

Git for Windows を使う

v2.1

Seiichi Nukayama

2023 年 9 月 24 日

目次

1	GitHub アカウントの作成	1
1.1	アカウントを作成する	1
1.2	アクセス・トークンの作成	6
2	Git for Windows のインストール	9
2.1	Git for Windows のダウンロード	9
2.2	Git for Windows のインストール	9
2.3	参考サイト	18
3	SourceTree のインストール	19
3.1	ダウンロードとインストール	19
3.2	参考サイト	22
4	とりあえず使ってみる	
	(GitHub にリポジトリを作成してから、それを自分の PC にクローンする方法)	23
4.1	GitHub に新しいリポジトリを作る	23
4.2	GitHub からクローン	26
4.3	リポジトリの作成とは?	28
5	Git とは、どういうものか?	29
5.1	ネットに接続しないで作業する	29
5.2	インターネットに接続して、github と連携する	34
6	主なコマンド	39
6.1	現在の状態を知る	39
6.2	編集・追加・削除をおこなったあと	39
6.3	リモートの変更を取り入れる	43

7	現在のフォルダを Git 対応にする	45
8	こんな時どうする?	46
8.1	git log で文字化け	46
8.2	git status で文字化け	46
8.3	リモート・レジストリにアップしたくないファイルがある	46

1 GitHub アカウントの作成

以下のページを参考に、アカウントを作成する。

アカウントの作成はメールアドレスによる認証が必要なぐらいで簡単にできると思うが、問題はそれあとのアクセストークンの設定だろう。GitHub は今まではパスワードによる認証をしていたのだが、それを廃止して、よりセキュアなアクセストークンによる認証に切り替えたのである。^{*1}

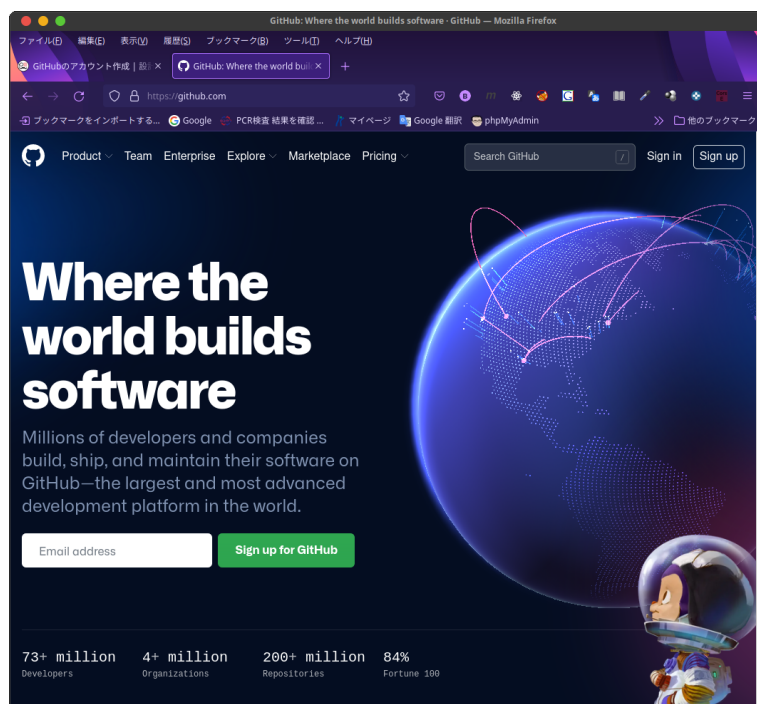
アクセストークンの設定についての参考ページは以下。

1. GitHub で個人アクセストークンを設定する
2. 新しい GitHub アカウントへのサインアップ

アカウントの作成については、たとえば以下。GitHub のアカウント作成

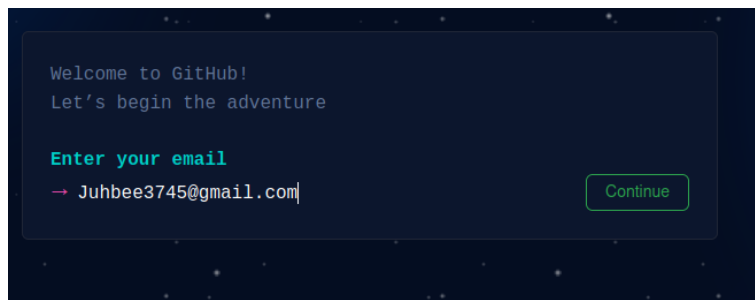
1.1 アカウントを作成する

まず、<https://github.com/> にアクセスする。

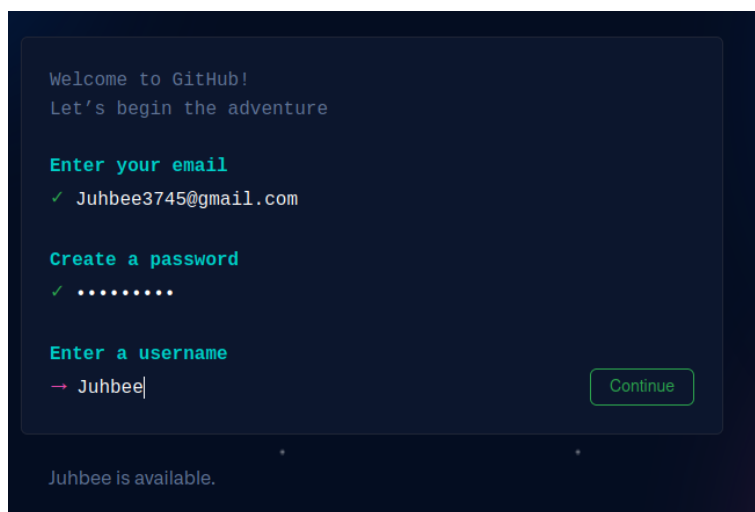


右上の **Sign up** をクリックする。("Sign in" はアカウントを持っている人のログイン用)

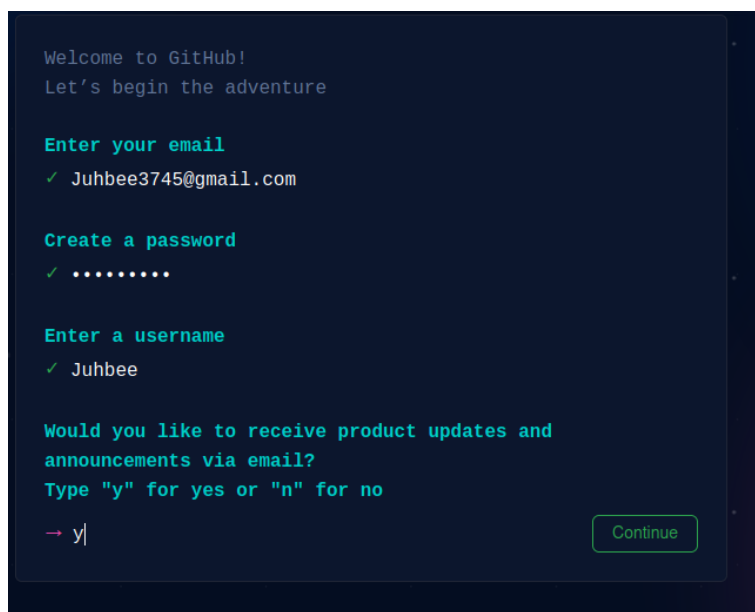
^{*1} SSH による認証——ここではやらないが、SSH によるログイン認証もできる。この場合、windows10 に OpenSSH をインストールして、ssh コマンドで認証鍵 (秘密鍵、公開鍵) を生成し、公開鍵を GitHub にアップロードすることになる。参考サイトは以下。Windows 10 で C 言語開発をしよう! GitHub.com を使う



E-Mail アドレスの入力。このアドレス宛てに、GitHub からログイン案内リンクが送られてくる。スマホのメールアドレスでもいけると思う。



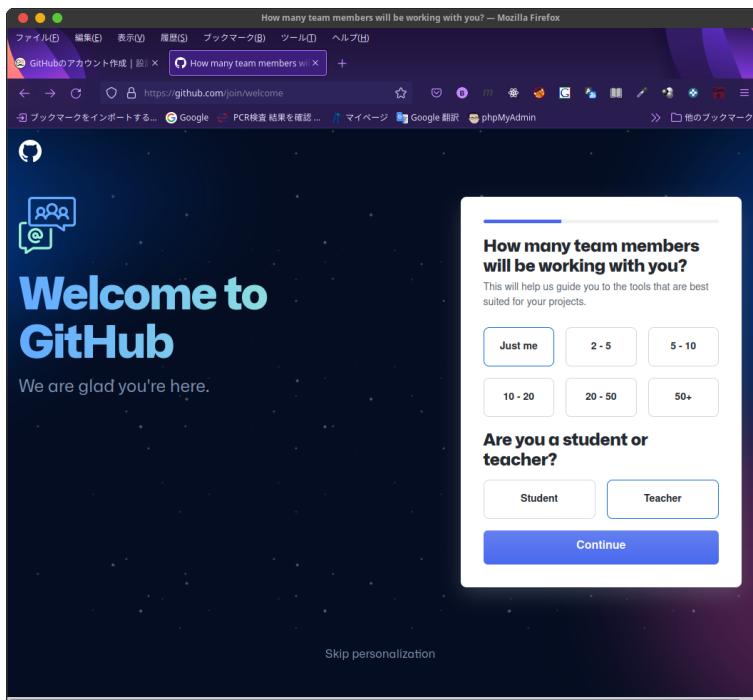
このパスワードは、GitHub にログインするためのもの。ユーザーネームも同じである。



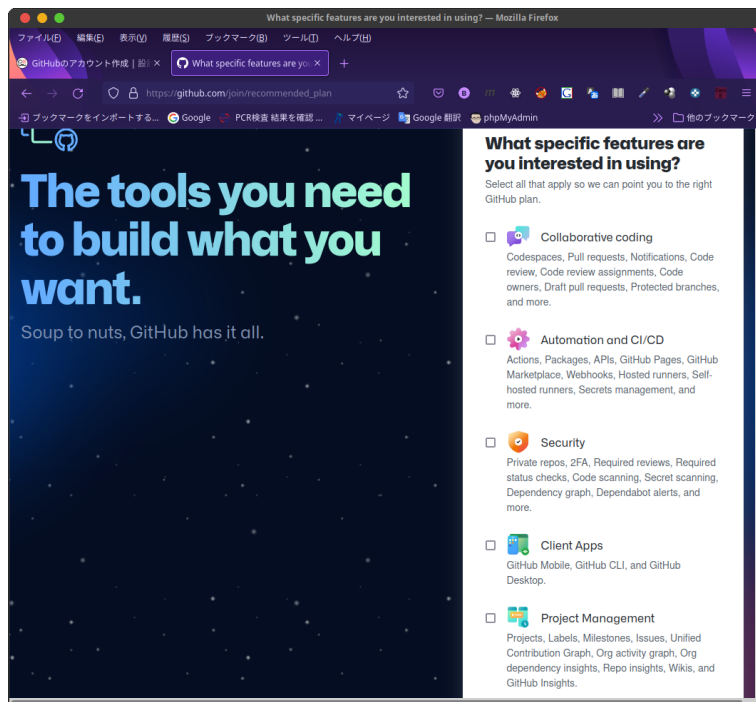
GitHub からのお便りを受け取るか？ということで、「y」でいいだろう。



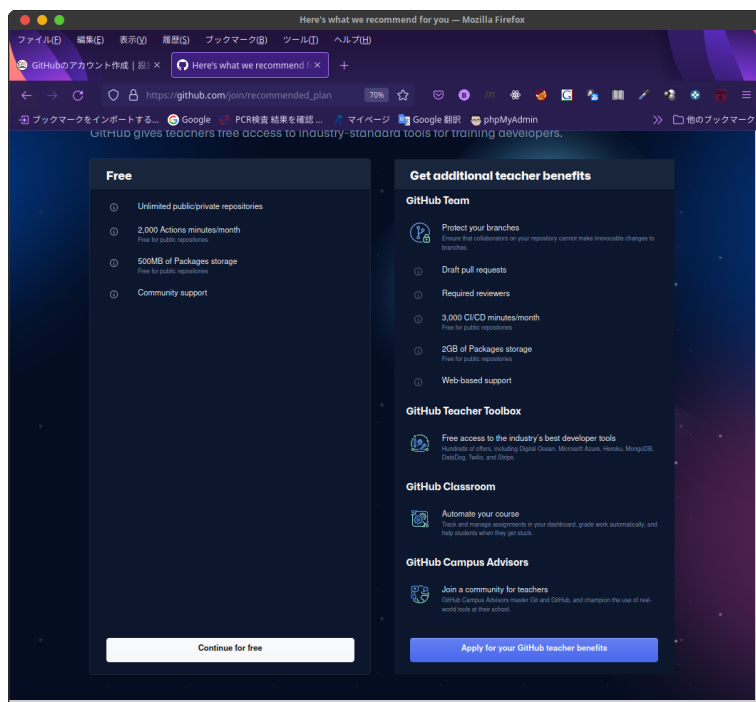
画像をクリックして、人間であることを証明する。



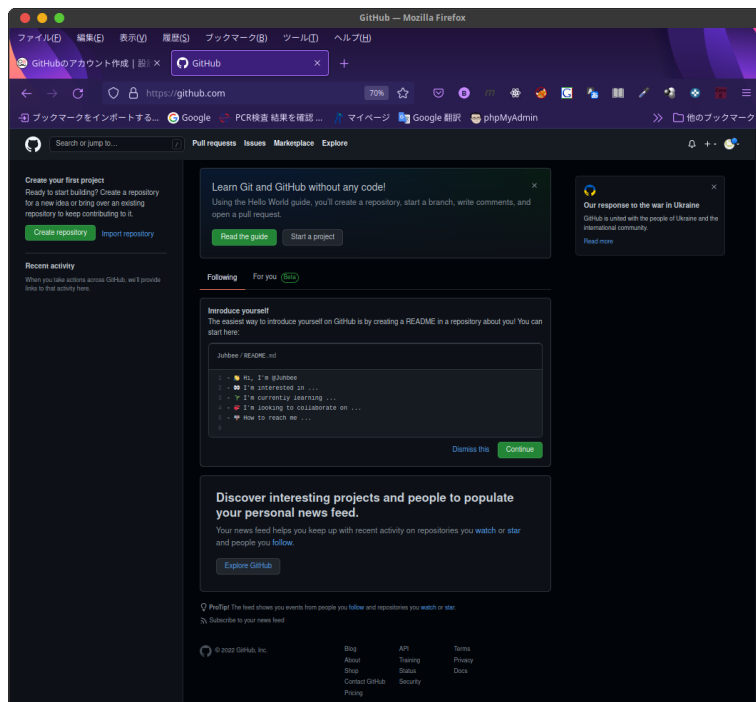
“Welcome” と表示され、右に質問ができるが、適当に答えておいていいと思う。



どういうことに興味があるかということかな？



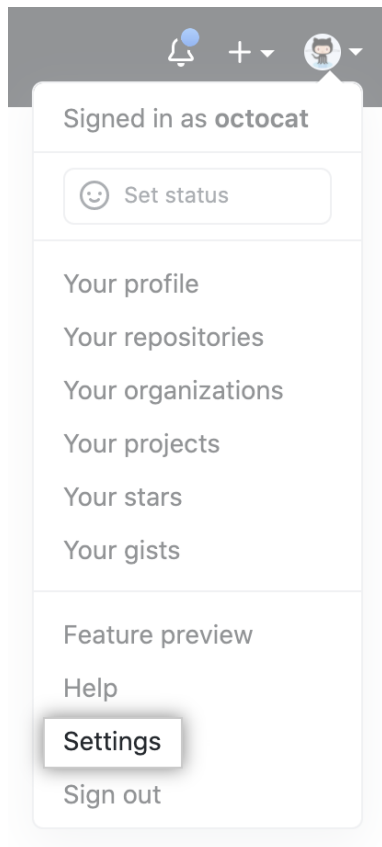
左側の Continue for free を選択する。



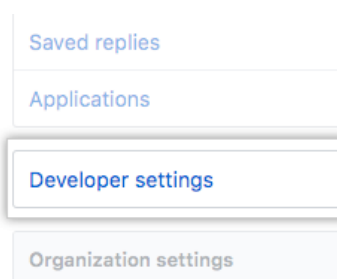
これが、自分のトップページ。

1.2 アクセス・トークンの作成

次に、アクセス・トークンを作成する。Git リポジトリを手元の PC にダウンロードすることを Git では“クローン”と呼ぶ。また、手元の PC でソースコードを編集し、それを Git リポジトリに反映させることを“プッシュ”と呼ぶ。クローンするにも、プッシュするにも、このアクセス・トークンが認証に使われる。



トップページの右上のアカウントのアイコンをクリックして、“settings”を選択する。



画面左側のメニューの一番下の“Developer settings”を選択する。

OAuth Apps

GitHub Apps

Personal access tokens

開いた画面左側のメニューの一番下 "Personal access tokens" を選択。

Personal access tokens

Generate new token

"Personal access tokens" というページの少し右上に "Generate new token" というボタンがあるので、それをクリック。

Note

My bash script

What's this token for?

トークンに名前をつける。

Expiration

7 days



The token will expire on Friday, Feb 8 2008

トークンの有効期限を設定する。3ヶ月などと設定することが推奨されている。"no expiration" も選択できる。

3ヶ月で再設定するのも面倒なので、「期限なし」でいいと思う。

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input type="checkbox"/> admin:org	Full control of orgs and teams
<input type="checkbox"/> write:org	Read and write org and team membership
<input type="checkbox"/> read:org	Read org and team membership
<input type="checkbox"/> admin:public_key	Full control of user public keys
<input type="checkbox"/> write:public_key	Write user public keys
<input type="checkbox"/> read:public_key	Read user public keys
<input type="checkbox"/> admin:repo_hook	Full control of repository hooks
<input type="checkbox"/> write:repo_hook	Write repository hooks
<input checked="" type="checkbox"/> read:repo_hook	Read repository hooks
<input type="checkbox"/> admin:org_hook	Full control of organization hooks
<input type="checkbox"/> gist	Create gists
<input type="checkbox"/> notifications	Access notifications
<input type="checkbox"/> user	Update all user data
<input type="checkbox"/> user:email	Access user email addresses (read-only)
<input type="checkbox"/> user:follow	Follow and unfollow users
<input type="checkbox"/> delete_repo	Delete repositories

トークンの有効範囲を設定する。最低、"repo"、"admin:repo_hook"、"delete_repo" は必要みたい。全部選択することをすすめているサイトもある。

<input type="checkbox"/> write:gpg_key	Write user gpg keys
<input type="checkbox"/> read:gpg_key	Read user gpg keys

"Generate token" ボタンをクリックして、トークンを生成する。

Tokens you have generated that can be used to access the GitHub API.

Make sure to copy your new personal access token now. You won't be able to see it again!

✓ ghp_IqIPN0ZH6z0wIEB4T9A2g4EHMy8Ji42q4HA8

トークン文字列が生成・表示されるので、コピーしておく。テラパッドにでも貼り付けて大事に保存しておく。(なくすと大変)

2 Git for Windows のインストール

SourceTree をインストールすると、Git for Windows は自動でインストールされる。しかし、ホーム・フォルダの AppData フォルダの奥深くインストールされてしまう。それでも、SourceTree 上の「ターミナル」を使えば、Git コマンドは使用できるのであるが、ここでは、Git for Windows を SourceTree とは別にインストールする。

Git for Windows のインストールも知っておくほうがいい。

2.1 Git for Windows のダウンロード

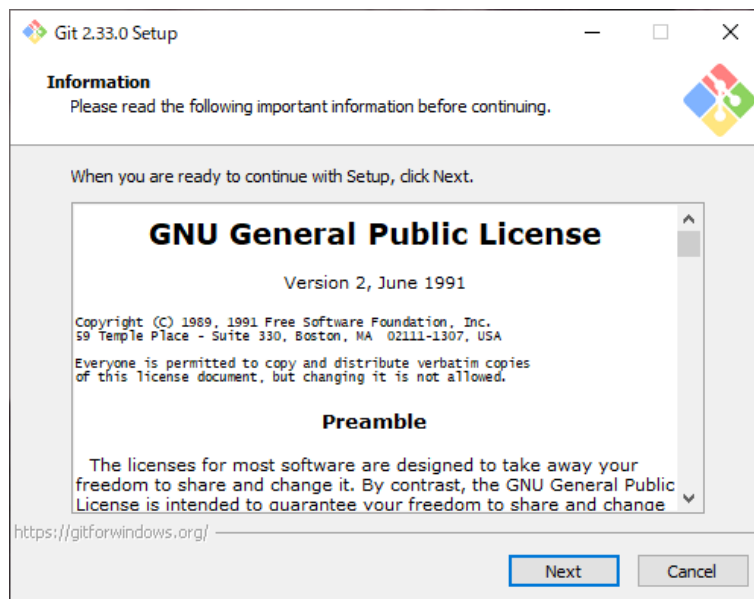
ダウンロードサイト

<https://gitforwindows.org/>

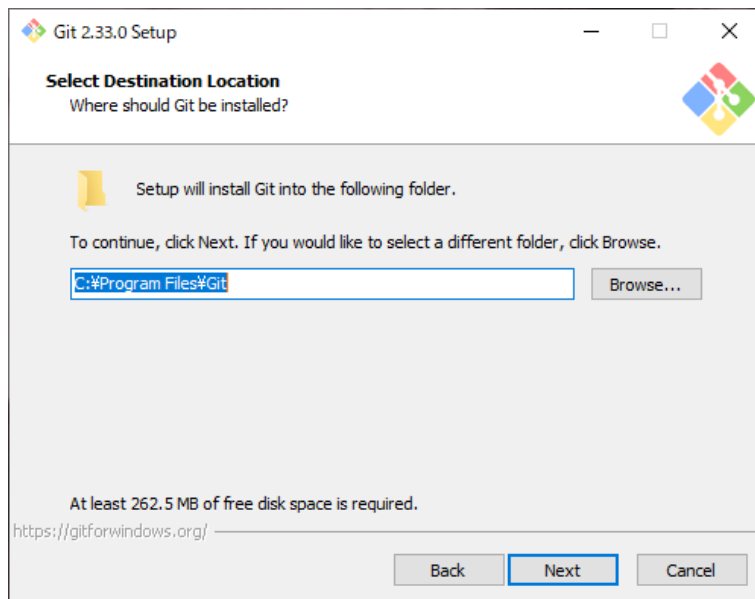
Git-2.35.1.2-64-bit.exe をダウンロードする。

2.2 Git for Windows のインストール

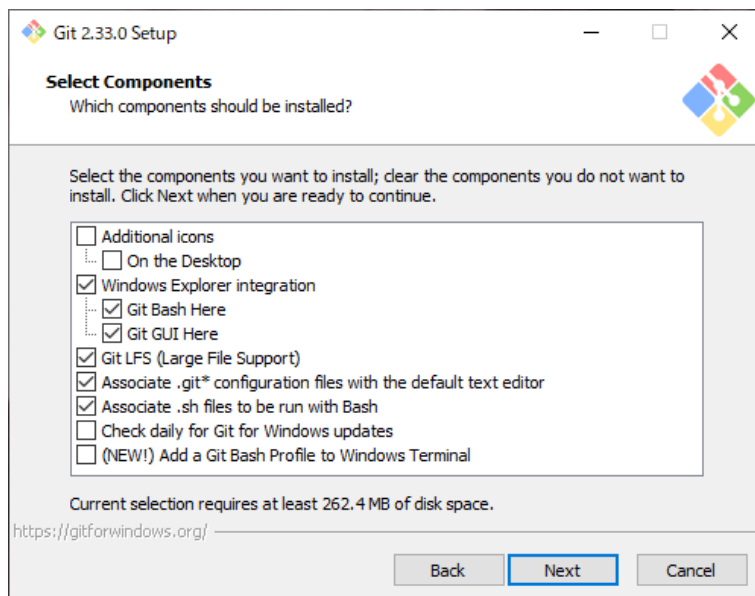
Git-2.33.0.2-64-bit.exe をダブルクリックでインストール開始。



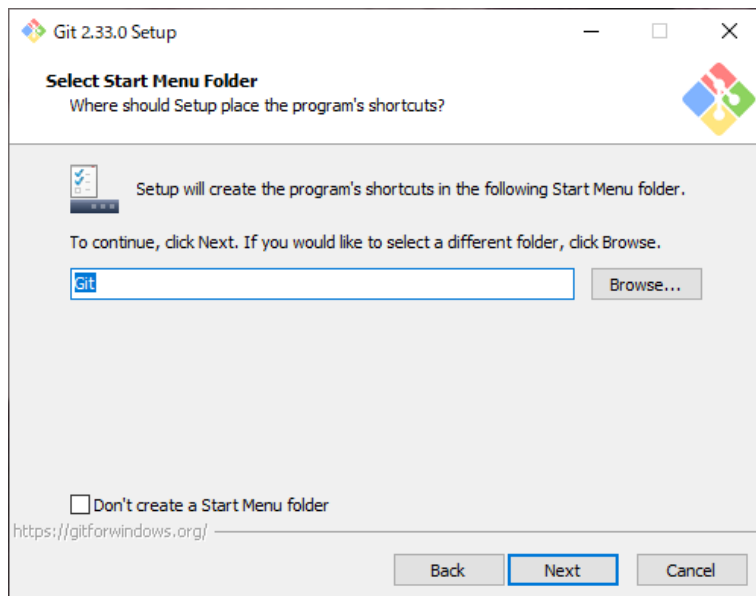
'Next' をクリック。



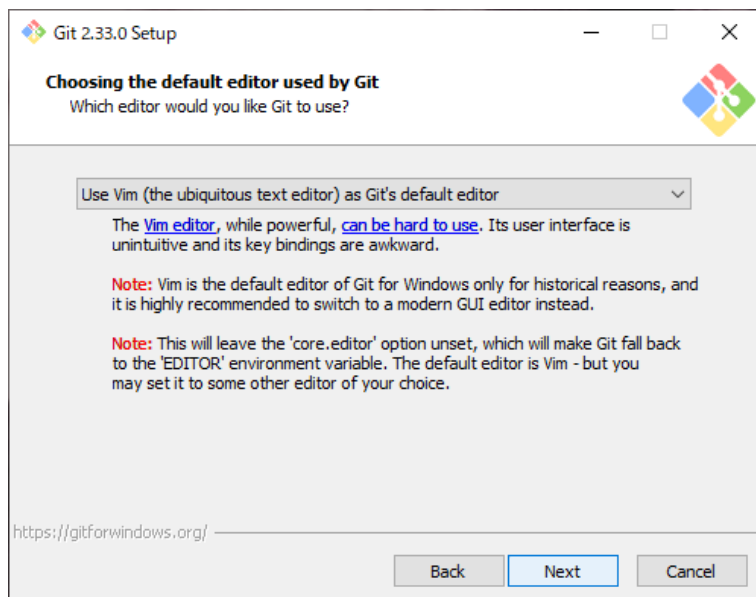
インストールされるフォルダは `c:\Program Files\Git` となる。
'Next' をクリック。



コンポーネントを選択。そのままでもよい。
あるいは下の "Check daily Git for Windows updates." (毎日 Git for Windows の更新を確認する) にチェックを入れてもよい。'Next' をクリック。

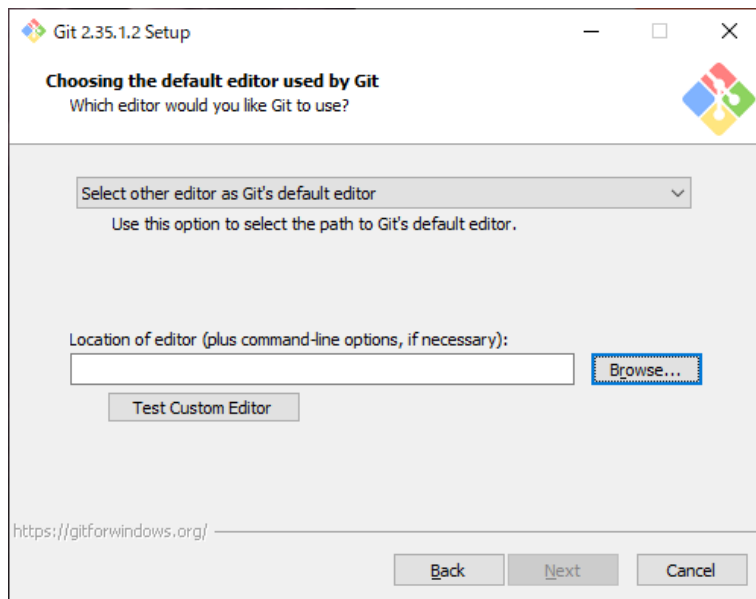


メニューの名前。そのままでもいい。'Next' をクリック。

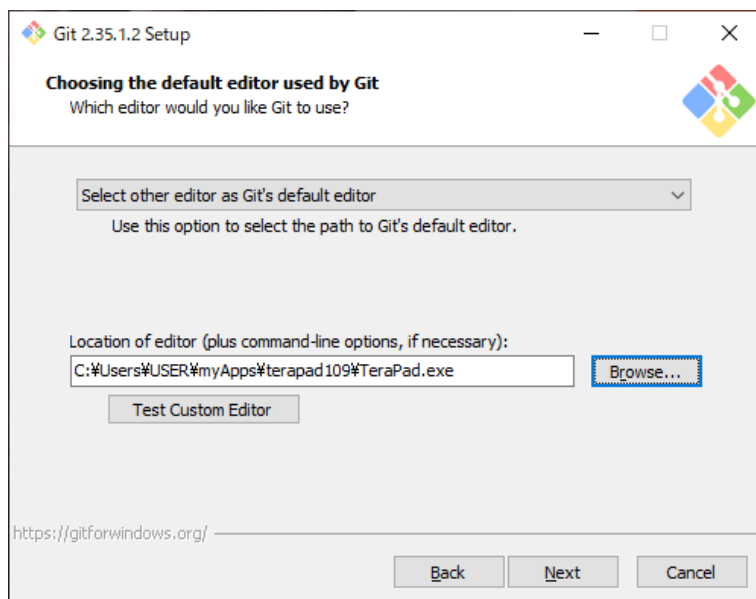


エディターの選択。そのままだと "vim" というエディタが選択される。ここではテラパッドを選択してみる。

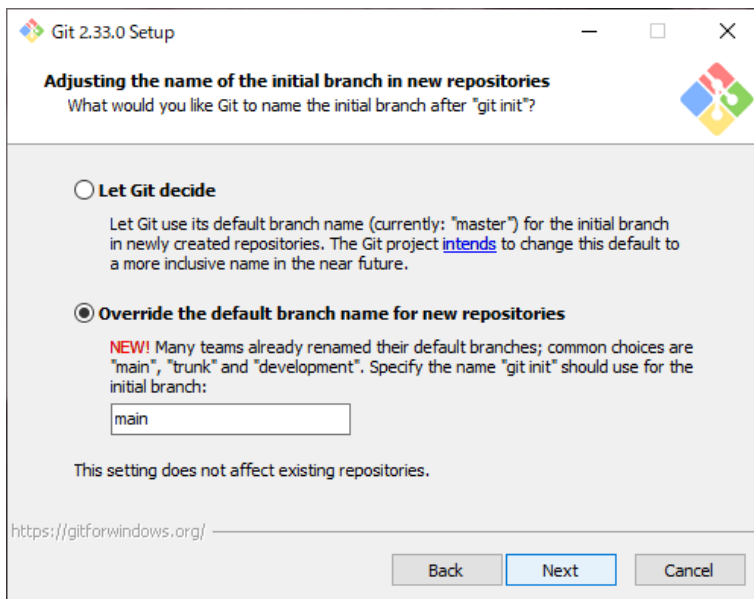
"use..." の右の三角をクリックして、"Select other editor as Git's default editor" を選択する。すると、以下のような画面になる。



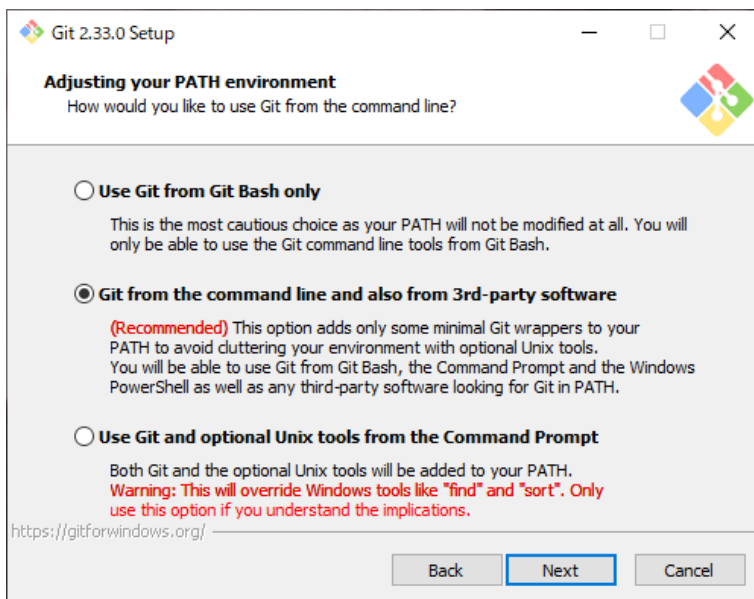
右の“Browse”をクリックして、“TeraPad.exe”がある場所を指定する。
“C:¥Program Files(x86)¥terapad¥TeraPad.exe”のはず。



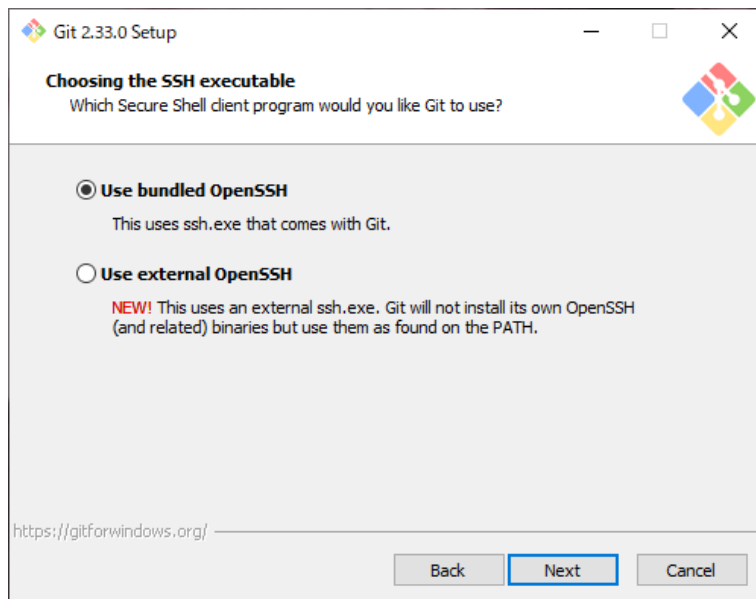
下の“Test Custom Editor”をクリックすると、テラパッドが起動して、文字が書かれている (英語)。適当に編集して保存すると、“success”と表示される。
‘Next’をクリック。



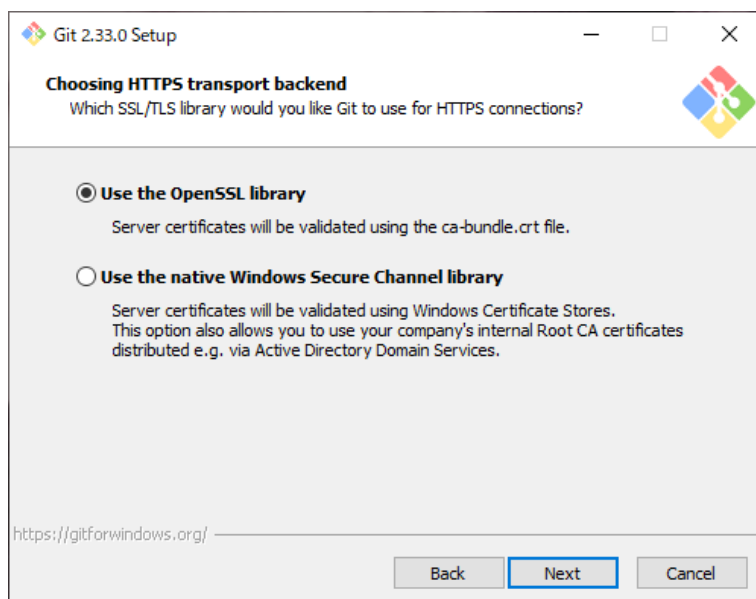
ここは、下の Override the default branch name for new repositories を選択する。
デフォルトのブランチ名として main を使えるようにしておく。



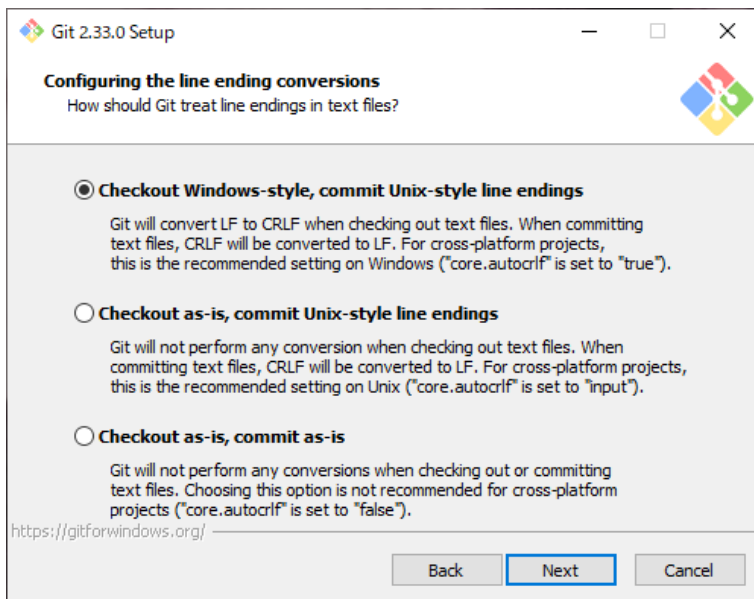
Git をコマンドラインで使うための設定。まん中の Git from the command line and also from 3rd-party software を選択する。



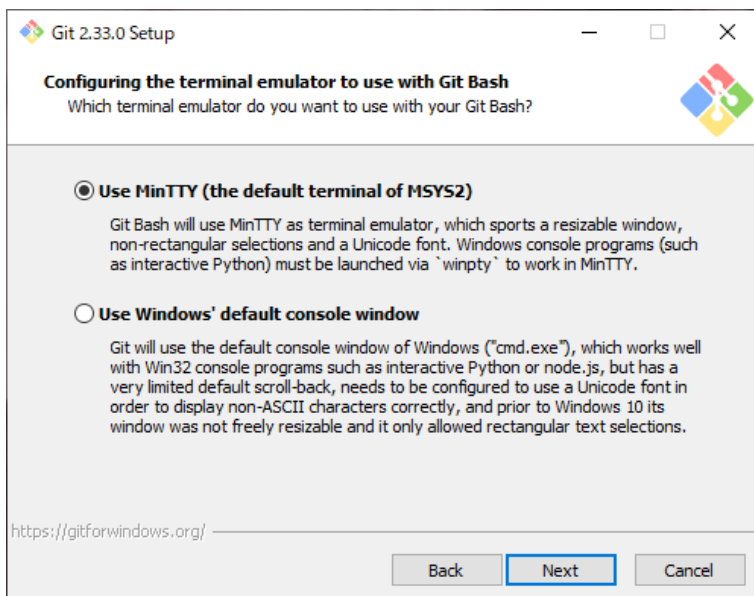
Use bundled OpenSSH を選択する。



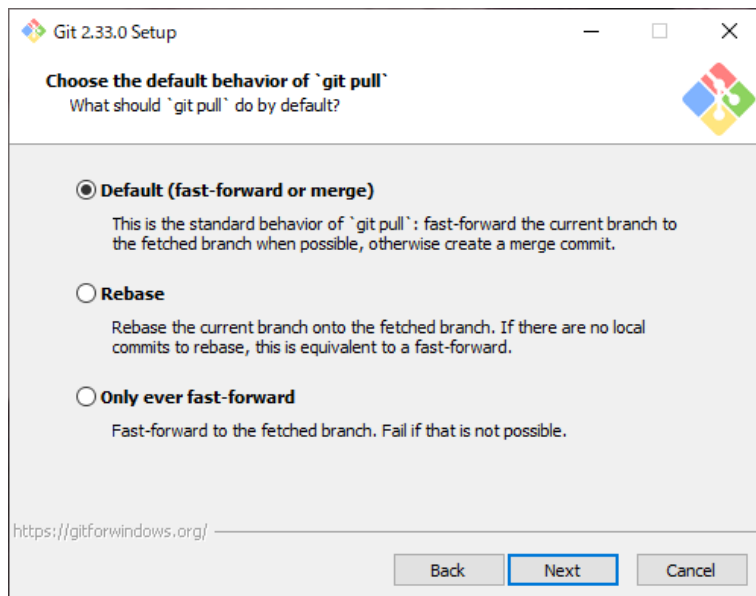
Use the OpenSSL library を選択する。



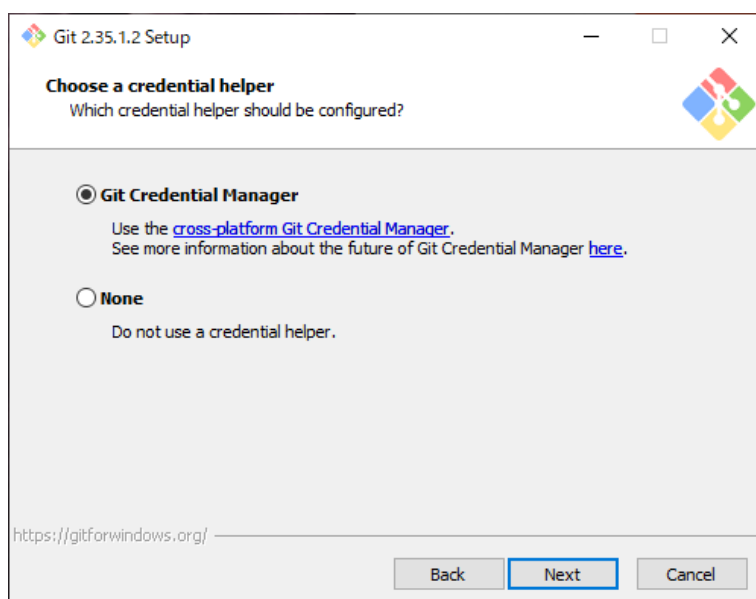
改行コードの設定。Mac や Unix では LF だけど、Windows は CRLF。職場では Windows。自宅では Mac という状態を考慮して、一番上の Checkout Windows-style, commit Unix-style line endings を選択。普通は、開発チームで話しあって決める。



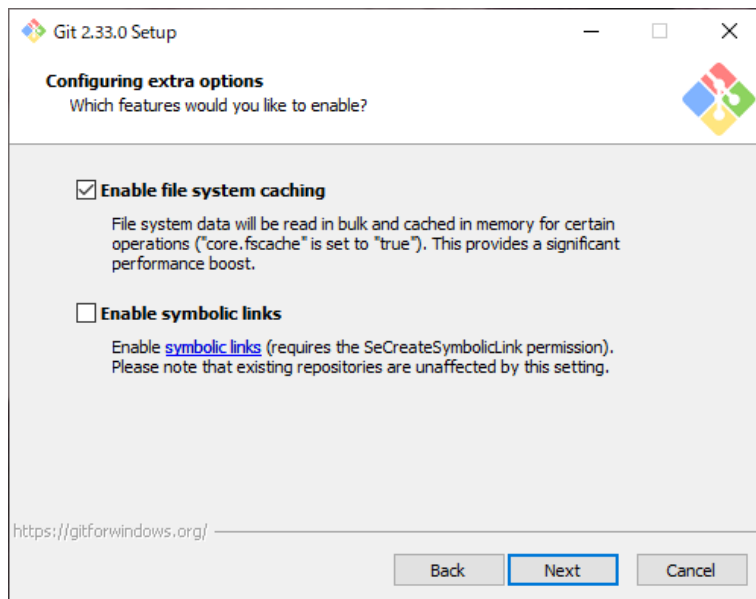
上の Use MinTTY (the default terminal of MSYS2) を選択。



一番上の Default (fast-forward or merge) を選択。

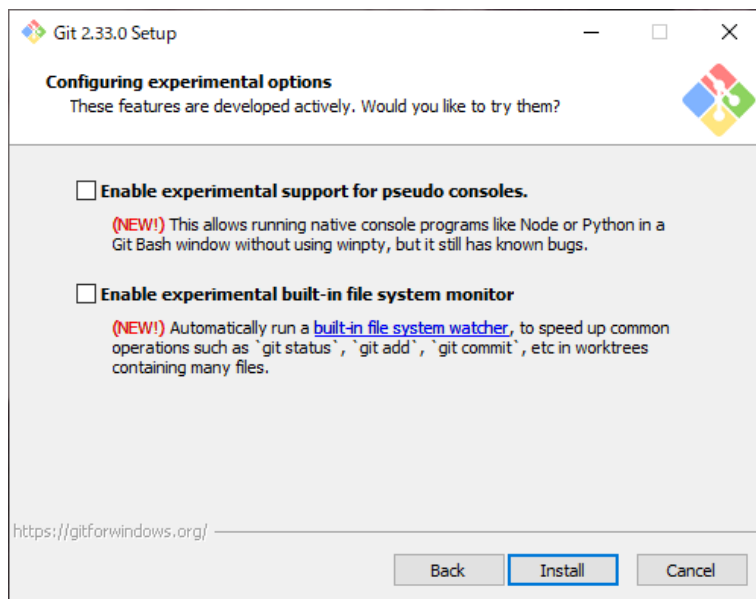


上の Git Credential Manager を選択。

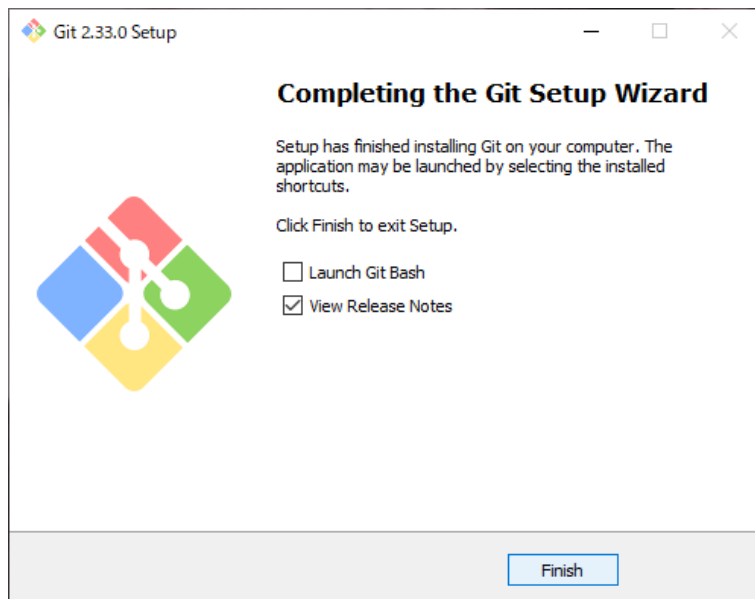


上の Enable file system caching にチェックを入れる。

下の Enable symbolic links にはチェックを入れない。



両方とも、チェックは入れない。



終了。

2.3 参考サイト

- Git インストール手順 < Windows 向け >
- Windows10 に Git をインストール (2021 年 09 月更新)

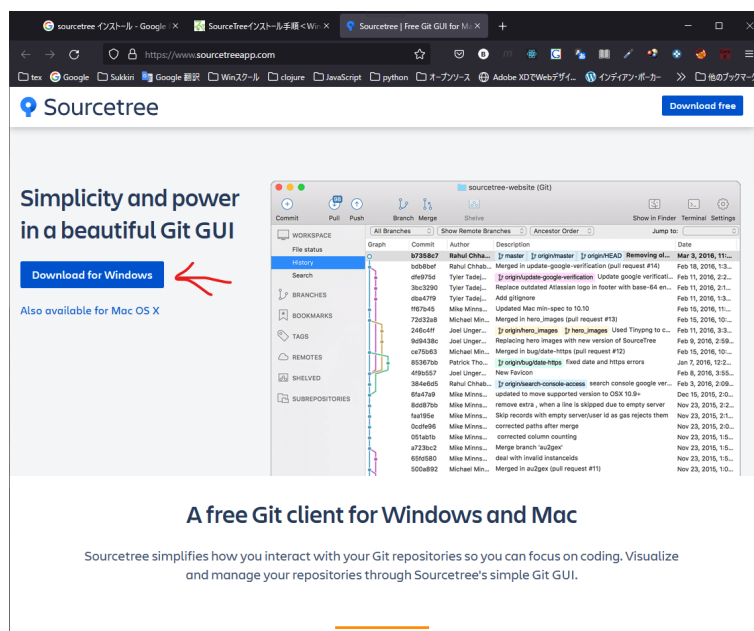
3 SourceTree のインストール

3.1 ダウンロードとインストール

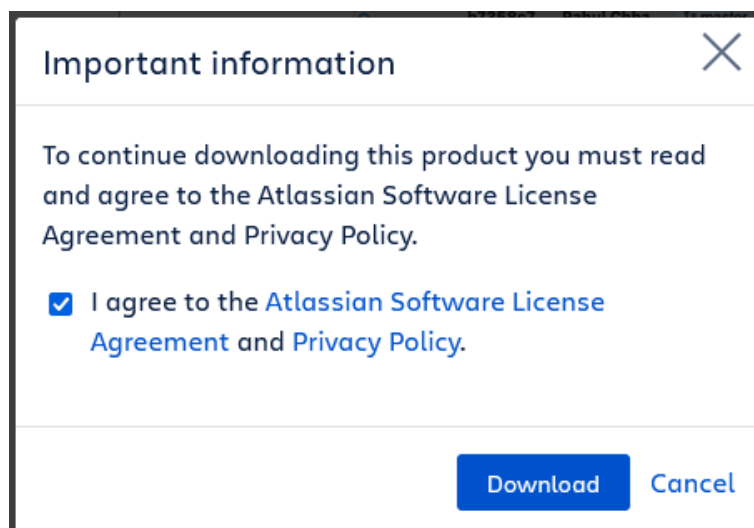
GitHub にアカウントがあるほうが、スムーズにいくかも。

ダウンロードサイト

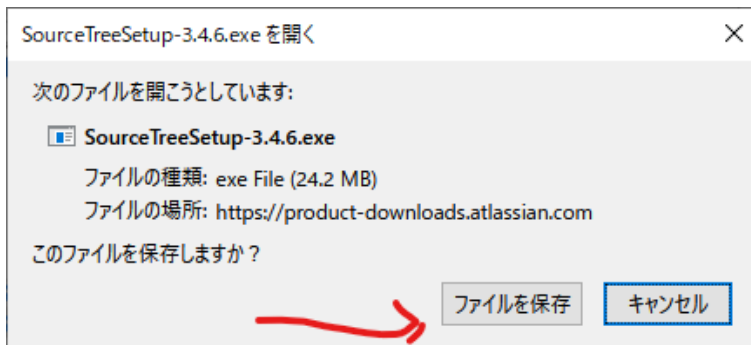
<https://www.sourcetreeapp.com/>



Download for Windows をクリックしてダウンロード開始。



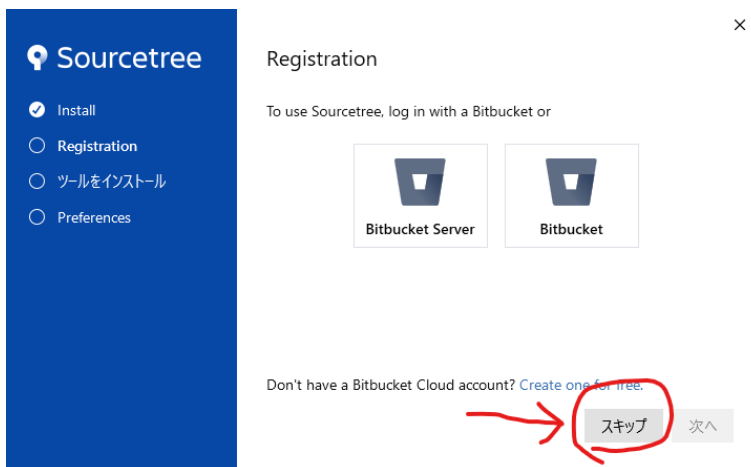
チェックして、ダウンロード。



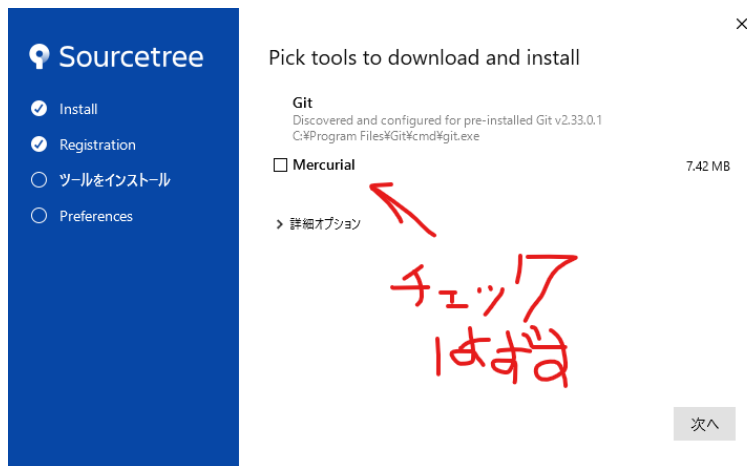
ファイルを保存 とする。



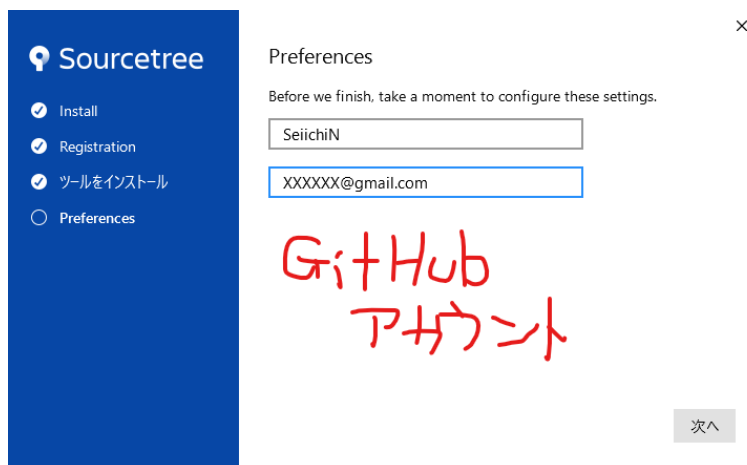
保存されたら、クリックしてインストールを始める。これは Firefox の画面。



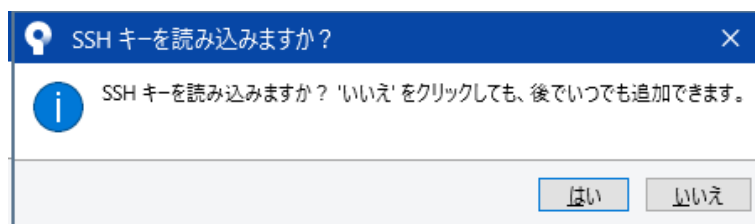
どうも、Sourcetree のログを BitBucket に残す設定みたいだけど、スキップする。
必要なら、あとでも設定できる。



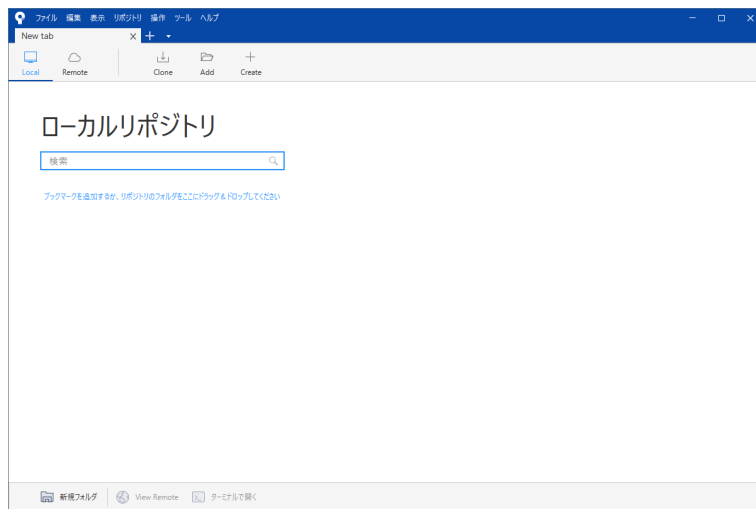
Mercurial のインストールは必要ない。チェックをはずす。



GitHub のアカウント名とメールアドレスを入力する。



SSH キーを読み込みますか? と聞かれたら「いいえ」を選択する。



終了。これが SourceTree の起動直後の画面。

3.2 参考サイト

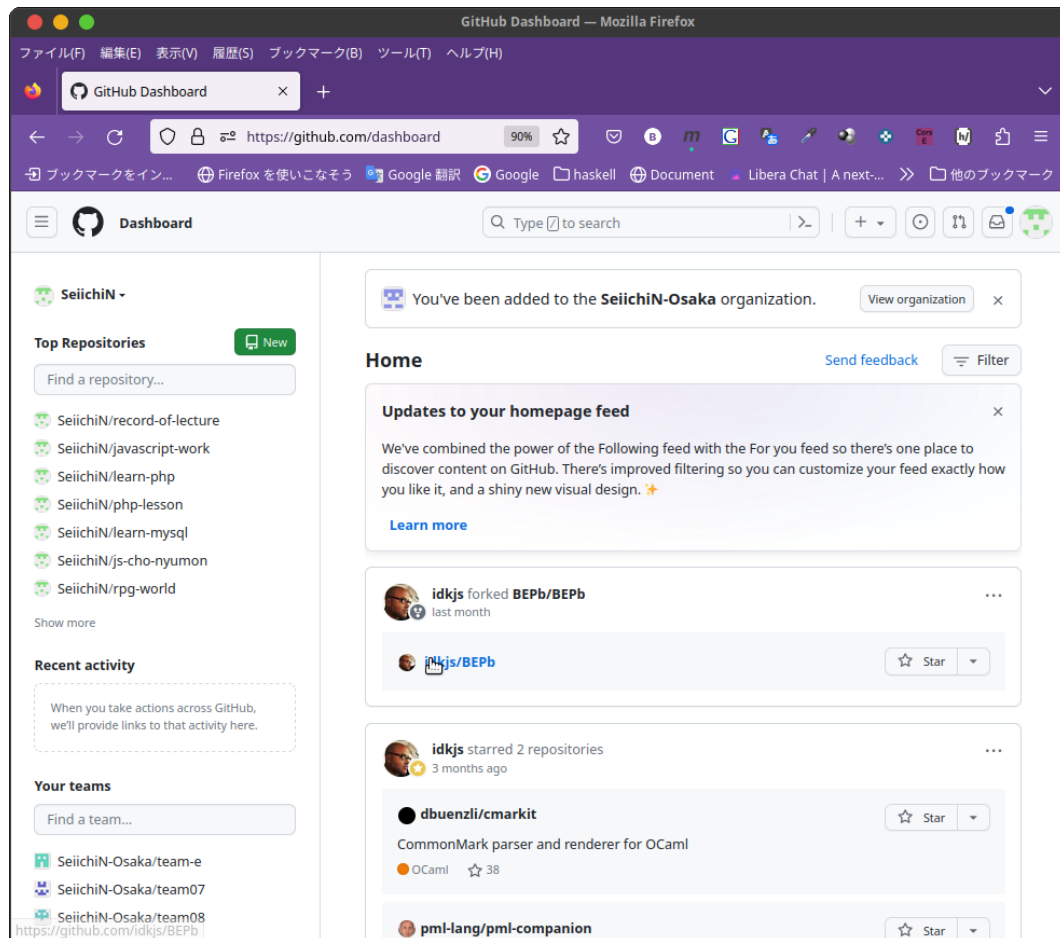
- SourceTree を windows にインストールする
- SourceTree インストール手順 < Windows 向け >

4 とりあえず使ってみる

(GitHub にリポジトリを作成してから、それを自分の PC にクローンする方法)

4.1 GitHub に新しいリポジトリを作る

GitHub にアクセスして、自分のページで「NEW」ボタンをクリックする。



あるいは、このようになっているかもしれない。

Firefox browser window showing the GitHub profile of SeiichiN.

Header: Your Repositories — Mozilla Firefox

Navigation: Overview, Repositories (135), Projects, Packages, Stars

Profile: SeiichiN (Avatar: Green and white geometric design)

Repositories:

- LaLa-work-servlet-jsp** (Private)
gitignoreのテスト
Java Updated 9 hours ago
- learn-mysql** (Public)
MySQLを学ぶ
TeX 1 MIT License Updated 2 days ago
- about-web-server** (Private)
Webサーバーについて
TeX MIT License Updated 3 days ago
- js-cho-nyumon** (Public)
JavaScript超入門
HTML Updated 4 days ago
- record-of-lecture** (Public)

Footer: Seiichi Nukayama, 5 followers · 2 following, billie175@gmail.com, http://nukblog.work/

4.1.1 設定項目

以下のような画面になる。

New repository — Mozilla Firefox

ファイル(F) 編集(E) 表示(V) 履歴(S) ブックマーク(B) ツール(T) ヘルプ(H)

New repository x +

← → ↻ 🔒 https://github.com/new 80% ☆

ブックマークをイン... Firefox を使いこなそう Google 翻訳 Google haskell Document Libera Chat | A next... >> 他のブックマーク

New repository 🔍 Type to search + - 🔍

Create a new repository

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

Required fields are marked with an asterisk (*).

Owner * Repository name * ①

Selichin /

Great repository names are short and memorable. Need inspiration? How about [musical-funicular](#) ?

② Description (optional)

Public ③

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

Private

You choose who can see and commit to this repository.

④ Initialize this repository with:

☒ Add a README file

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

⑤ Add .gitignore

.gitignore template: None

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

⑥ Choose a license

License: None

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

⑦ You are creating a public repository in your personal account.

Create repository

1. Repository name:

英数字を使ってリポジトリ名を記述する。

2. Description:

概要。英字でも日本語でもよい。このリポジトリについての簡単な説明を記述する。

3. Public: このリポジトリをインターネット上に公開する。

Private: このリポジトリをインターネット上に公開しない。最初なので、Private を選択しておく。

4. Add a README file:

README.md を作成する。これは選択しておかないと、リポジトリが表示されない。

5. Add .gitignore:

この選択肢の中にこのリポジトリで使う言語があれば選択しておく。無ければ選択しない。

.gitignore ファイルは後から作成できる。

6. Choose a license:

ライセンス条項 (著作権) の選択。一番緩いのは MIT ライセンス。よくわからなければ、選択せずにおく。

7. Create repository:

これをクリックして、リポジトリを作成する。

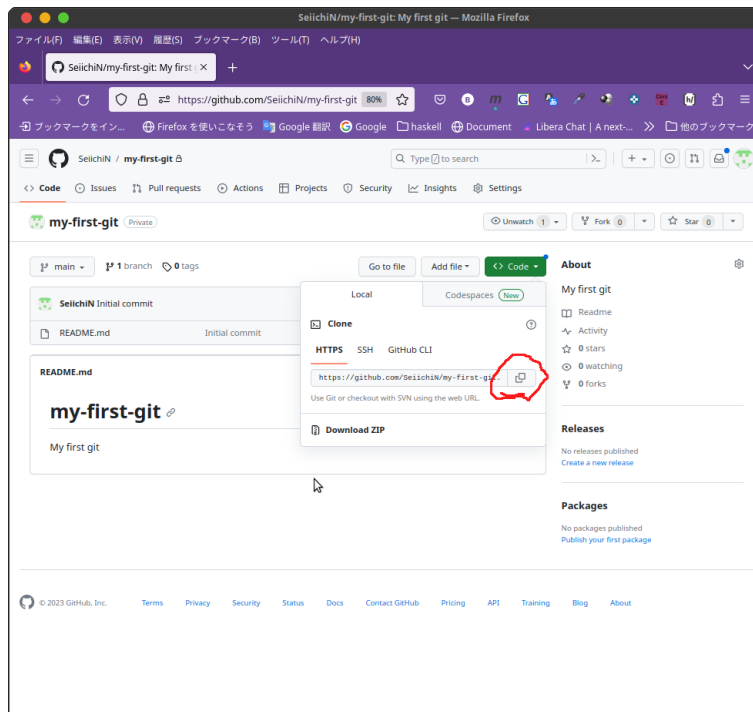
ここでは、以下のように設定する。

1. Repository name: my-first-git
2. Description: my-first-git
3. Private: 最初なので、Private を選択しておく。
4. Add a README file: 選択する。
5. Add .gitignore: 選択しない。
6. Choose a license: 選択しない。

4.2 GitHub からクローン

4.2.1 リポジトリの URL をコピーする

今作成したリポジトリをクローンする。



緑の Code と書かれたボタンをクリックして、'HTTPS' の URL をコピーする。
コピーボタンがあるので、それをクリックすると、簡単にコピーできる。

4.2.2 適当なフォルダでクローンする

クローンするためのフォルダを開く。

どのフォルダの中にクローンするかを決めて、そのフォルダで右クリックし、git bash を選択する。

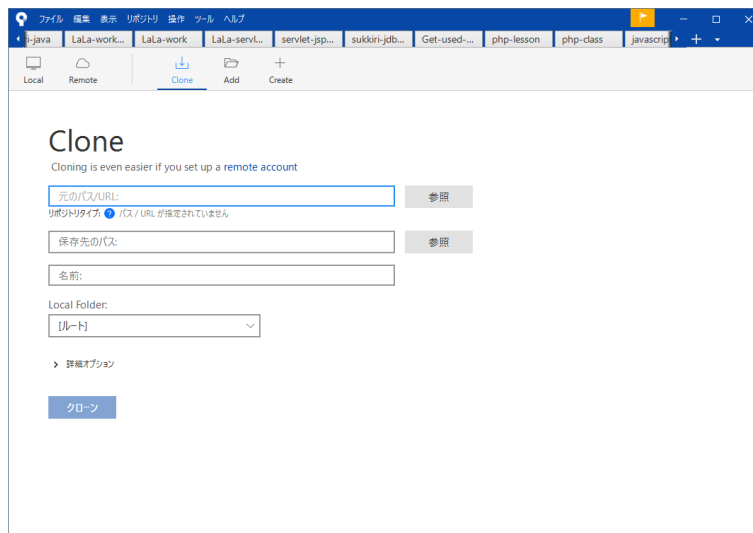
キーボードから 'git clone (半角空白)' と入力し、そのあとに先ほどの URL を貼り付ける。マウスを右クリックで貼り付けられるはず。

```
> git clone https://github.com/SeiichiN/my-first-git.git
```

すると、そこに 'my-first-git' というフォルダが作成される。

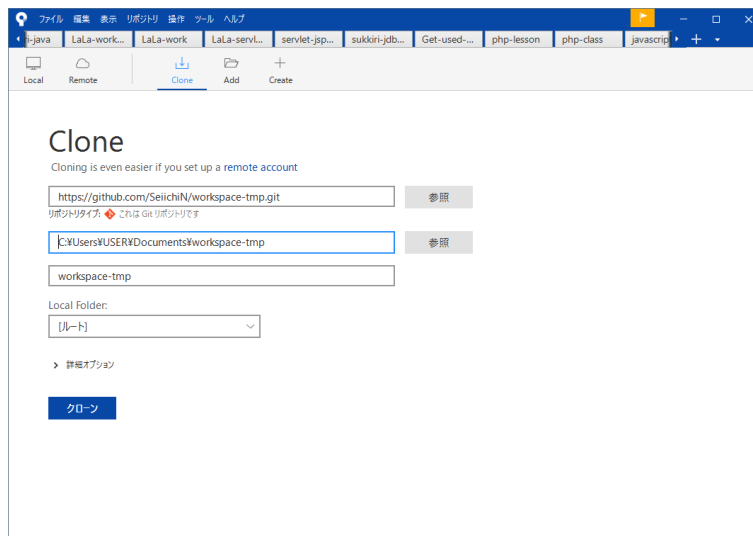
4.2.3 SourceTree でクローンする

SourceTree の場合は、“ファイル” — “新規/クローンを作成する” をクリック。



開いた画面の一番上“元のパス/URL”には、GitHub でコピーしておいた リポジトリのアドレスを貼り付ける。

“保存先のパス”には、クローンするフォルダを指定する。今回は、“my-first-git” というリポジトリをコピーすることなので、たとえば“ドキュメント”フォルダに新しく“my-first-git”というフォルダを作成し、そこにクローンすることにする。



“クローン” ボタンをクリックすると、クローンされる。

4.2.4 git clone コマンドでクローンしたフォルダを SourceTree で開く

“git clone” コマンドを使ってクローンしたフォルダは、SourceTree の“ファイル” — “開く” で、クローンしたフォルダを指定するだけである。ブックマークするか？と聞かれるので、「はい」とする。

4.3 リポジトリの作成とは？

リポジトリとは「倉庫」という意味で、GitHub ではファイル群の置き場所を‘リポジトリ’と呼んでいる。ローカル環境 (自分のコンピュータ) にあるのは‘ローカル・リポジトリ’。GitHub にあるのは‘リモート・リポジトリ’となる。

リポジトリを作成するには、以下の 2 通りが考えられる。

1. GitHub に新しいリポジトリを作成し、現在開発作業をしているフォルダをそのリポジトリと結びつける。
2. GitHub に新しいリポジトリを作成し、それを手元のコンピュータの適当な場所でクローンして、できたフォルダに入って開発作業を始める。

1 番目の方法は、すでに開発作業にとりかかっていて、GitHub にはまだリポジトリを作成していない場合である。

2 番目の方法は、最初から GitHub と連携しながら開発をスタートさせるやり方である。

5 Git とは、どういうものか?

5.1 ネットに接続しないで作業する

今からの作業は、インターネットに接続していなくてもできる。

5.1.1 git 用のフォルダを作成する

適当な場所 (たとえばドキュメントフォルダ) でこれから作業するフォルダを作成する。

たとえば `my-git-first` だとする。

そのフォルダを開いて、そのフォルダの中に、"index.html" を作成する。

適当な内容で作成する。

リスト 1 index.html

```
1 <!doctype html>
2 <html lang="ja">
3   <head>
4     <meta charset="utf-8"/>
5     <title>my-first-git</title>
6   </head>
7   <body>
8     <h1>初めてのgit</h1>
9   </body>
10 </html>
```

5.1.2 そのフォルダで git bash here

"my-first-git" のフォルダの中にマウスポインタを置き、右クリックして、"git bash here" を選択する。

すると、そのフォルダで "git bash" が開く。

"git bash" とは、UNIX コマンドを使えるようにしたコマンドプロンプトのようなものである。

ここで、以下のようにコマンドを実行する。

```
$ git init
```

すると、そのフォルダの中に ".git" というフォルダができる。これは、git システムが使用する大事なフォルダである。

次に、以下のコマンドを実行する。

```
$ git status
```

すると、以下のように応答が返ってくる。

```

billi@DESKTOP-MUEKACK MINGW64 ~/Documents/my-first-git (main)
$ git status
On branch main

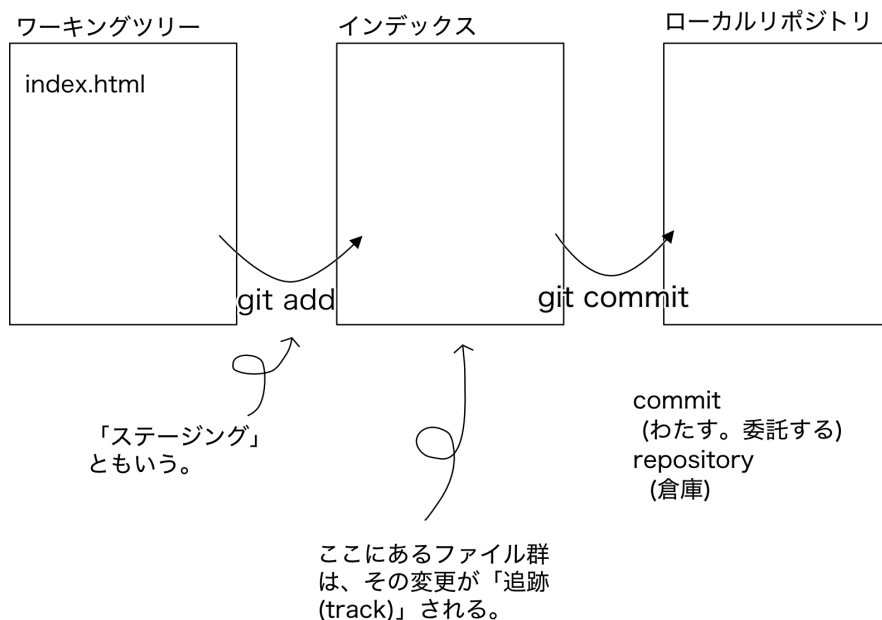
No commits yet

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

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

```

On branch main – 現在は main ブランチである。
 No commits yet – まだコミットしていない。
 Untracked files – 追跡されていないファイル
 (コミットに含めるには git add <files> とせよ)
 index.html – これがまだ追跡されていない
 コミットにつけ加えたものはない。しかし、追跡されていないファイルはある。
 ("git add" をすれば追跡する)



5.1.3 git add

ここで、言われるように git add する。

```
$ git add index.html
```

これは、Git の管理対象として index.html を登録したことになる。このことを " インデックスの対象とする " あるいは " ステージングする " と言う。

```
$ git status
```

git status で情報を見る。

```
billi@DESKTOP-MUEKACK MINGW64 ~/Documents/my-first-git (main)
$ git status
On branch main

No commits yet

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

これで index.html は、管理の対象となった。あとはコミットすると、ローカル・リポジトリへの追加が確定する。

もしも、やはり管理対象から外す (ステージングをキャンセルする) には、

```
$ git rm --cached index.html
```

とする。

5.1.4 git commit

"git add" でファイルの変更をインデックスに追加したら、次に "git commit" でリポジトリにファイルを保存する。

この時、メッセージを添付することが義務づけられている。

このメッセージには、ファイルのどこを修正したのか、なんのために修正したのかなど、あとから振り返ったり、他の人がその変更をみて理解できるようにする。

Enter キーで改行し、複数行でメッセージを書いてもよい。その方がわかりやすい。

```
$ git commit -m "index.html を作成。"
```

あるいは

```
$ git commit -m "index.html を作成。
> 今のところ<h1>要素だけ。
> 画像をつける予定。"
```

と、複数行で書く。

これで、ファイルに加えた変更を Git リポジトリに反映できた。

5.1.5 ファイルを編集する

index.html を編集する。

リスト 2 index.html

```
1 <!doctype html>
2 <html lang="ja">
3   <head>
4     <meta charset="utf-8"/>
5     <title>my-first-git</title>
6   </head>
7   <body>
8     <h1>初めてのgit</h1>
9     
10  </body>
11 </html>
```

それから、git status とする。

```
billi@DESKTOP-MUEKACK MINGW64 ~/Documents/my-first-git (main)
$ git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

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

Changes not staged for commit: すなわち、“コミットに向けてステージングされていない変更”として、

modified: index.html

が表示されている。

“modified” というのは、“変更された” という意味である。

要するに、“index.html が変更されましたが、どうしますか？” と尋ねてきている。

これに対して 2 つの選択がある。

```
git add index.html
```

“index.html” をインデックスに追加する。(ステージングする)

```
git restore index.html
```

“index.html” に加えた変更を破棄する。

ここでは “git add index.html” と、インデックスに追加する。

```
$ git add index.html  
$ git status
```

```
billi@DESKTOP-MUEKACK MINGW64 ~/Documents/my-first-git (main)  
$ git status  
On branch main  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
    modified:   index.html
```

次にリポジトリにコミットする。

```
$ git commit -m "index.html を修正。  
> 画像を追加"
```

git bash で日本語が文字化けする場合

git bash で日本語が文字化けする場合、以下の対処がある。

(1) 左上のアイコンをクリックして、“Options...” を選択。“text” の項目で日本語フォントを選択する。

“locale” は “ja_JP”

“Character set” は “UTF-8” を選択する。

(2) \$ git config --global core.quotePath false を実行する。

これとは別に、“Window” の “UI language” を “ja” にすると、メニューなどが日本語になる。

変更したら、git bash を再起動すると反映される。

5.2 インターネットに接続して、github と連携する

今までの作業は、インターネットに接続する必要はない。また、作業したリポジトリは”ローカル・リポジトリ”と呼ばれる。

これからやる作業は、

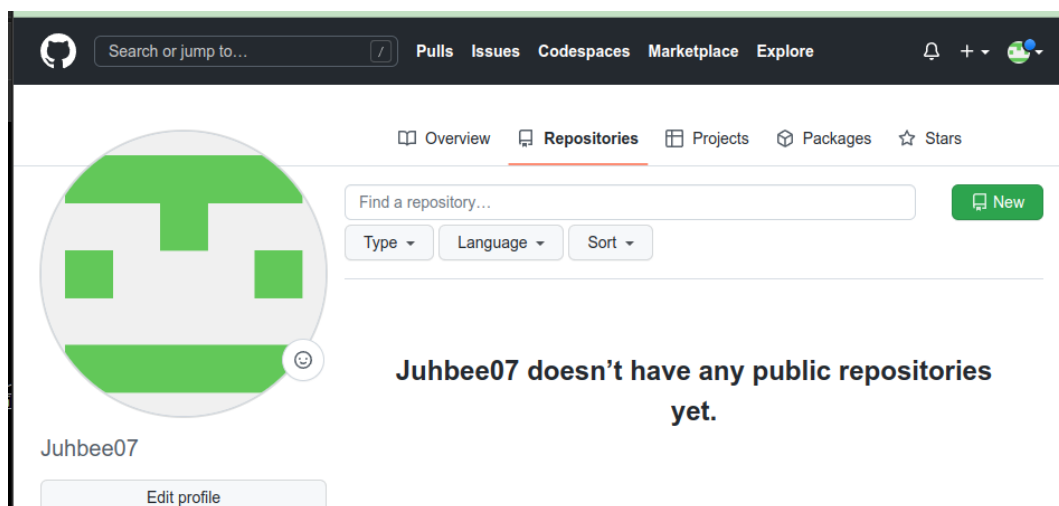
1. github にリポジトリを作成する。(これを”リモート・リポジトリ”と呼ぶ。)
2. ローカル・リポジトリをリモート・リポジトリに反映させる。(同期する)

ということになる。

5.2.1 github にリポジトリを作成する

<https://github.com/<アカウント名>> にアクセスし、sign in する。

new をクリックして、リポジトリを作成する。




“Create a new repository” というページが開く。以下のように入力する。

- Repository Name
first-git
- Description
初めての Git
- Private を選択

Create a new repository

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

Owner *

 Juhbee07 ▾

Repository name *

/ first-git ✓

Great repository names are short and memorable. Need inspiration? How about **psychic-doodle**?

Description (optional)

初めてのGit

☐ Public

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

☒ Private

You choose who can see and commit to this repository.

Initialize this repository with:

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

☐ Add a README file

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

Add .gitignore

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

.gitignore template: None ▾

Choose a license

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

License: None ▾

 You are creating a private repository in your personal account.

Create repository

今回は何もしない

他の項目は、今回は何もしない。その状態で "Create repository" をクリックする。
すると、新しいページが開いて、コマンドがいろいろ表示されている。そのうちの以下の部分を参考にコマンドを実行することになる。

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

```
echo "# first-git" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/Juhbee07/first-git.git
git push -u origin main
```

今回は不要

5.2.2 ローカルリポジトリとリモートリポジトリを同期させる

さきほど作成した "my-first-git" フォルダの中で作業する。

ここには、".git" フォルダと "index.html" の 2 つがあるだけである。

このフォルダの中で マウスを右クリックして "git bash here" を選択する。

"git status" とすると、

```
On branch main
nothing to commit, working tree clean
```

と表示される。また、"git log" とすると、今までの作業のログが表示される。

このローカルリポジトリを今作成した GitHub のリポジトリにアップロードする。

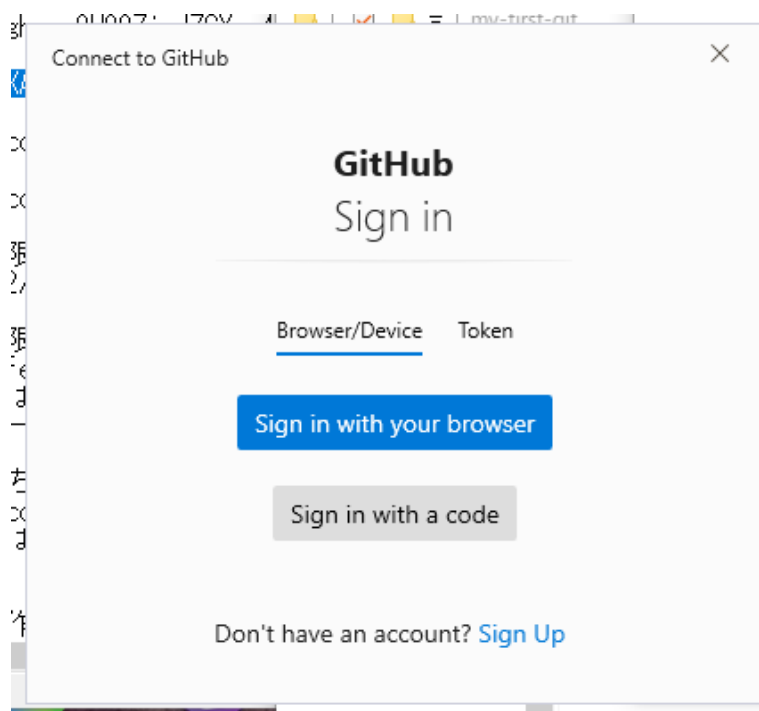
以下の手順でおこなう。これは、github のページに表示されていたコマンドを順におこなうのだが、"git init" は、このフォルダを作成したときにすでに実行しているので、もうしなくてよい。(これは初期化コマンドなので、これを実行すると、初期化されてしまう)

```
1 echo "# first-git" >> README.md
2 git add README.md
3 git commit -m 'first commit to remote repository'
4 git branch -M main
5 git remote add origin https://github.com/<ユーザー名>/first-git.git
6 git push -u origin main
```

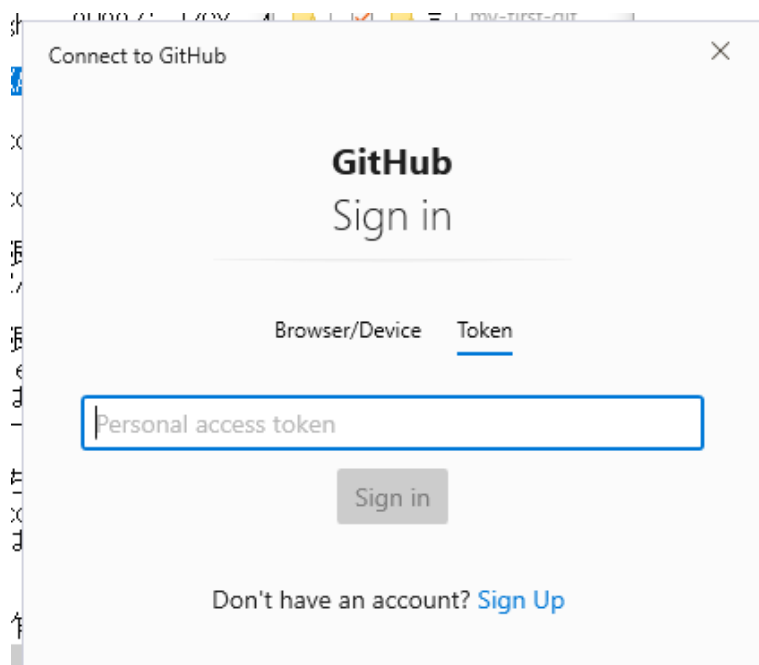
1. README.md というファイルを作成し、"# first-git" という文字列をその末尾につけ加えている。README.md に記述した内容は、github の自分のページに表示される。拡張子 ".md" は、Markdown 形式ということ。
2. git のインデックスに README.md を追加。
3. ローカルリポジトリに追加。メッセージも記述。
4. 現在のブランチを "main" に変更。(もともと main だけど)
5. github.com に作成したリポジトリを "origin" という名前でリモートリポジトリとして登録する。
6. 現在のローカルリポジトリの内容を origin の main ブランチとしてアップする。

最後の "git push -u origin main" を実行すると、Windows の資格情報マネージャーが起動し、以下のよう

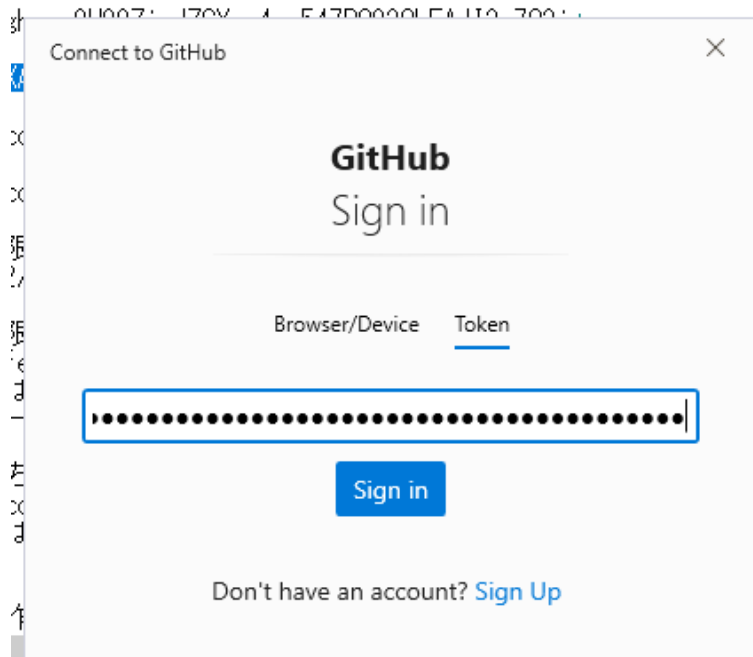
なダイアログが表示されるかもしれない。



この場合は、“Token” タブを選択し、



GitHub の“settings” ページで作成して保存しておいたアクセストークン文字列をここに貼りつける。



これで “Sign in” とすると、ずらずらと文字列が表示され、最後に

```
* [new branch]      main -> main  
branch 'main' set up to track 'origin/main'.
```

と表示され、アップに成功したことがわかる。

Git のローカルリポジトリを作る方法

自分の PC にローカルリポジトリを作成する方法は 2 つある。

(1) まず、自分の PC に作業するフォルダを作成し、“git init” として Git フォルダとする。それから、GitHub にリポジトリを作成して “git remote add origin https://github.com/<username>/<repository name>.git” とする方法。これは今回のやり方である。

(2) まず、GitHub にリポジトリを作成する。それから、適当なフォルダで、“git clone https://github.com/<username>/<repository name>.git” とする方法。こちらの方が簡単である。

6 主なコマンド

6.1 現在の状態を知る

6.1.1 git bash の場合

git status	現在の状態を表示する。もしも、変更がなければ、つまり最新の状態であれば特に何も表示されない。(up to date と表示される) なにか変更されたファイルがあれば、そのファイルあるいはフォルダが表示される。
git status -u	フォルダだけでなく、ファイルも表示される。
git fetch	リモート・リポジトリの状態を調べ、ローカル・リポジトリを更新する。

現在の状態を知るのがまず大事。そのために、git fetch としてから、git status または git status -u とする。さて、いくつかのファイルを編集したり、追加、削除したりして、その状態を GitHub に更新するには、以下のようにする。

6.1.2 SourceTree の場合

SourceTree の場合は、左の"WORKSPACE" で "Hlstory" を選択していると、ログも含め、現在の状況を表示してくれる。

また、更新するには、上の「フェッチ」をクリックするとよい。

6.2 編集・追加・削除をおこなったあと

6.2.1 git bash の場合

git add . (ピリオド)	変更を ローカル・リポジトリに追加する。このコマンドで削除も反映される。
git commit -m "..."	変更内容を記述する。これは記録として残る。
git push origin main	変更をリモート・リポジトリに反映させる。
git log	push がうまくいけば、commit で記述した文字列が git log で表示される。

Git では、ファイル群の管理を 3 つの場所でやっている。

1 つめは、現在のフォルダ。このフォルダをエディタを使って編集している。

2 つめは、ローカル・リポジトリ。現在使用しているフォルダ内に作成されている。このローカル・リポジトリに変更したファイル群を追加するのが、`git add .` である。

そして、このローカル・リポジトリに追加する際に、どういう変更をしたのか、それを記録に残すのが `git commit -m "..."` である。これにはわかりやすく書いておくほうがよい。

それは、Git はチーム開発で使われることが多いので、他のメンバーに対して自分の行った変更を明示する必要があるからである。

また、チーム開発でなくても、自分が行った変更でも時間が経つと忘れてしまうので、自分に対しても変更箇所を明示するほうがよいからである。

たとえば、`git commit -m "index.html のなかのマウスクリックで画像が変更できるようにした。"` など

と書けばよい。

さて、ローカル・リポジトリとリモート・リポジトリは、常に同じ状態を維持する必要がある。しかし、今は、編集作業をおこなって、'git add .' と 'git commit -m "..."' によって、ローカル・リポジトリとリモート・リポジトリに相異が生まれている。つまり、ローカル・リポジトリに加えた変更を、リモート・リポジトリに反映させなければならない。

そのためのコマンドが、`git push origin main` である。

'origin' とは、リモート・リポジトリの別名。'main' とは ブランチ名 である。

GitHub にリポジトリを作成したとき、最初に与えられるブランチ名が 'main' なのである。

少し前までは 'master' だった。ただ マスターだと「主人」という意味にもとれるので、GitHub 事務局で検討され、変更になったらしい。

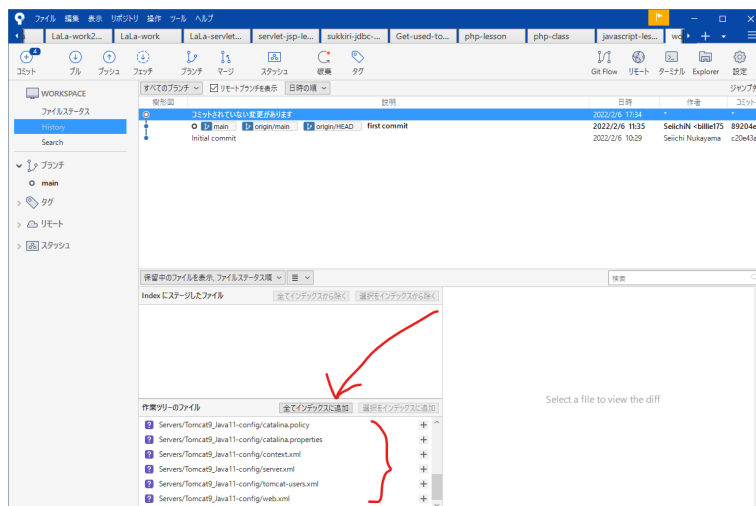
'main' ブランチで開発をすすめる、ある段階である機能を追加することになったとして、その場合、その追加機能は 'ver2' ブランチで作成することとし、'main' ブランチはそのまま開発をすすめる。そして、できあがった段階で 'ver2' ブランチを 'main' ブランチにとりこむことも可能である。

プッシュしたあとは、`git log` で、プッシュがうまくいったことを確認する。

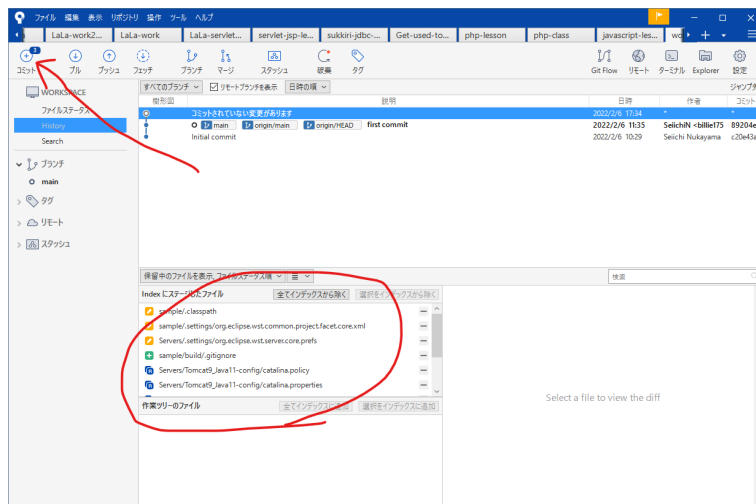
6.2.2 SourceTree で更新作業をおこなう

編集作業を行うと、SourceTree の画面で WORKSPACE を "History" にすると、左下の「作業ツリーのファイル」に変更が加わったファイルがずらっと並んでいる。

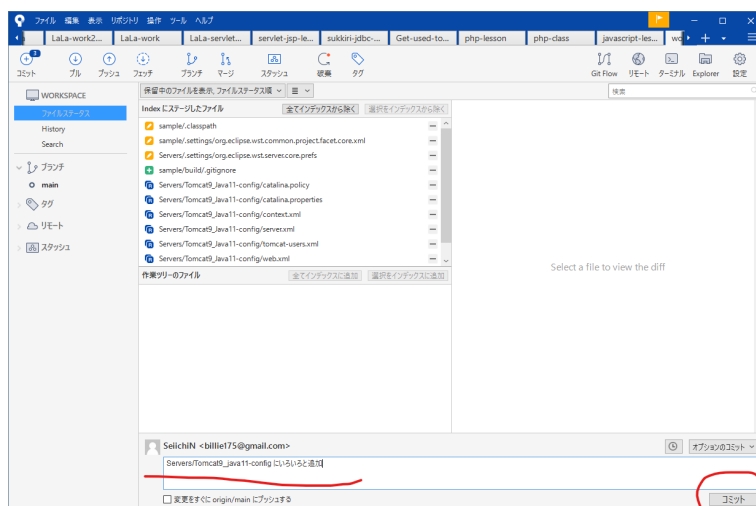
ここで「全てインデックスに追加」をクリックする。これが `git add .` に相当する。



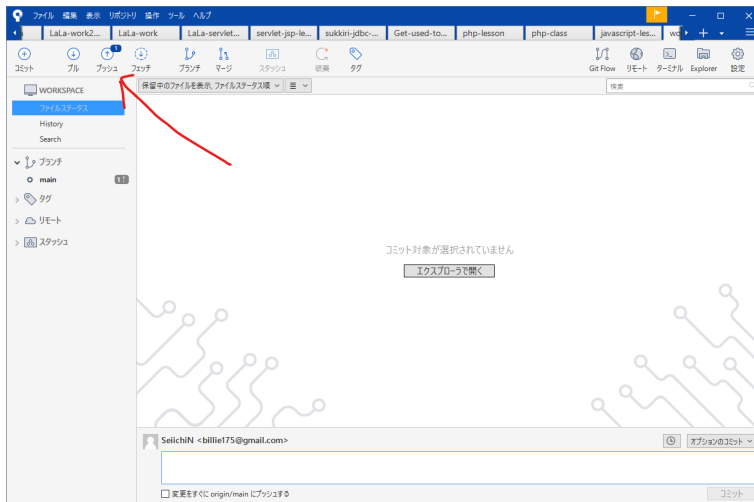
すると、「Index にステージしたファイル」にファイル群が移動する。これが「ステージング」。そして、左上の「コミット」をクリックする。



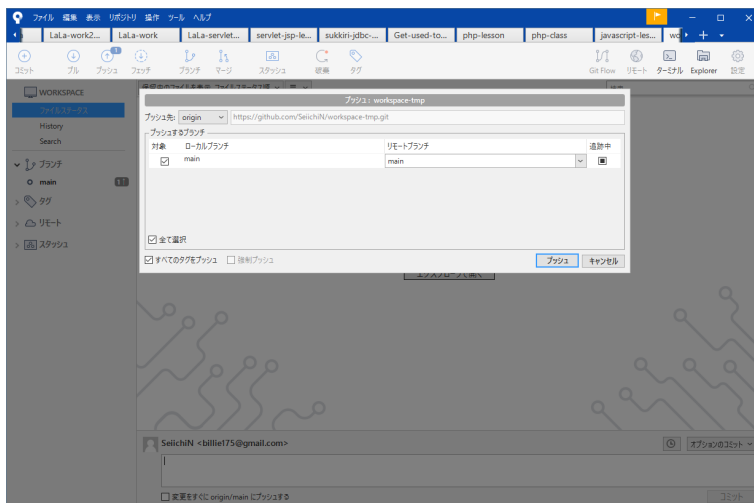
画面下に文字入力できる四角いスペースが現れるので、そこにコメントを入力する。編集した内容がわかるように、コメントを残す。それから「コミット」ボタンをクリックする。これが、`git commit -m "...comment..."` に相当する。



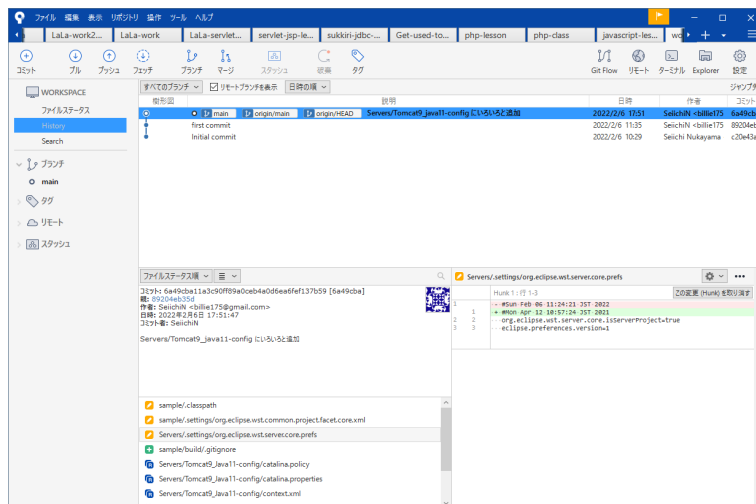
次に、上の「プッシュ」ボタンをクリックする。



すると、「プッシュするブランチ」を尋ねてくるが、たいいていはそのままよい。ブランチを main 以外に作成していないなら、main でよい。



これでリモート・リポジトリにアップされる。



これが、`git push origin main` に相当する。

6.3 リモートの変更を取り入れる

6.3.1 git bash の場合

チーム開発などで、他の人と共同で開発作業を行っている場合、他の人が加えた変更を自分のコンピュータ内に取り入れたい、つまりローカル・リポジトリに反映させたいときがある。

また、個人で開発している場合でも、GitHub を使えば、複数のコンピュータで開発できるので、他のコンピュータで開発していたものを、別のコンピュータに取り込みたいときがある。

そんな場合のコマンドがこれ。

<code>git fetch</code>	リモート・リポジトリに接続し、変更があるかどうかを調べる。もし、変更が無ければ何も表示されない。変更があれば、何やかやと表示される。
<code>git status</code>	現在の状態を知らせてくれる。もし、リモートに変更があり、それを取り込む必要があれば、'git pull' と表示してくれる。
<code>git pull</code>	リモートの変更をローカル (手元の PC) に取入れる。

まず、`git fetch` とするのが大事。リモート情報を更新する。それから `git status` とする。

`git status` では、いろいろな情報を教えてくれる。英語での表示だけど、理解する必要がある。

ローカルよりもリモートが進んでいる場合は、`git pull` でよい。

また、リモートのほうが進んでいるんだけど、リモートにはないファイルがローカルにある場合、上に 'git pull' と表示され、下に その新しいファイルが赤字で表示される。この場合は 'git pull' でよい。ローカルを最新の状態にしてから、'git add .' からのコマンドで 'git push' して、ローカルの新しいファイルをリモートに反映させるとよい。

問題は、同じファイルなんだけど、ローカルでも手を加えていて、別のコンピュータでも手を加えていて、別のコンピュータでの作業がリモートに登録されている場合である。これを Git では コンフリクト (衝突) という。この場合は、二つのファイルが合体され、違っている箇所がマークされているから、手作業で修正していくことになる。

僕はこの作業が邪魔くさいので、手元 (ローカル) の衝突しているファイルをいったん別のファイル名に変えて、それから 'git pull' している。そのようにして衝突を回避してから、二つのファイルを見比べて修正している。

6.3.2 SourceTree の場合

SourceTree では、上の「フェッチ」ボタンをクリックするだけである。もし、リモートに変更があれば、「プル」ボタンに数字が表示されるので、「プル」をクリックする。

7 現在のフォルダを Git 対応にする

現在 `c:\¥myapp` というフォルダで作業しているとする。このフォルダを "Git" の管理下におきたいとする。その場合、以下の順に作業をすればよい。

1. GitHub にこのフォルダ専用のリポジトリを作成する。以下、この myapp フォルダの中での作業となる。
2. `> git init`
3. `> git add .(ピリオド)`
4. `> git commit -m "first commit"`
5. `> git branch -M main`
6. `> git remote add origin https://github.com/ユーザー名/リポジトリ名.git`
7. `> git push -u origin main`

Github で新たにこのフォルダ専用のリポジトリを作成するときは、リポジトリ名と Description、それと public、private の選択だけを指定し、README.md の作成、.gitignore の作成、ライセンスの指定については、何も指定しないようにしなければならない。

そうすれば、上で書いた手順が Github により表示されるので、それに従えばよい。

8 こんな時どうする？

8.1 git log で文字化け

Windows のコマンドプロンプトで 'git log' とすると文字化けする。これは、Windows のコマンドプロンプトの文字コードが 'Shift_JIS (CP932)' だからである。

Windows の 'システム環境変数' に以下を追加する。

```
変数 LESSCHARSET  
値 UTF-8
```

8.2 git status で文字化け

'git status' とすると、日本語のファイル名やフォルダ名が文字化けする。これは、日本語が URL エンコードされるからである。

以下のコマンドを実行する。

```
> git config --global core.quotePath false
```

8.3 リモート・レジストリにアップしたくないファイルがある

作業をしていると、このローカル環境にだけあって、リモート・レジストリに置きたくないファイルが出てくる。たとえば、エディタが自動で作成するバックアップファイルとか。そういうのはチーム開発者たちで共有する必要のないファイルである。また、Eclipse で作業していると、そのローカル独自の Java 環境ができたりする。チーム内で開発環境を統一するとしても。そんな場合はローカルの開発環境の設定ファイル、あるいはプロジェクトの設定ファイルは共有しないほうがよい。

そのような場合、Git では .gitignore ファイルに記述しておけば、そのファイルやフォルダは管理対象から除外される。

記述例

```
*      末尾が のファイルは除外  
pdf/   pdf というフォルダは除外  
*.txt  拡張子が .txt のファイルは除外
```