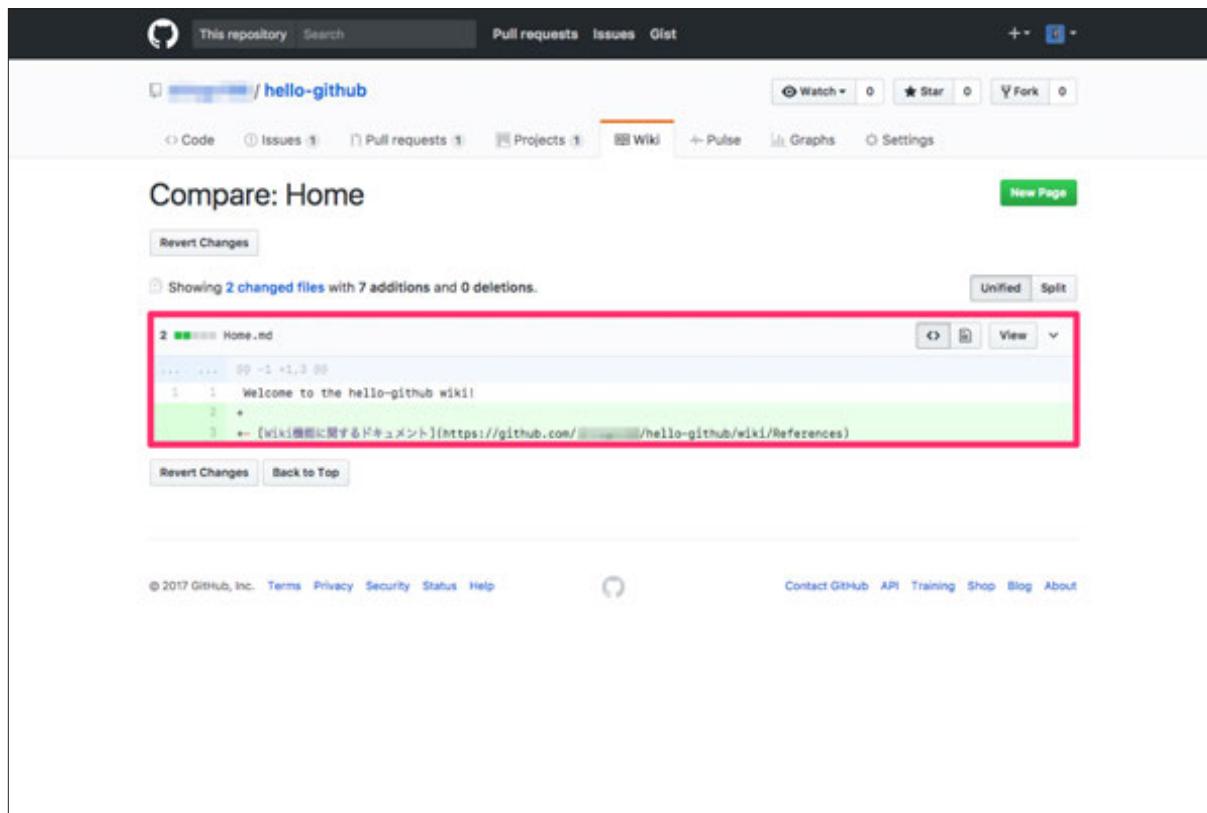


図 43 Wiki の差分表示

Wiki ページを削除する

Wiki の特定のページを削除するには、編集ページを開き「Delete Page」をクリックします。

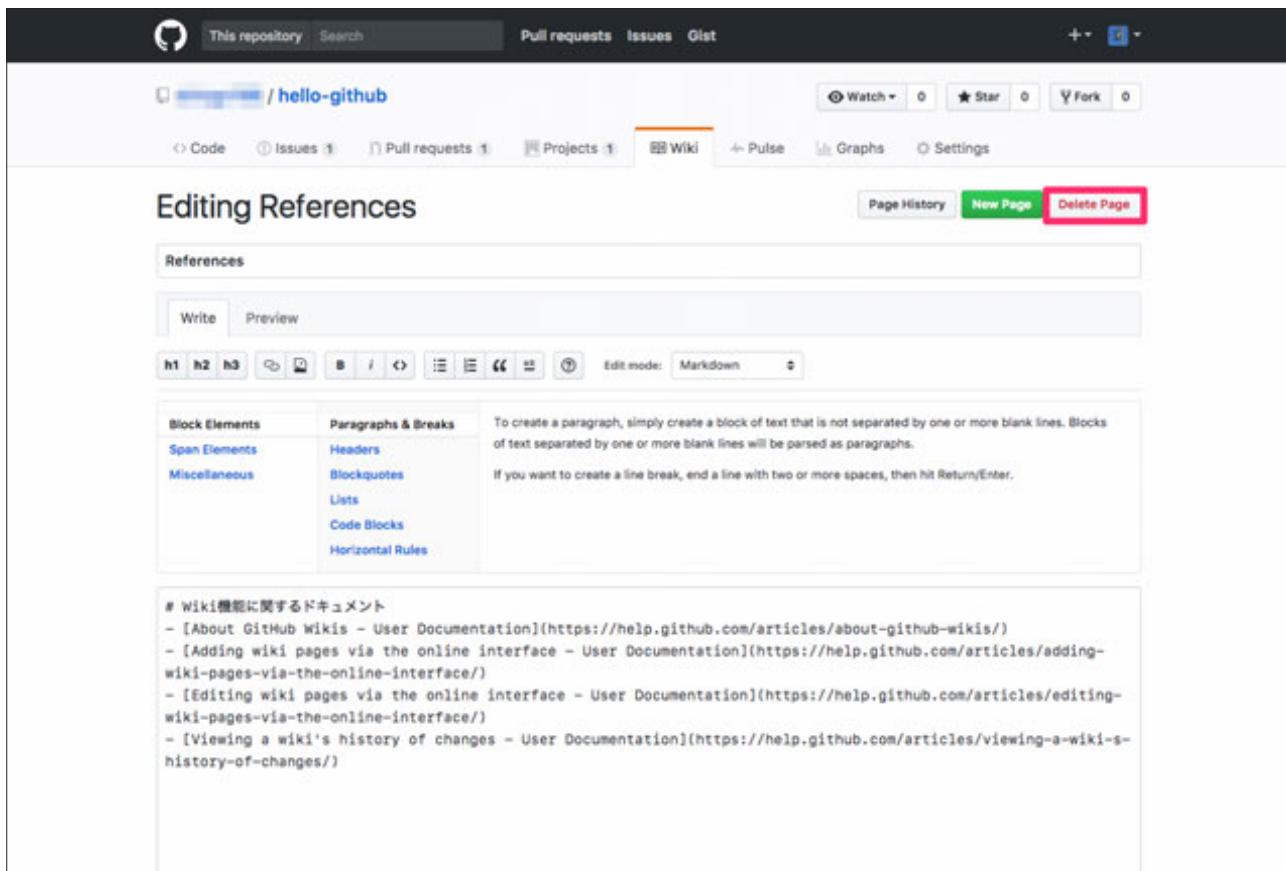


図 44 Wiki 編集ページ上の「Delete Page」ボタン

「Delete Page」をクリックすると、ブラウザの確認ダイアログが表示されます。「OK」をクリックすると、ページの削除を実行できます。

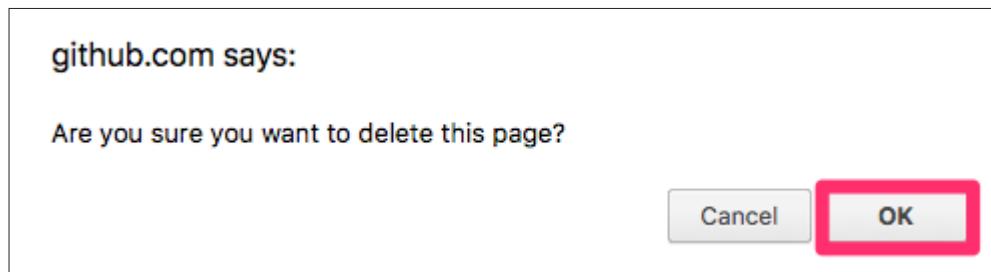


図 45 ページ削除確認ダイアログ

Slack との連携を例に見る、GitHub の外部サービス連携機能

GitHub はさまざまな外部サービスと連携できます。GitHub 上でのイベントをトリガーに外部サービスの機能を利用したり、外部サービス側から GitHub の機能を利用したりすることができます。

本稿では、コミュニケーションツール「Slack」と GitHub を連携する方法を解説していきます。

Slack と GitHub を連携させると、下記のようなイベントが発生したタイミングでメッセージを Slack に投稿することができます。

- コードがプッシュされた
- イssueにコメントが付いた
- プルリクエストが作成された

ここからは、Slack と GitHub を連携させる手順を解説していきます。Slack 側での操作がメインになりますが、外部サービス連携の大体のイメージをつかんでいただければ問題ありません。

対象チームの Slack に GitHub 連携をインストールする

GitHub 連携を行う対象のチームの Slack を開き、メニューの「Apps & Integrations」をクリックします。

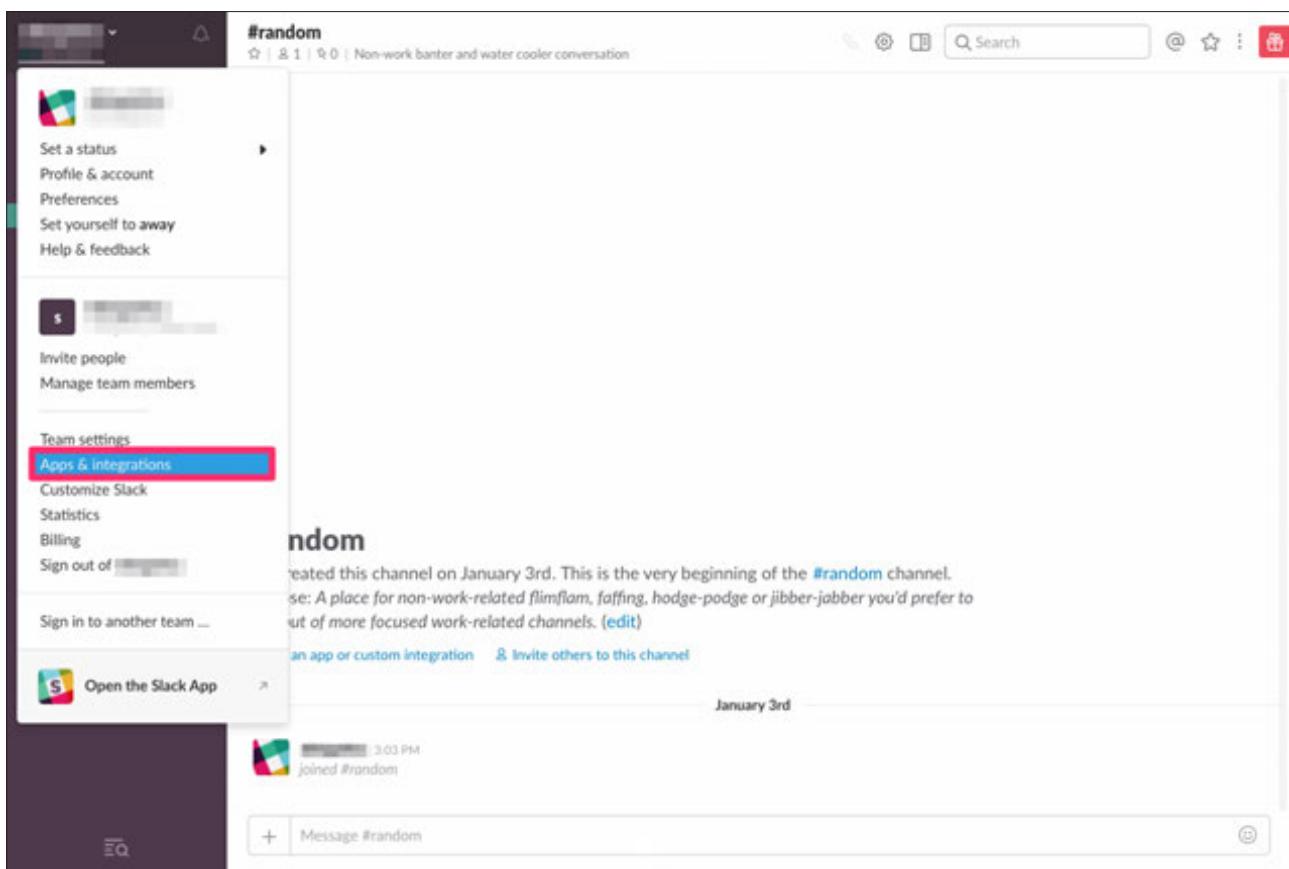


図 46 Slack のメニュー上の「Apps & Integrations」

外部サービスとの連携に関するページが表示されるので、テキストボックスに「github」と入力し、検索結果の「GitHub」をクリックします。

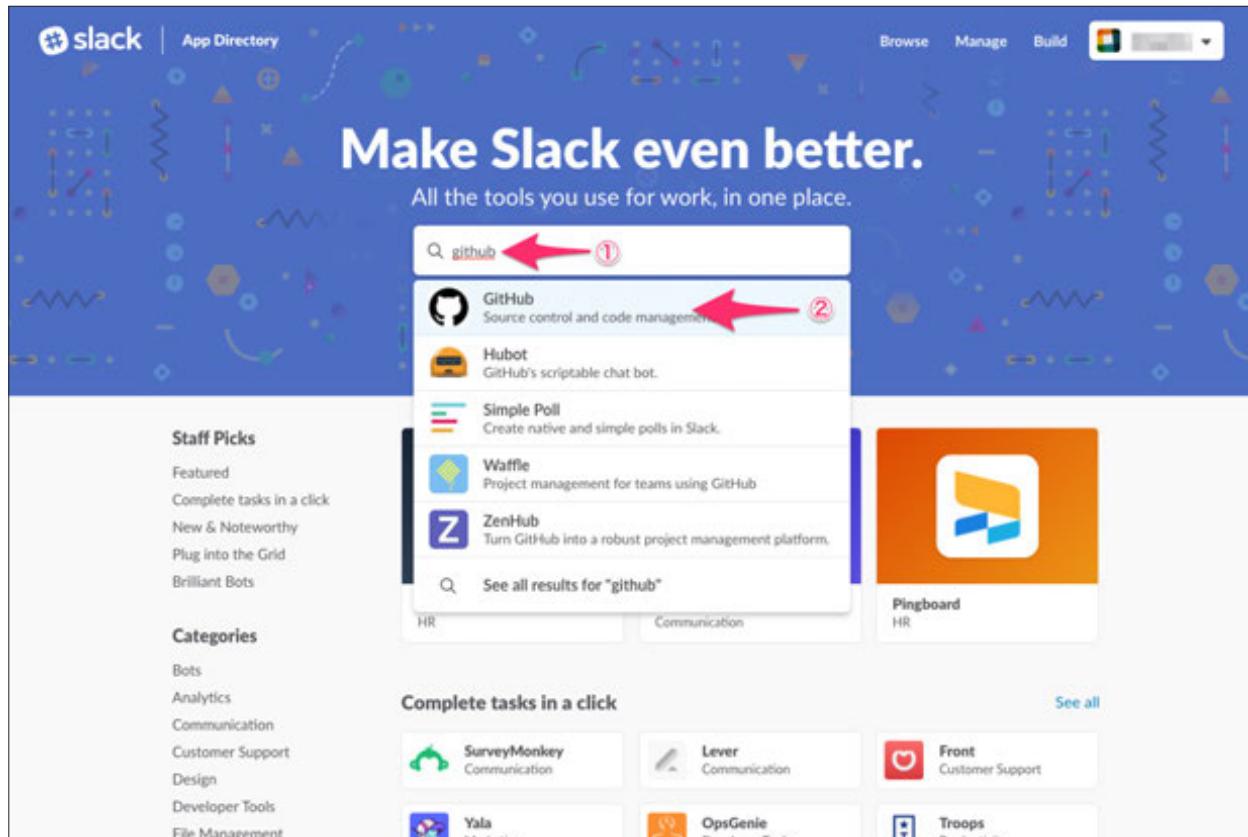


図 47 「GitHub」を検索

GitHubに関するページが表示されるので、「Install」をクリックします。

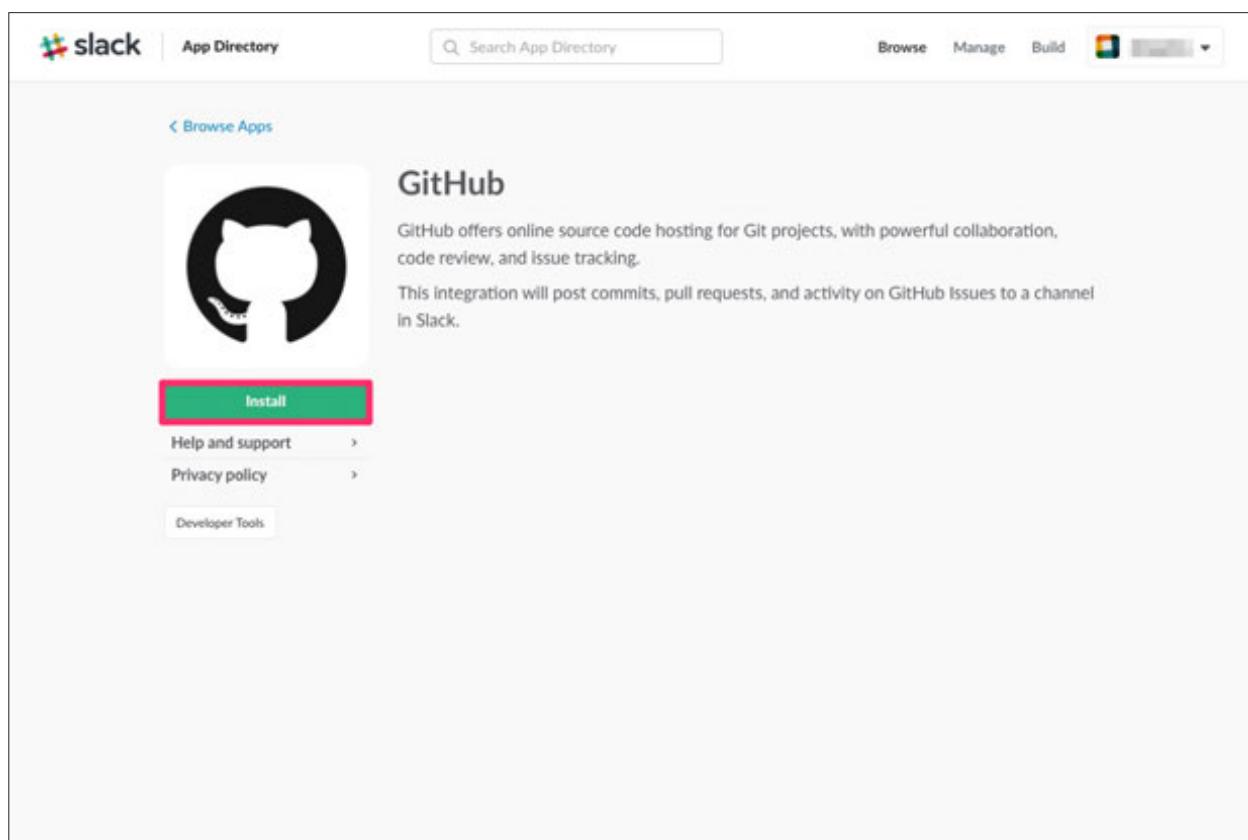


図 48 GitHub 連携のインストール

GitHub 上でのイベントを投稿したいチャンネルを選択し、「Add GitHub Integration」をクリックします。

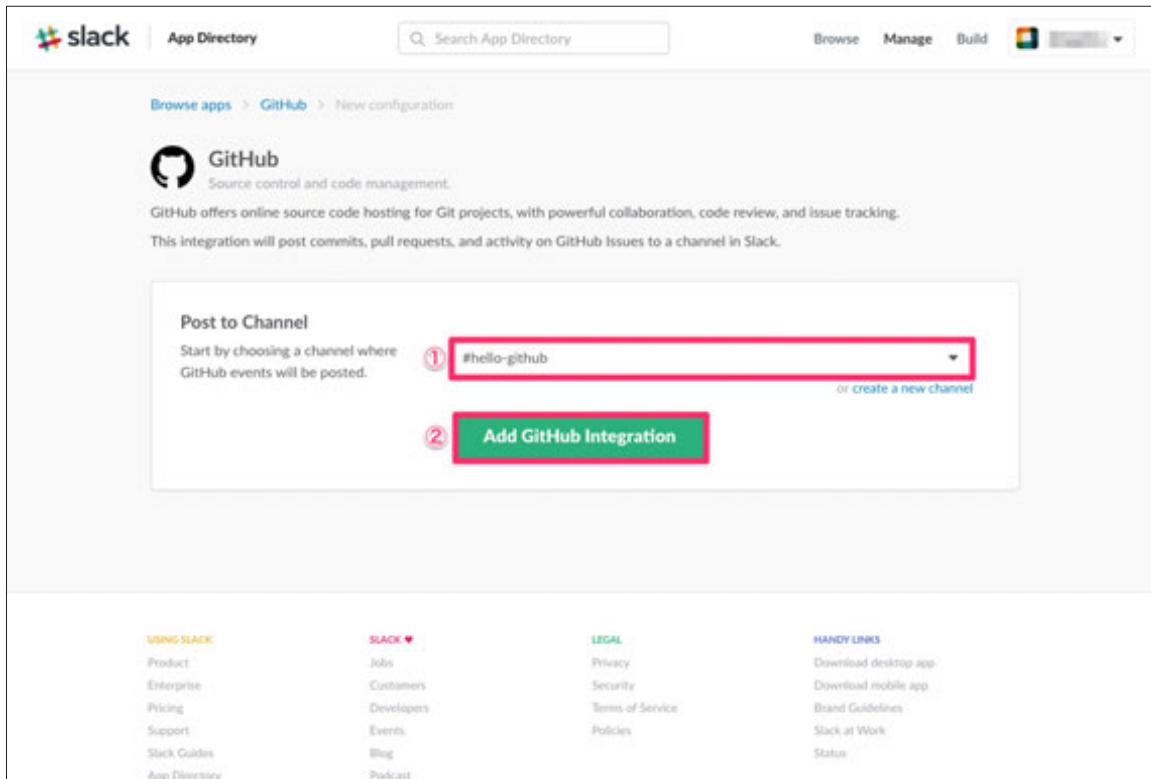


図 49 チャンネル選択

GitHub アカウントとの連携を実行する

「Authentication your GitHub account」をクリックして、GitHub アカウントとの連携に進みます。

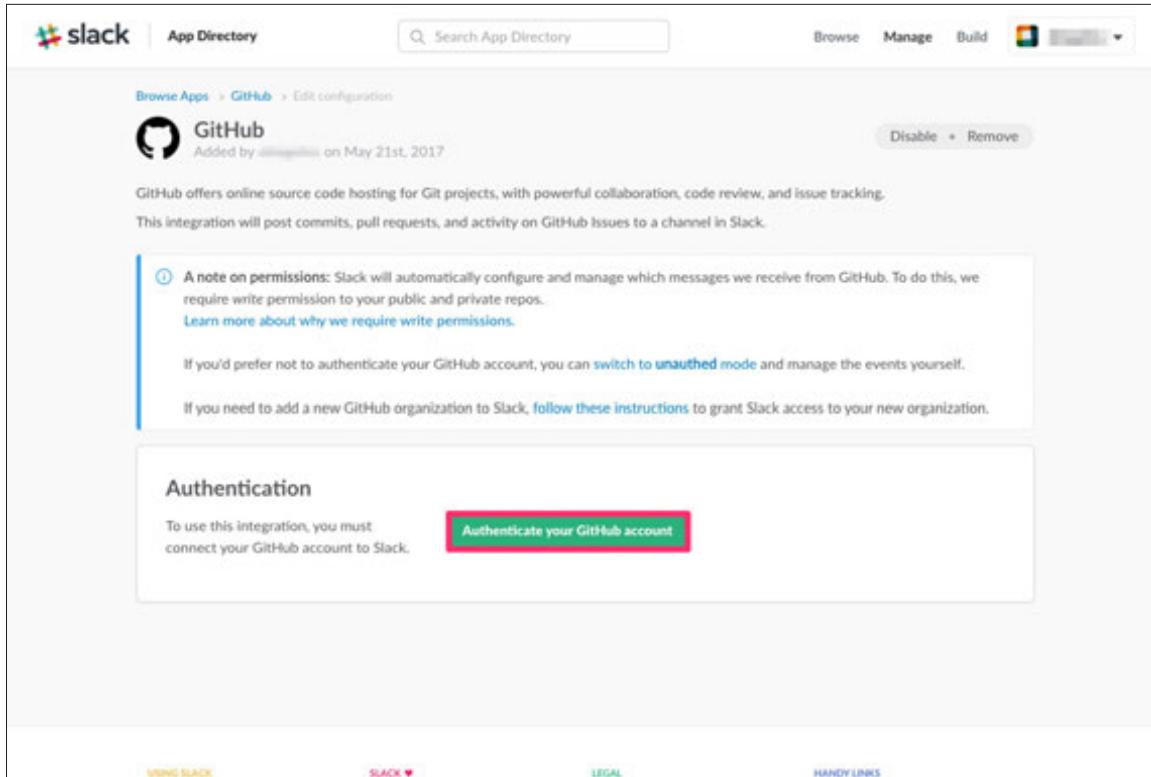


図 50 GitHub アカウントとの連携を実行

Slack がリポジトリへのアクセスを求めている旨のページが表示されます。

内容に問題がなければ、「Authorize slackplat」をクリックします。

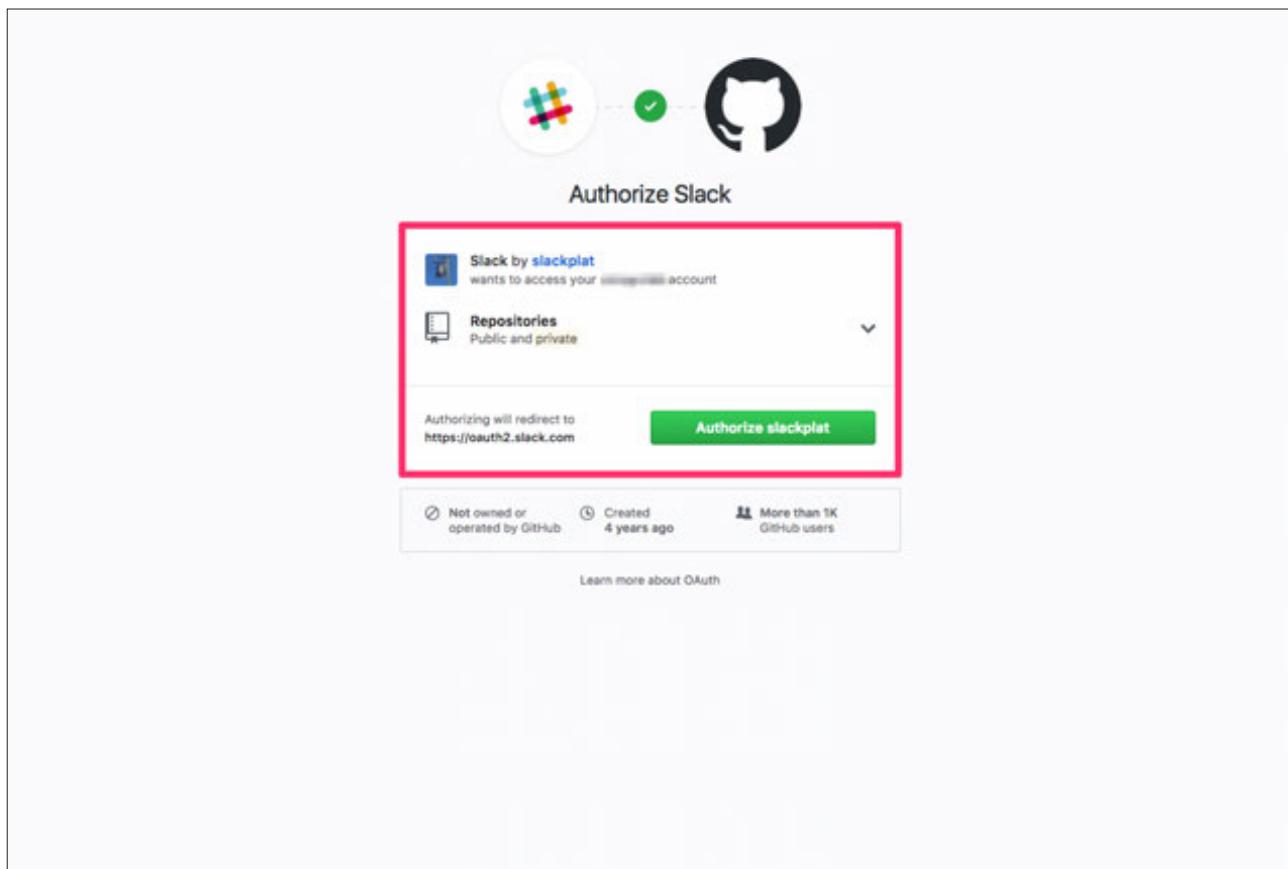


図 51 アクセス権限の確認

パスワードの確認が求められたら、GitHub アカウントのパスワードを入力し、「Confirm password」をクリックします。

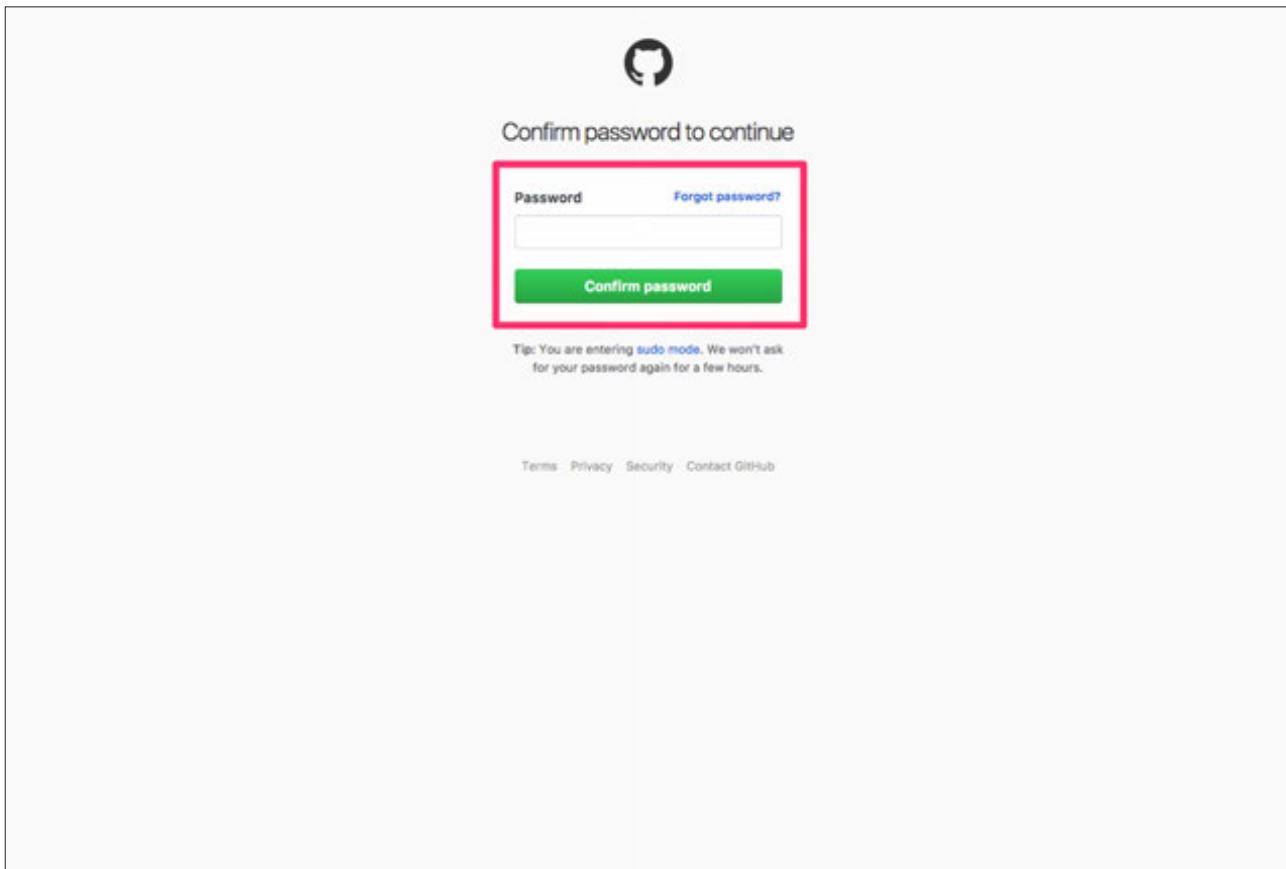


図 52 パスワードの確認

連携の内容を設定する

GitHub アカウントとの連携処理が完了すると、連携の内容を設定するページが表示されます。

Repositories エリアのコンポーネントをクリックして、対象リポジトリを選択します。

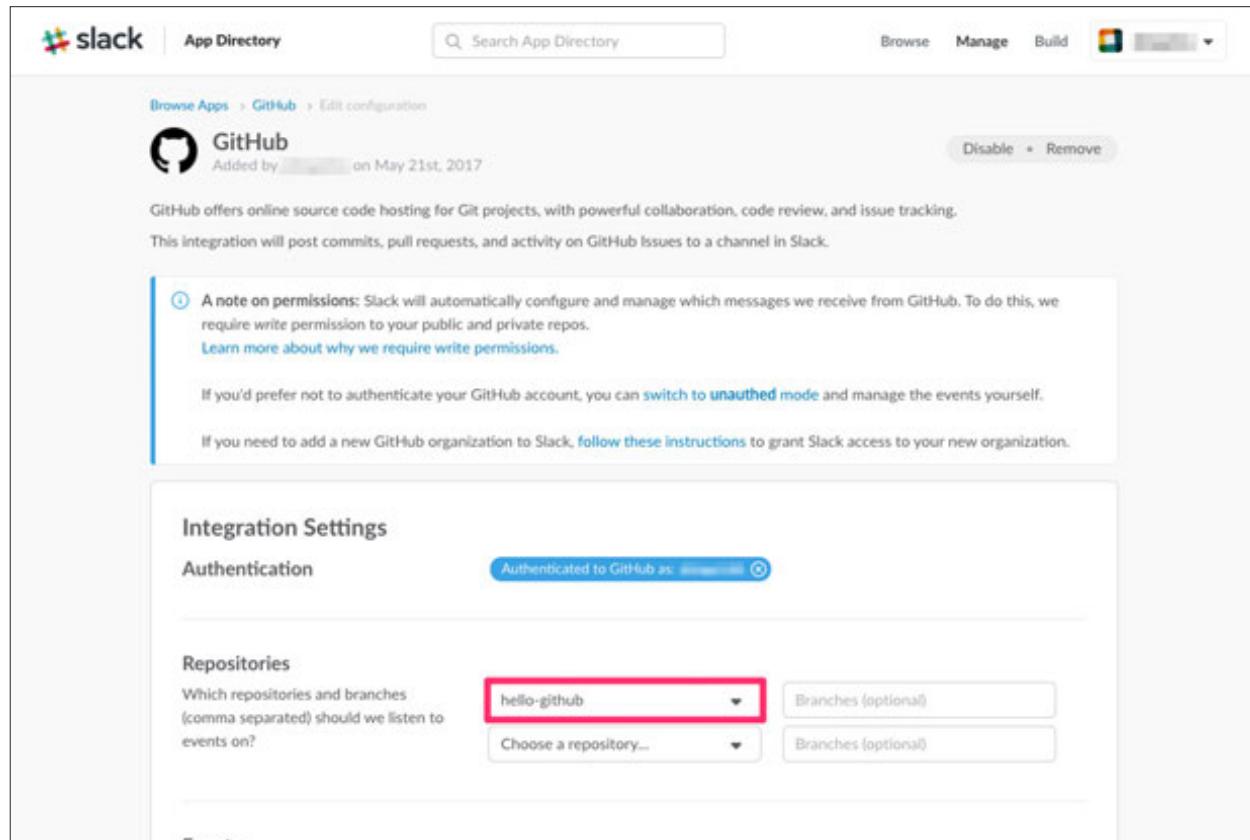


図 53 対象リポジトリを選択

Events エリアでは、どのイベントを Slack に投稿するかを選択できます。

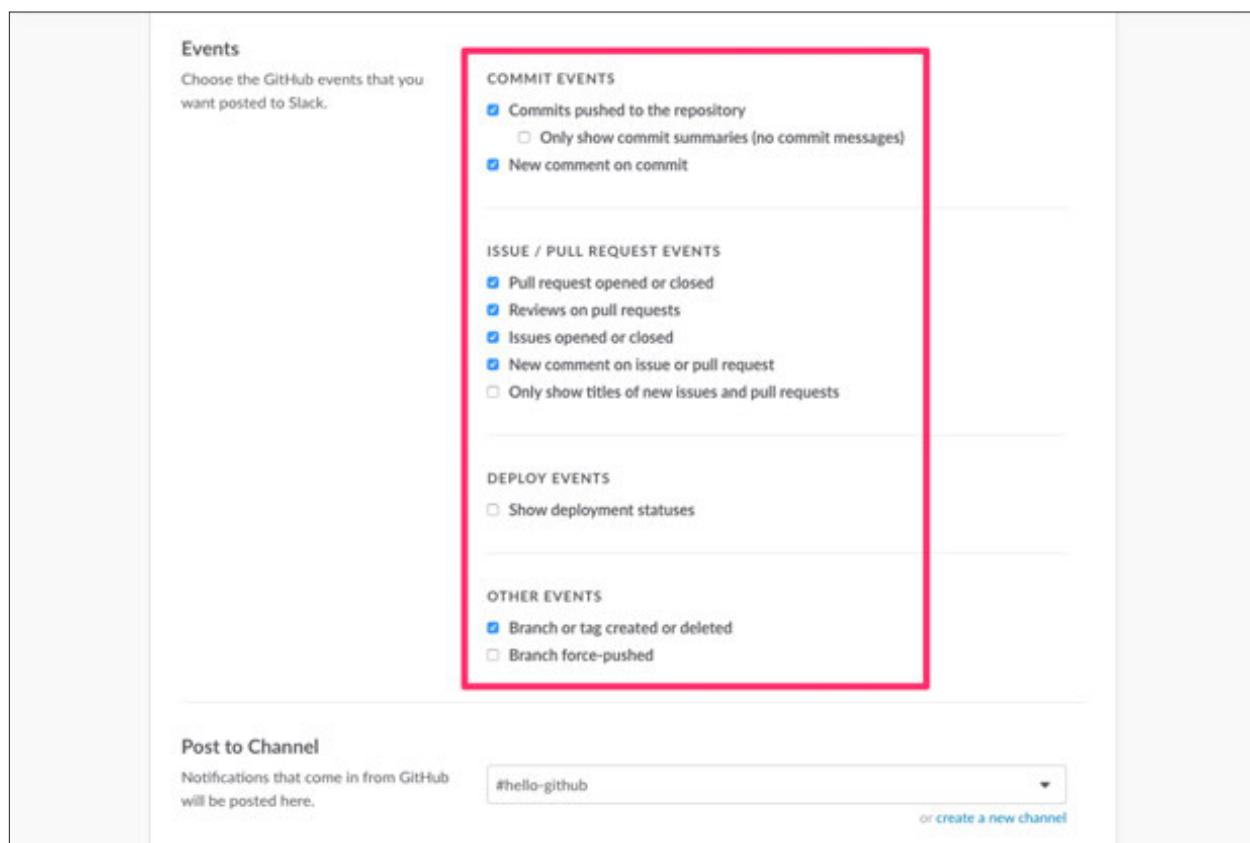


図 54 投稿させるイベントを選択

さらにその下のエリアでは、投稿先のチャンネルや投稿時のアイコンなどを設定できます。

今回はそのまま「Save Integration」をクリックして、設定を実行します。

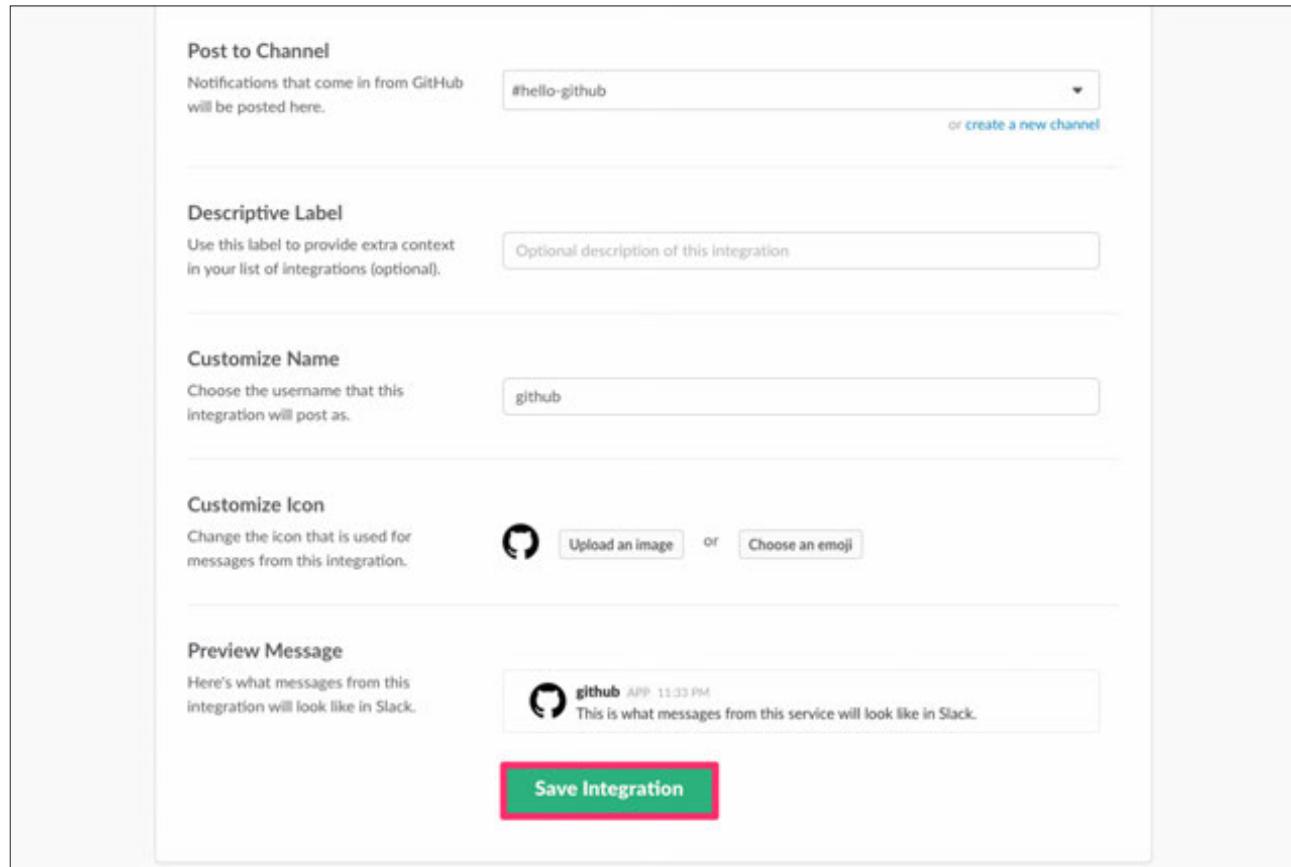


図 55 設定を実行

連携の動作を確認する

GitHub でイシューを作成すると、イシューが作成された旨のメッセージが Slack に投稿されました。

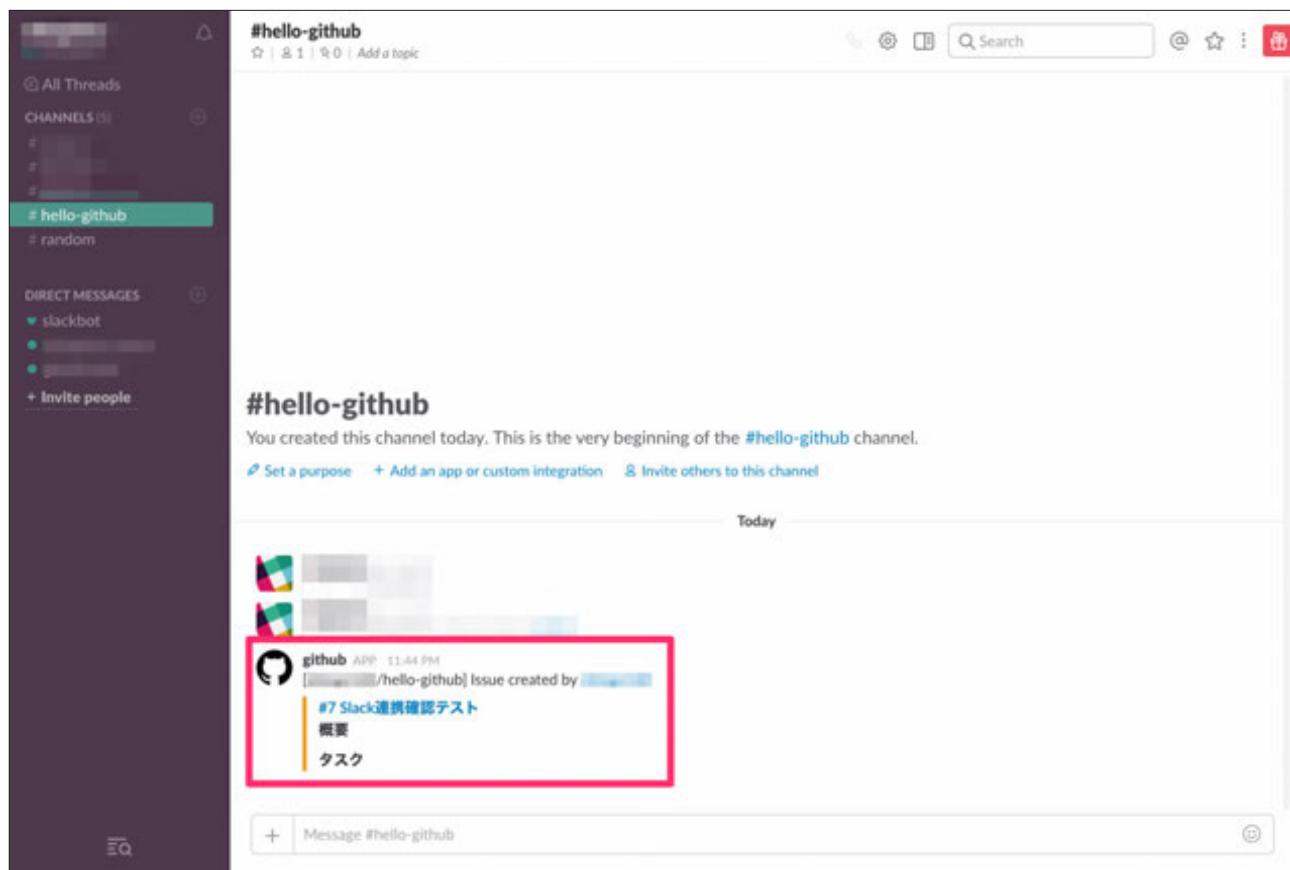


図 56 Slack 連携の動作結果

次回は「SVN（Subversion）との連携や移行」について

本稿では、Releases、Wiki、Pulse、Graphs、外部サービスとの連携を解説しました。

前回の記事で解説した「イシューやプルリクエストの周辺機能」と同様に、今回解説した機能も必ず使用しなければいけない機能ではありませんが、これらの機能をうまく活用できれば、開発にまつわる作業を改善できるかと思います。

次回は「SVN（Subversion）との連携や移行」について解説する予定です。お楽しみに!

14. たった3つで共存できる、Git / GitHub と Subversion (SVN) の連携、移行に関する基本操作

(2017年07月03日)

本連載では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説しています。今回は、「git svn」コマンドを使ってGitからSVNリポジトリへアクセスする操作、「GitHub Importer」を使ってSVNからGitHubへ移行する操作、SVNクライアントからGitHubリポジトリへアクセスする操作について。

Subversion (SVN) 環境がある現場、必見

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終える頃には、GitやGitHubの基本的な操作が身に付いた状態になっていると思います。

前回の記事「GitHubとSlackの連携の基本&知られざる便利機能 Wiki、Releases、Graphs、Pulse」では Pulse、Graphs、Releases、Wiki、外部サービスとの連携について解説しました。

連載第14回の本稿では「Apache Subversion (SVN) との連携や移行」について、以下の3つの基本的な操作を順に解説していきます。

1. 「git svn」コマンドを使用して、GitからSubversionリポジトリへアクセスする（図1の【1】の操作）
2. 「GitHub Importer」を使用して、SubversionからGitHubへの移行を実行する（図1の【2】の操作）
3. SubversionクライアントからGitHubリポジトリへアクセスする（図1の【3】の操作）

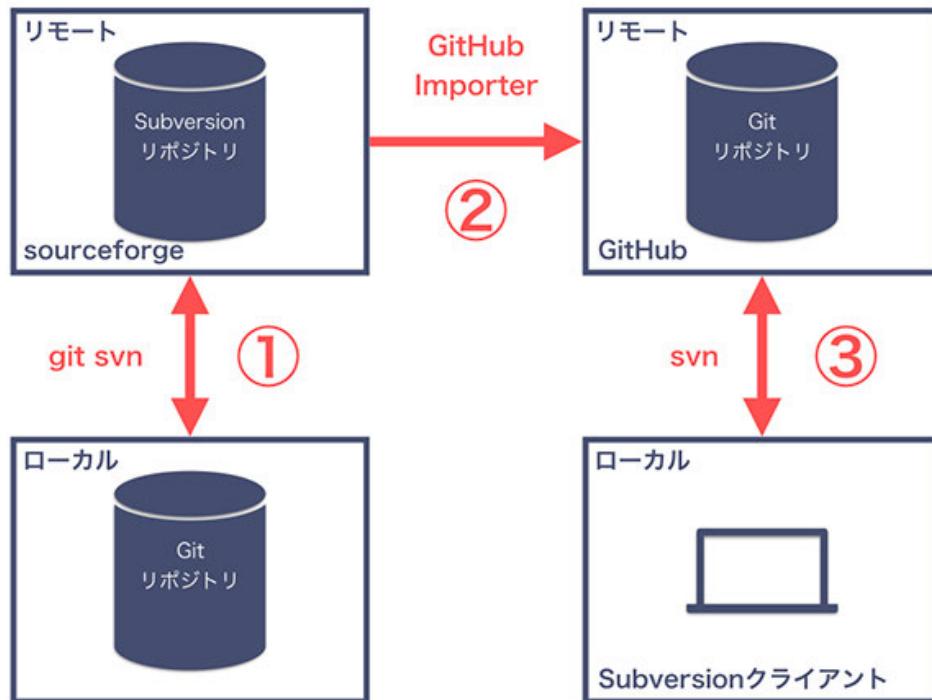


図 1 本稿で解説する 3 つの操作

環境構築について

本稿における作業に必要な環境を構築する手順は省略します。以下の環境を各自ご用意ください。

- 「git svn」コマンドを実行できる環境
- 「svn」コマンドを実行できる環境
- Subversion リポジトリ（一般的な構造である「trunk/」「branches/」「tags/」を採用したリポジトリ）

「git svn」コマンドについては、本連載の第 1 回記事で Git のインストールが完了していれば、既に利用できるかと思います。「svn」のパッケージはこちらから入手できます。

Subversion リポジトリは、sourceforge などのホスティングサービスを利用して作成できます。

【1】「git svn」コマンドを使用して、Git から Subversion リポジトリへアクセスする

「git svn」コマンドは、Subversion リポジトリと Git リポジトリの橋渡しを行ってくれるもので

このコマンドを使用すると、ローカルで Git の機能を活用した作業を行いつつ、その結果を Subversion リポジトリに反映できます。

今回は、以下の 3 つの操作を行っていきます。

- Subversion リポジトリから、ローカルの Git リポジトリを作成する
- ローカルの Git リポジトリに変更を行い、Subversion リポジトリへ反映する
- Subversion リポジトリへの変更を確認する

Subversion リポジトリから、ローカルの Git リポジトリを作成する

作業用の適当なディレクトリに移動します。

```
$ cd /Users/hirayashingo/at-it-14/work-with-git-svn
```

「git svn clone -s {Repository URL}」コマンドを使用して、Subversion リポジトリ全体をローカルの Git リポジトリにインポートします。今回使用した Subversion リポジトリはコミット数が 10 以下の大きさでしたが、インポートするのに 1 分ぐらいかかりました。

```
$ git svn clone -s svn+ssh://username@svn.code.sf.net/p/sample/code/
```

インポートが完了すると、作業用ディレクトリ内に Git リポジトリが作成されているのを確認できます。

今回試した環境の場合、この時点でのブランチは以下のようになりました。

```
$ git branch -a
* master
  remotes/origin/feature
  remotes/origin/tags/release-0.1
  remotes/origin/trunk
```

「git checkout -b」コマンドを使用すると、リモートブランチを元にローカルにブランチを作成できます。

```
$ git checkout -b feature origin/feature
Switched to a new branch 'feature'

$ git branch
* feature
  master
```

ここまで操作によって、Subversion リポジトリから Git リポジトリをローカルに作成できました（この Git リポジトリから GitHub の新規リポジトリを作成すれば、Subversion リポジトリから GitHub への移行を完了させることができます）。

ローカルの Git リポジトリに変更を行い、Subversion リポジトリへ反映する

次に、ローカルの Git リポジトリに変更を行い、その変更を Subversion リポジトリに反映します。

ファイルを変更し、コミットします。

```
$ echo "Add text from git-svn" >> hoge.txt
$ git add hoge.txt
$ git commit -m "Update hoge.txt"
[master 48e1db7] Update hoge.txt
 1 file changed, 1 insertion(+), 1 deletion(-)
```

「git svn dcommit」コマンドを使用して、変更を Subversion リポジトリに反映します。

```
$ git svn dcommit
Committing to svn+ssh://username@svn.code.sf.net/p/sample/code/trunk ...
M hoge.txt
Committed r8
M hoge.txt
r8 = 1a255450e2c77a2d594daadbcbe1d583890f7608 (refs/remotes/origin/trunk)
No changes between 48e1db7dc129d38208928192c19ab41f76304588 and refs/remotes/origin/
trunk
Resetting to the latest refs/remotes/origin/trunk
```

Subversion リポジトリへの変更を確認する

最後に、Subversion クライアントから、Subversion リポジトリの最新データを取得し、「git svn」コマンドによる変更が Subversion リポジトリに反映されていることを確認します。

Subversion リポジトリの作業コピーが存在するディレクトリに移動します。

```
$ cd /Users/hirayashingo/at-it-14/svn-working-copy
```

作業コピー更新し、ログを見ると「git svn」コマンドによる変更を確認できました。

```
$ svn update
$ svn log
-----
r8 | username | 2017-06-24 12:34:21 +0900 (Sat, 24 Jun 2017) | 1 line
Update hoge.txt
-----
....
```

「git svn」コマンドを使用して、Git から Subversion リポジトリへアクセスする操作の解説は以上です。

[2] 「GitHub Importer」を使用して、Subversion から GitHub への移行を実行する

GitHub への移行を行ってくれるツール「GitHub Importer」を使用して、Subversion リポジトリ上のコードを GitHub へ移行してみます。

GitHub Importer を開きます。

Subversion リポジトリの URL、新しく作る Git リポジトリの名前を入力し、「Begin import」をクリックします。

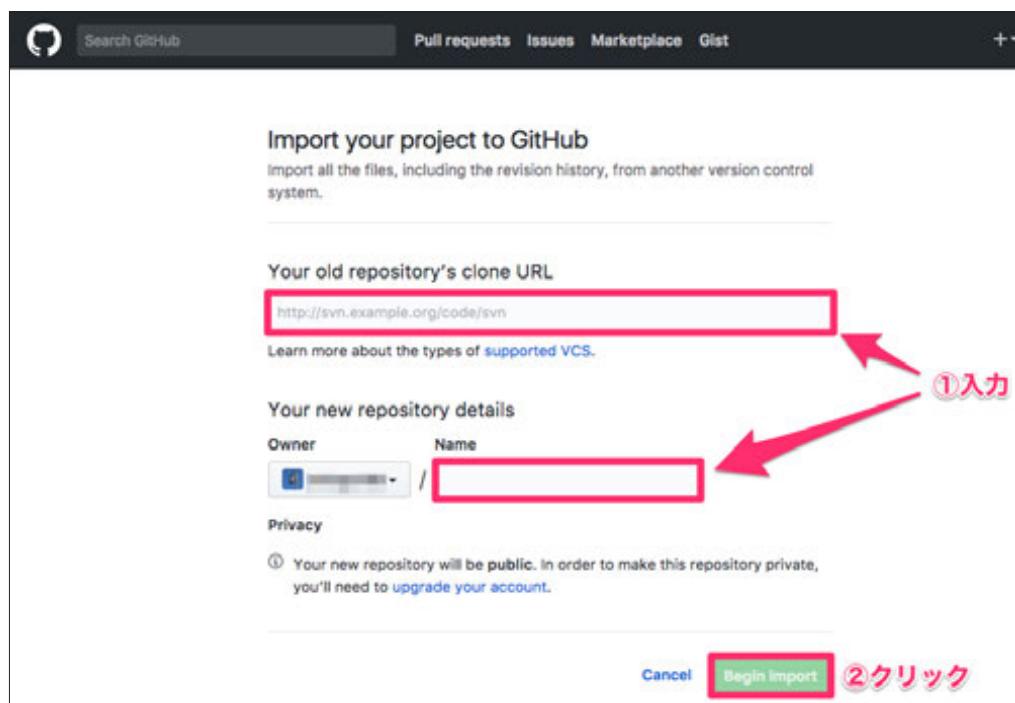


図 2 GitHub Importer

移行処理が開始します。

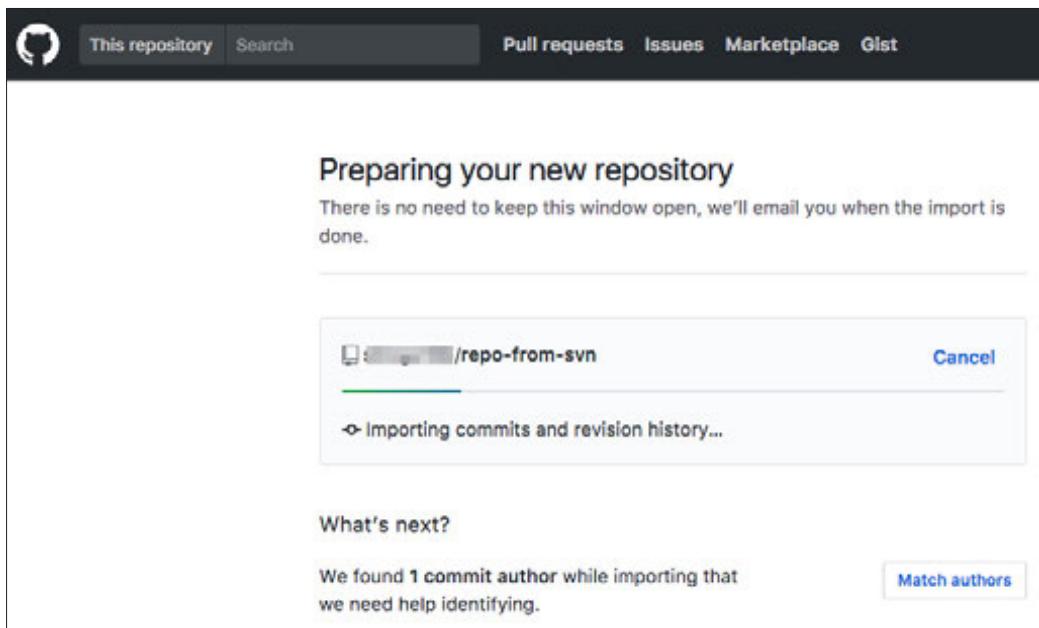


図3 GitHub Importer（移行処理中）

移行が終わると以下のように「Importing complete！」というメッセージが表示されます。

そのメッセージの右に新規作成された Git リポジトリへのリンクがあるので、そこをクリックします。

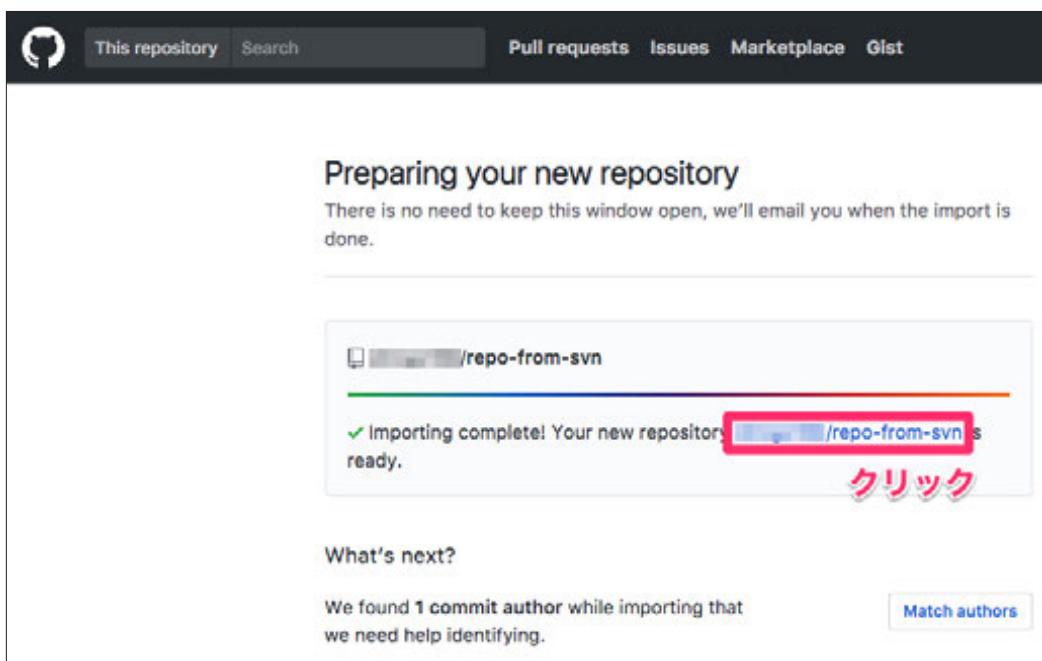


図4 GitHub Importer（移行完了）

Git リポジトリのページを表示できました。

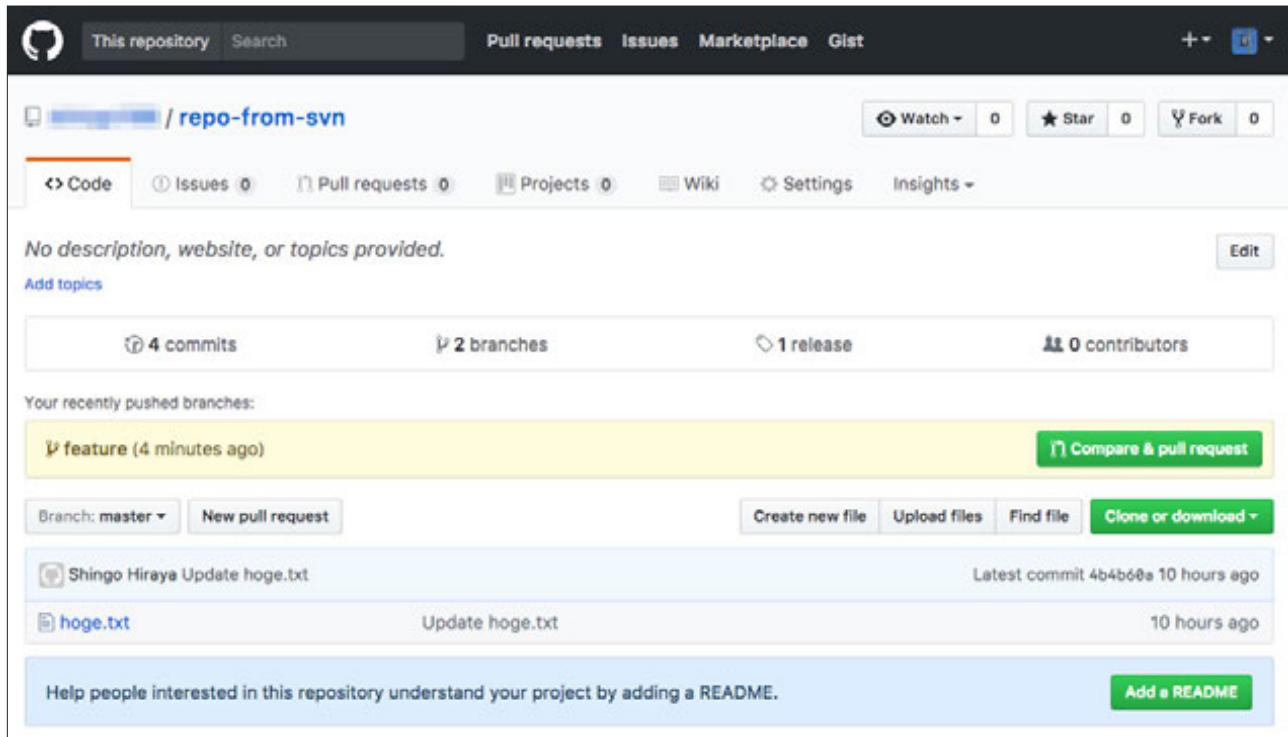


図 5 移行処理によって作成されたリポジトリのページ

「GitHub Importer」を使用して、Subversion から GitHub への移行を実行する操作の解説は以上になります。

[3] Subversion クライアントから GitHub リポジトリへアクセスする

GitHub のリポジトリは、Subversion のクライアントからアクセスできます。

本連載のこれまでの記事で作成してきた、GitHub 上の「hello-github」リポジトリに対し、幾つかの操作を Subversion のクライアント「svn」から行ってみます。

チェックアウトする

まずは、「svn checkout」コマンドを使用して、GitHub のリポジトリをチェックアウトします。

作業用の適当なディレクトリに移動します。

```
$ cd /Users/hirayashingo/at-it-14/work-from-svn-to-github
```

「svn checkout」コマンドを使用して、リポジトリの空のチェックアウトを行います。

```
$ svn checkout --depth empty https://github.com/username/hello-github.git
Checked out revision 11.
```

作成されるディレクトリに移動します。

```
$ cd hello-github.git
```

「trunk」 ブランチを取得します。通常、「master」 ブランチが「trunk」 ブランチにマッピングされます。

```
$ svn update trunk
Updating 'trunk':
A    trunk
A    trunk/ISSUE_TEMPLATE.md
A    trunk/PULL_REQUEST_TEMPLATE.md
A    trunk/README.md
Updated to revision 11.
```

「branches」 ディレクトリの空のチェックアウトを取得します。このディレクトリ内に作業用のブランチを作ることができます。

```
$ svn update --depth empty branches
Updating 'branches':
A    branches
Updated to revision 11.
```

ブランチを作成する

次に、ブランチを作成してみます。「svn copy」 コマンドを使用して、新しいブランチを作成します。

```
$ svn copy trunk branches/more_awesome
A           branches/more_awesome
```

「svn commit」 コマンドを使用して、リモートに反映します。

```
$ svn commit -m 'Added more_awesome topic branch'
Authentication realm: <https://github.com:443> GitHub
Username: username@example.com
Password for 'username@example.com': ****
Adding           branches/more_awesome
Committing transaction...
Committed revision 13.
```

ここまで操作で、ブランチを作成し、そのブランチを GitHub のリポジトリに反映する作業が完了しました。

GitHub のリポジトリページで、新しいブランチが作成されていることを確認できると思います。

The screenshot shows the GitHub repository page for 'hello-github'. At the top, there are navigation links for 'Pull requests', 'Issues', 'Marketplace', and 'Gist'. Below the header, it says 'No description, website, or topics provided.' and has a 'Add topics' button. It displays statistics: 9 commits, 3 branches, 1 release, and 2 contributors. Under 'Your recently pushed branches:', the 'more_awesome' branch is listed with a timestamp of '26 minutes ago'. There is a 'Compare & pull request' button next to it. A dropdown menu titled 'Switch branches/tags' is open, showing 'feature/add-contributing-md' and 'master'. The 'more_awesome' branch is selected and highlighted with a red box. The main content area shows the repository's README with the heading 'hello-github' and the text 'GitHubの機能を試すためのリポジトリです。Git連載記事の作業用のリポジトリです。' Below this is a '参考ページ' section with a link to 'GitHub Help'.

図 6 ブランチ作成後のリポジトリのページ

コミットを実行する

最後に、ファイルの変更を行い、コミットを実行してみます。

「README.md」ファイルに1行追加します

```
$ echo "- [Support for Subversion clients](https://goo.gl/XnrvnR)" >> trunk/README.md
```

「svn commit」コマンドを使用してコミットを実行します。

```
$ svn commit -m "Update README.md"
Sending      trunk/README.md
Transmitting file data .done
Committing transaction...
Committed revision 14.
```

GitHub のリポジトリページで、今回追加したコミットを確認できるかと思います。

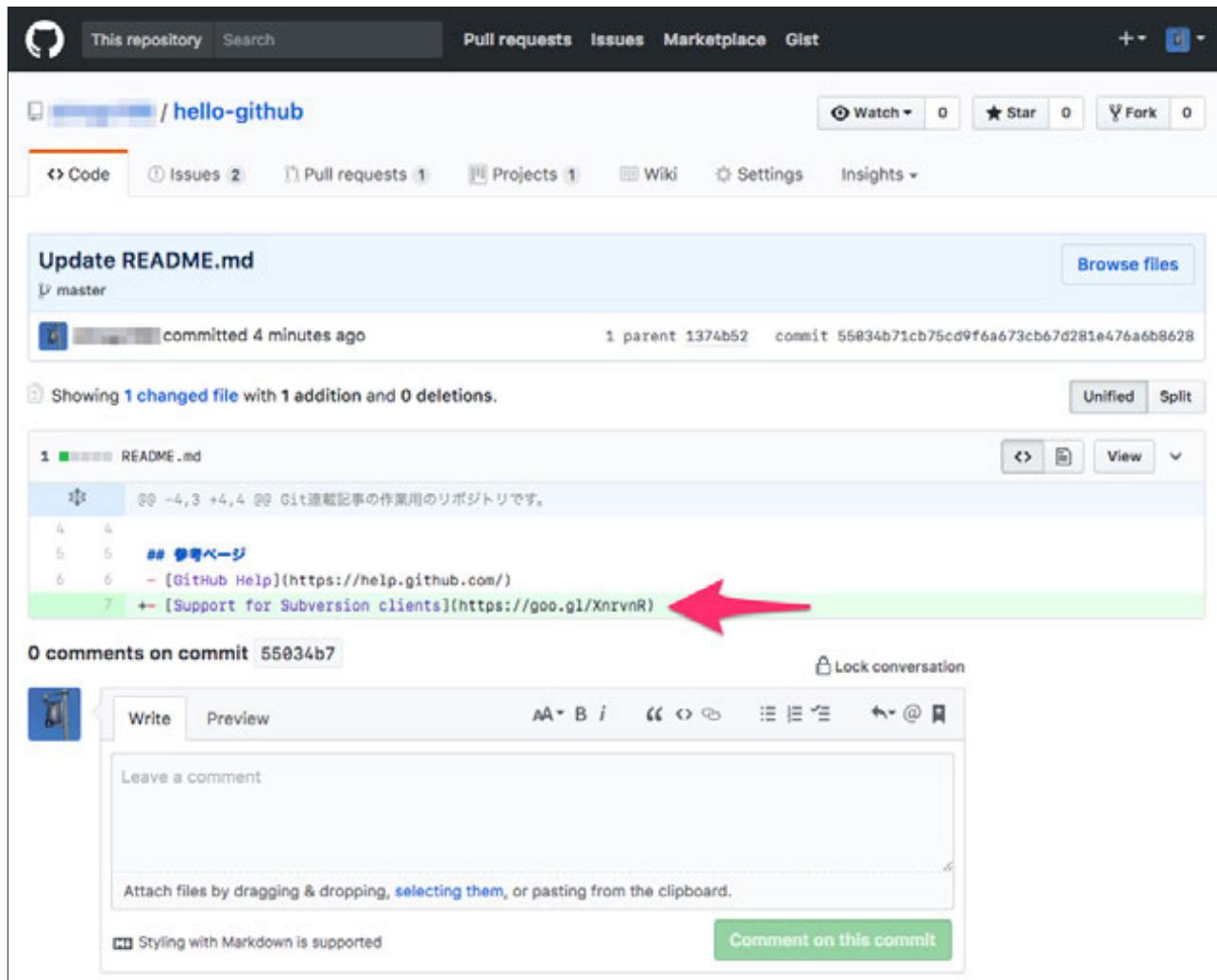


図 7 コミット実行後のリポジトリのページ

Subversion クライアントから GitHub リポジトリへアクセスする操作の解説は以上です。

次回は「Git / GitHub のワークフロー」について

連載第 14 回目の本稿では「Subversion との連携や移行」について、3 つの基本的な操作を解説しました。

次回は「Git / GitHub のワークフロー」について解説する予定です。お楽しみに!

15. 【図解】git-flow、GitHub Flow を開発現場で使い始めるためにこれだけは覚えておこう

(2017年08月01日)

本連載では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説しています。最終回は、幾つか存在するバージョン管理のワークフローのうち「git-flow」「GitHub Flow」の概要を解説します。

本連載「こっそり始める Git / GitHub 超入門」では、バージョン管理システム「Git」とGitのホスティングサービスの1つ「GitHub」を使うために必要な知識を基礎から解説していきます。具体的な操作を交えながら解説していくので、本連載を最後まで読み終える頃には、GitやGitHubの基本的な操作が身に付いた状態になっていると思います。

前回記事「たった3つで共存できる、Git / GitHubとSubversion (SVN) の連携、移行に関する基本操作」では、Subversionとの連携や移行について解説しました。

本連載の最終回となる今回のテーマは「Git / GitHub のワークフロー」です。幾つか存在するバージョン管理のワークフローのうち「git-flow」「GitHub Flow」の概要を解説します。

git-flow

「git-flow」は Vincent Driessen 氏の「A successful Git branching model」を基にしたワークフローです。

他のワークフローと比べると、大規模で複雑な構成になっています。

デスクトップ／モバイルアプリケーションのように「リリース」を必要とするソフトウェアの開発に適しています。

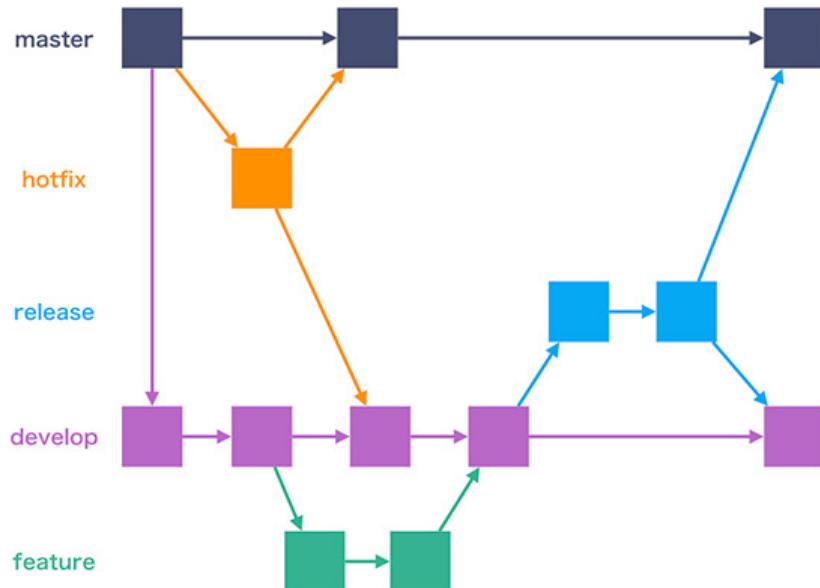


図 1 git-flow

ランチの種類と用途

git-flow を使用した開発では「メインブランチ」とそれ以外の「サポートブランチ」を使用します。

• メインブランチ

メインブランチには「master」と「develop」の2つのブランチがあります。これらのブランチは常に存在します。

種類	用途
master	リリース済みのソースコードを管理する
develop	開発中のソースコードを管理する

• サポートプランチ

タスクごとに「フィーチャー」「リリース」「ホットフィックス」のいずれかのプランチを作成し、作業を行います。

これらのブランチは master または develop ブランチから作成され、作業が完了すると削除されます。

種類	分歧元	マージ先	ブランチ名の慣習	用途
フィーチャー	develop	develop	master、develop、release-*、hotfix-*以外	機能実装やバグ修正などの開発作業を行う
リリース	develop	developとmaster	release-*	リリース準備作業を行う
ホットフィックス	master	developとmaster	otfix-*	緊急の修正作業を行う

開発フローの例

git-flow を使用した開発フローの例を見ていきましょう。

- **develop ブランチを作成する**

まずは、master ブランチから develop ブランチを作成します。

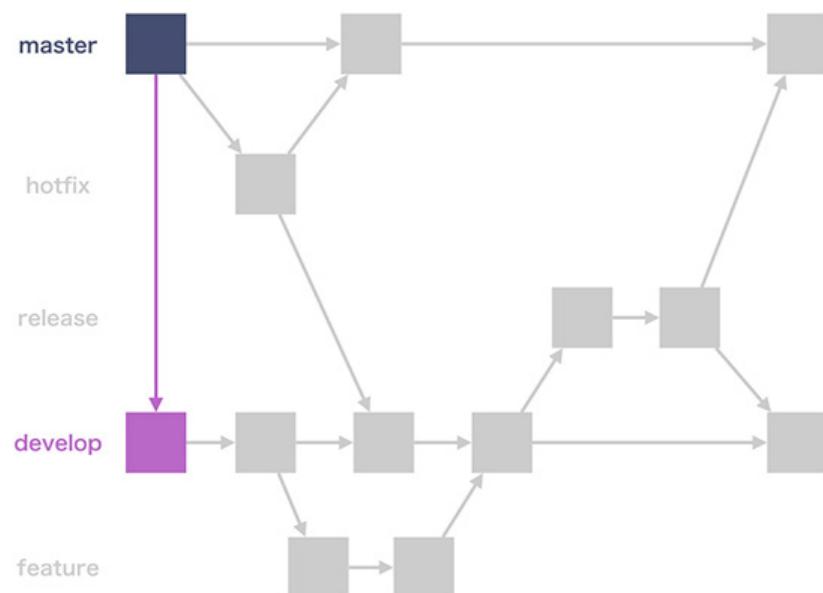


図 2 develop ブランチを作成

- **機能実装を開始する**

develop ブランチからフィーチャーブランチを作成し、機能実装作業を開始します。コミットはフィーチャーブランチに対して行います。

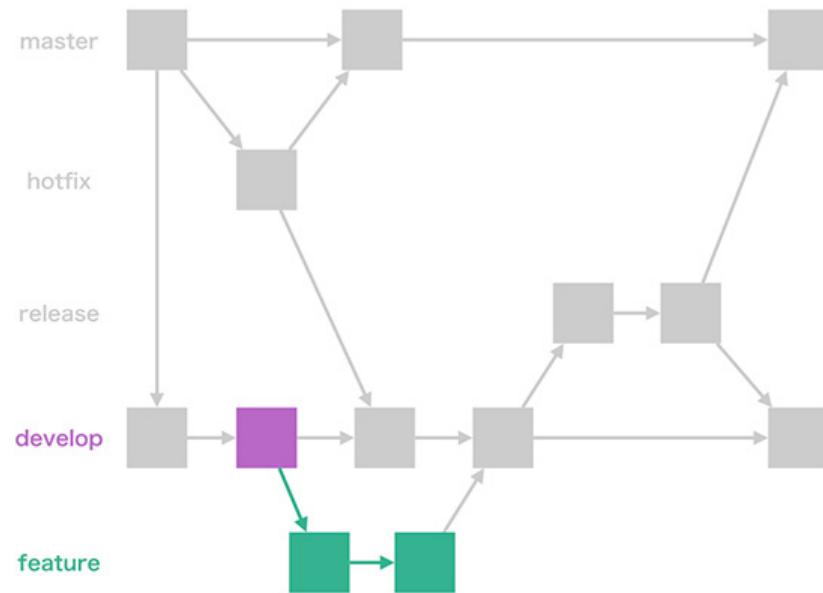


図 3 フィーチャーブランチを作成

- 機能実装を完了する

フィーチャーブランチでの作業が完了したら、フィーチャーブランチを develop ブランチにマージします。

マージ完了後にフィーチャーブランチを削除します。

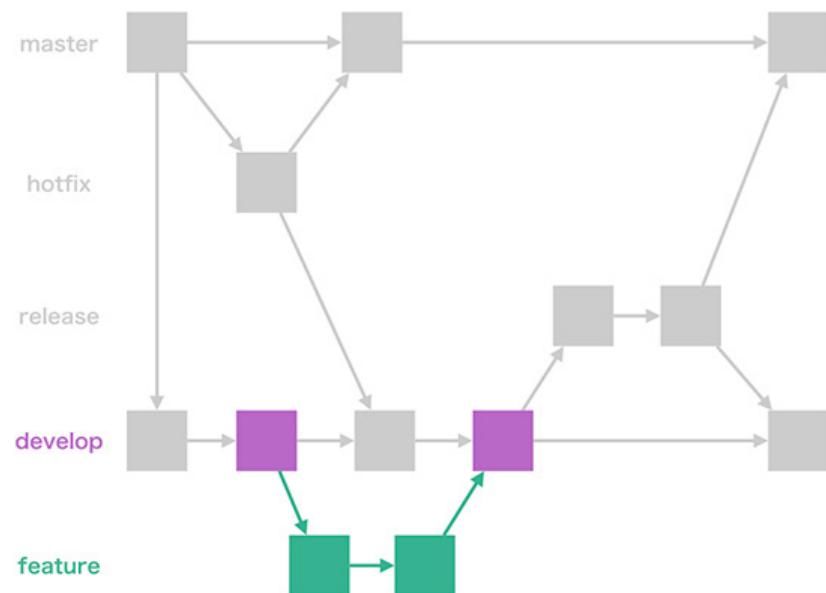


図 4 フィーチャーブランチを develop ブランチにマージ

- リリース準備を開始する

機能実装が終わりリリースできる状態になったら、develop ブランチからリリースブランチを作成します。

バージョン番号やドキュメントの更新などのリリース準備作業を行います。

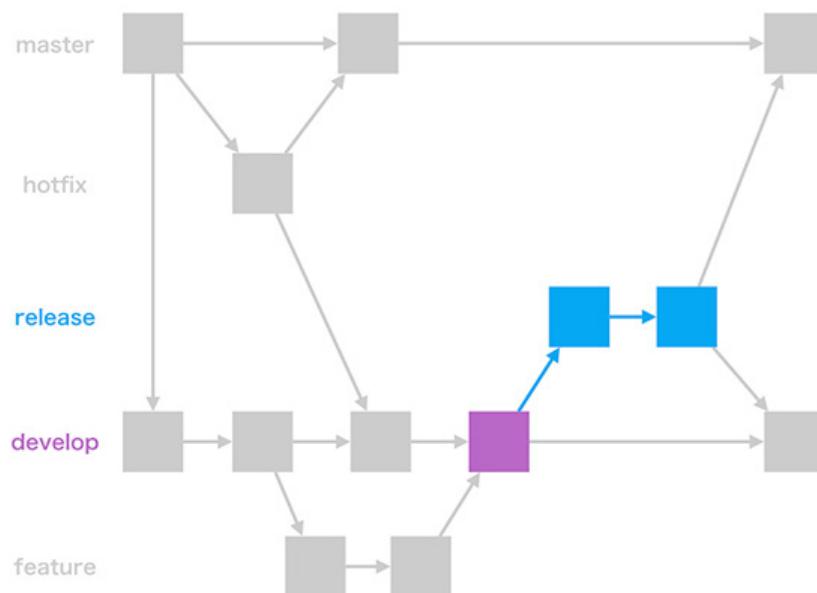


図 5 リリースブランチを作成

- リリース準備を完了する

リリースブランチでの作業が完了したら、リリースブランチを master と develop ブランチにマージします。

マージ完了後にリリースブランチを削除します。

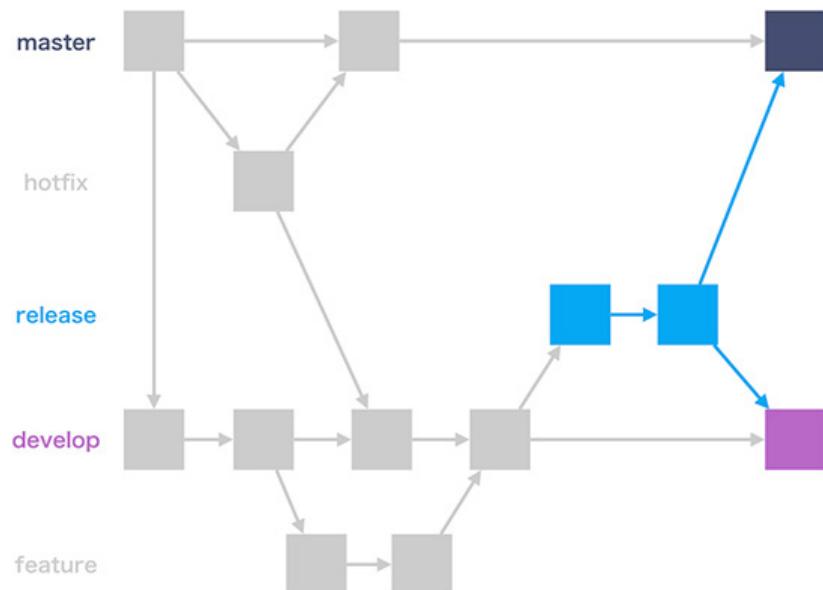


図 6 リリースブランチを master と develop ブランチにマージ

- 緊急の修正作業を開始する

リリース後に緊急の修正作業が発生した場合は、master ブランチからホットフィックスブランチを作成し、修正作業を行います。

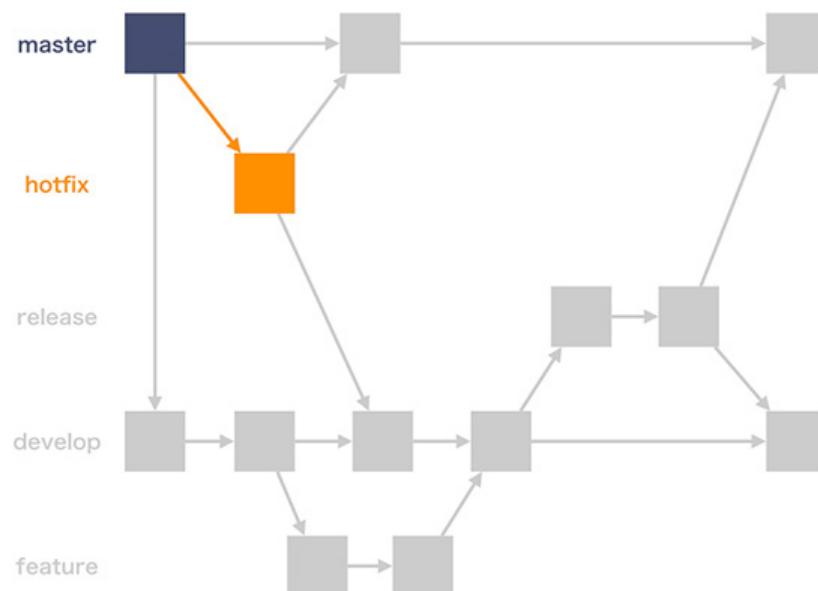


図 7 ホットフィックスブランチを作成

- 緊急の修正作業を完了する

ホットフィックスブランチでの作業が完了したら、ホットフィックスブランチを master と develop ブランチにマージします。

マージ完了後にホットフィックスブランチを削除します。

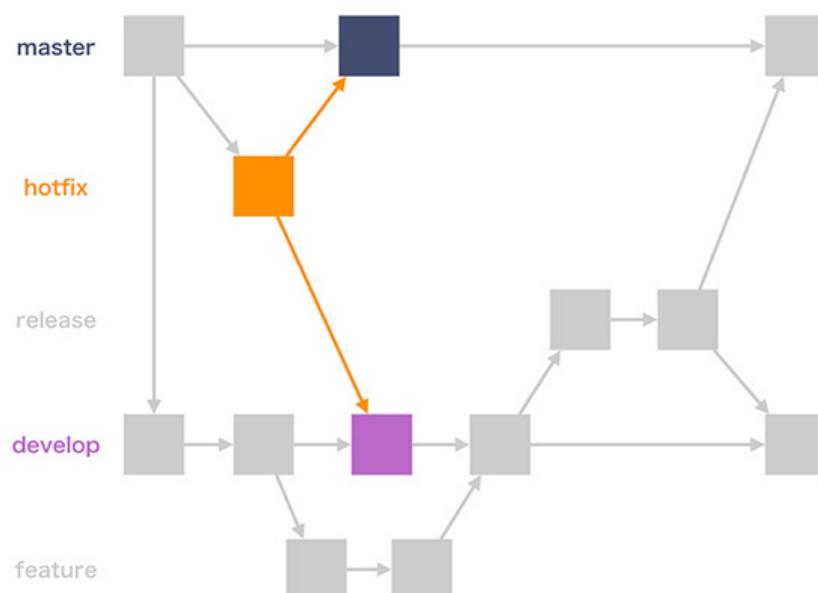


図 8 ホットフィックスブランチを master と develop ブランチにマージ

git-flow の概要の解説は以上です。

GitHub Flow

「GitHub Flow」は「GitHub」の開発で使用されているワークフローであり、「git-flow」に比べてシンプルな構成になっています。

1日に複数回デプロイを行うような Web アプリケーションの開発に適しています。

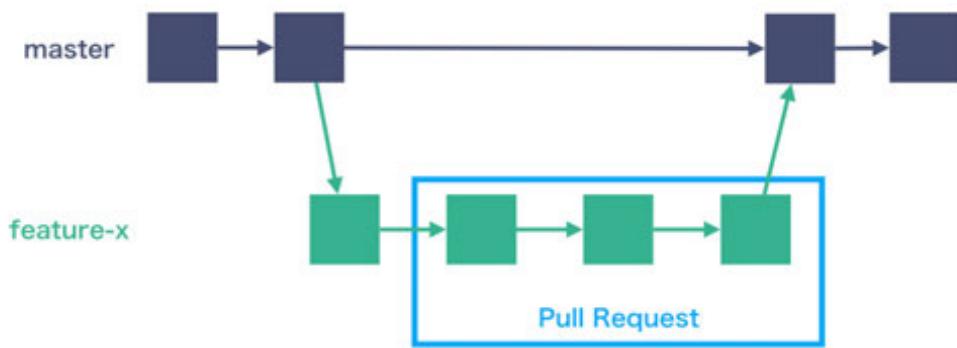


図 9 GitHub Flow

6つのルール

GitHub Flow には以下の 6 つのルールがあります。【ルール 1】が最も重要で、それ以外のルールは【ルール 1】を実現するために存在します。

- 【ルール 1】 master ブランチは常にデプロイ可能である
- 【ルール 2】 作業用ブランチを master から作成する（例：new-oauth2-scopes）
- 【ルール 3】 作業用ブランチを定期的にプッシュする
- 【ルール 4】 プルリクエストを活用する
- 【ルール 5】 プルリクエストが承認されたら master へマージする
- 【ルール 6】 master へのマージが完了したら直ちにデプロイを行う

開発フローの例

GitHub Flow を使用した開発フローの例を見ていきましょう。

- [1] 開発作業を行う

作業開始時に作業用ブランチを master ブランチから作成します。

git-flow では、「フィーチャー」「リリース」「ホットfixes」のいずれかのブランチを master または develop ブランチから作成しますが、GitHub Flow では、全てのブランチを master ブランチから作成します。

ブランチ名は、何の作業を行っているかが分かる名前にします。また、作業用ブランチは定期的にリモートリポジトリにプッシュするようにします。これによって、他の開発者の作業状況を把握できるようになります。

本ステップで行う作業は、[連載第 10 回記事の「プルリクエスト作成の準備」](#)で解説しています。

- [2] プルリクエストを行う

作業用ブランチを master ブランチへマージできる状態になったら、プルリクエストを作成して他の開発者にコードレビューを依頼します。そして、プルリクエストが承認されたら master へマージします。

GitHub Flow を使用した開発では、プルリクエストを積極的に活用します。作業完了後のコードレビューだけではなく、作業途中の実装への助言を求める場合などにも使えます。

本ステップで行う作業は、[連載第 10 回記事の「プルリクエストを作成する」「プルリクエストをレビューする」「プルリクエストをマージする」](#)で解説しています。

- [3] デプロイする

master へのマージが完了したら直ちにデプロイを行います。

GitHub Flow の概要の解説は以上です。

参考ページも見てみよう

本稿では「git-flow」と「GitHub Flow」の概要を解説しました。

どちらのワークフローも「厳密な規則」ではありません。また、それぞれのワークフローには長所・短所があります。

実際の開発では、git-flow または GitHub Flow をそのまま使用することではなく、ワークフローの一部を修正するなどの調整が必要になるかと思います。

git-flow と GitHub Flow の詳しい情報については、以下の参考ページをご覧ください。

- A successful Git branching model » nvie.com
- 見えないチカラ : A successful Git branching model を翻訳しました
- GitHub Flow - Scott Chacon
- GitHub Flow (Japanese translation)
- いまさら聞けない、成功するブランチモデルと git-flow の基礎知識 - @ IT