

「はじめてのGit入門」 ～コマンド不要！GUIで学ぶ履歴管理と共同作業～

2025/10/28

Kazuma Sekiguchi

今回のAgenda

- Gitとは？履歴管理と共同開発の基本
- SourceTreeでできること
- GitHubの役割とメリット
- リポジトリ、コミット、プッシュなどの基本概念
- ローカルリポジトリの作成と履歴確認
- リモートリポジトリとの連携
- ブランチ作成とコンフリクト解消の実践

Gitとは

- リーナストーバルズが1週間で開発したと言われる分散型バージョン管理システム
 - バージョン管理システムとは、ファイルの変更履歴を管理・記録するための仕組み
 - 変更の比較、復元（前のバージョンに戻す）が可能
- バージョン管理を使わないと、新しいファイルは管理できるが、古いものに戻すことはできない
 - ファイル名を変えて保存することで回避できるが、ファイル数が増えるし、管理が複雑になる
 - 誰が作成したのか、誰がどこを変更したのかが分からない

Gitとは

- 分散型バージョン管理システム
 - Linux開発のために必要に迫られて作成したという話
- 履歴管理と共同開発を効率化するツールとして、ほぼ開発でのデファクトスタンダード
 - 開発においてはGitを中心とした開発フローが一般的
- オフラインで作業ができる
 - 常時ネットに接続している必要は無い
 - 旧来のバージョン管理システムでは、常時ネットに接続している必要があるなど制約が多かった

Gitを使う

- Gitを使うことで、ファイルのバージョン管理が可能
 - バージョン=履歴
 - 誰がどこを変更し、いつファイルを登録したのかが分かる
 - 必要に応じて1つ前や2つ前など、自由な状態に戻すことが可能
- 同時に複数の人が同じ開発に携わったとしてもファイルの一貫性を保つことが可能になる
 - 最終的にGitリポジトリに戻すことで、常に最新状態をGitに保持することが可能となる

Gitは2つから成立

- Gitは分散型のため、サーバーがなくてもローカルで履歴管理ができる
 - 一般的にGitはクライアントソフトとか、仕組みのことを指す
 - Gitを多人数で共有し使うためにはサーバー側の仕組みが必要
- Git自体はコマンドラインで動作するプログラム
 - GUIは持たないため、コマンドを打って利用する必要がある
 - 視覚的ではないのと、コマンドを覚える必要がある
- サーバー側はGithubなどのサービスを利用するのが一般的
 - 他のオープンソースのGitLabなどを使い、自分のサーバーにGitのリポジトリを共有する仕組みを構築することも可能

SourceTree

- SourceTreeはGitをGUIで操作するためのソフトウェア
 - 最近ではIDE（開発環境）にGitのGUIが統合されているケースもある
 - SourceTree1つで管理できる点では楽
- SourceTreeを使うと、コマンドを覚えなくてもGitを使うことが可能
 - 複雑なことをするときにはGitコマンドを使えた方がよい



Gitの用語

- Gitの用語は複雑
 - きちんと把握しておかないとGitを使えない

用語	説明
リポジトリ	プロジェクトの履歴管理場所
ステージ(Stage)	コミット前の準備領域
コミット(Commit)	履歴を記録
プッシュ(Push)	履歴をGitのサーバー(GitHubなど)へ送信
フェッチ(Fetch)	リモートの更新だけ確認
プル(Pull)	Gitのサーバー(GitHubなど)から更新(またはデータ)を取得
ブランチ(Branch)	作業の分岐
マージ(Merge)	ブランチの統合

Gitでのワークフロー

- 作業ディレクトリに対してローカルリポジトリを作成
 - ローカルでバージョンを管理するための仕組みを構築
- ファイルを作成したり変更したりする
 - ステージングする（ステージに配置する）
 - 後から見て分かるように、ファイルの変更点などをコメントとして記入し、記録する
- コミットして、ファイルをバージョン履歴に登録する
 - ローカルリポジトリに履歴に登録する
- プッシュして、サーバーのリポジトリに変更に登録する

GitHubの役割

- Githubはリポジトリをオンラインで管理する場所
 - オンラインで管理することで、複数人開発時にリポジトリのデータを他の人がサーバーから取得することができる
 - Githubにおいたリポジトリを他の開発者がプルすることで、ローカルリポジトリに変更を反映させることができる
 - 変更履歴を確認することができる

初回のコミット

- リポジトリに最初のデータを登録する
 - .gitignoreファイルを入れておく
 - Gitで管理しないファイルやフォルダーを指定しておく
 - .envファイルなどの秘密情報や接続情報などを公開しないようにするため
 - 一時フォルダーや一時ファイルなどを指定しておきGitの管理対象外にする
 - この辺まで含めておくとファイルが多数になる上、開発に直接関係の無いファイルまで大量に共有することになる
 - ほとんどの場合は、README.mdなどのファイルも入れてことが多い
 - プロジェクトの説明などを記述しておく
- コミット時には説明を記入しておく
 - どういう変更をしたのかななどを記入

.gitignoreの例

- .envファイルは接続情報やパスワードなどの情報を誤って公開しないために除外する
 - .envファイルに接続情報やパスワードをまとめておくことが多いため、公開してしまうと事故に繋がる
- node_modules/やvendor/などは別の仕組みで各自でローカルだけ導入する
 - ライブラリー用のディレクトリ

```
# OSの不要ファイル
.DS_Store
Thumbs.db

# IDEの設定ファイル
.idea/
.vscode/

# ログファイル
*.log

# 環境設定
.env

# ビルド/依存フォルダ
node_modules/
vendor/
dist/
```

Gitでのフロー

プル

- 最新のファイルを取得する

ファイルを
編集

コミット

- コミットメッセージを記入

プッシュ前に 再度プル

- 競合を確認する

プッシュ

- サーバー側のリモートリポジトリに反映される

Gitでのフロー

- たまに他の人のファイルを取得する
- 作業開始前におこなうのが一般的
- Fetchは更新の確認だけする、Pullは実際にファイルを落としてきて、変更を取り込む



Fetchして更新が無い
かを確認

更新があったら、Pull
してローカルリポジ
トリに反映させる

SourceTreeの見方

ワークスペース
ファイルステータス
履歴
検索

検索

ブランチ
master
readme

タグ

リモート

スタッシュ

すべてのブランチ
リモートブランチを表示
日時の順

作者の名前
ジャンプ先:

説明	日時	作者	コミット
Merge branch 'readme'	2025/10/27 22:54	SEKIGUCHI Kazuma	
マスターブランチ	2025/10/27 22:53	SEKIGUCHI Kazuma	6106c87
GitHub/readme	2025/10/27 22:53	SEKIGUCHI Kazuma	c62bc7c
readme	2025/10/27 22:52	SEKIGUCHI Kazuma	768e6b0
Readmeの追加	2025/10/27 22:52	SEKIGUCHI Kazuma	768e6b0
gitignoreの登録	2025/10/27 22:26	SEKIGUCHI Kazuma	e5b28dd

ファイルステータス順
(1 ファイル)

readme.txt

コミット: 6106c875e9b96e8abf9f122244aff864ddcff2f1 [6106c87]
親: 768e6b052d
作者: SEKIGUCHI Kazuma <sekiguchi@comcent.co.jp>
日時: 2025年10月27日 22:53:45
コミット者: SEKIGUCHI Kazuma

マスターブランチ

readme.txt

ファイル 編集 表示 リポジトリ 操作 ツール ヘルプ

testrepository

コミット プル プッシュ フェッチ ブランチ マージ ステータス 破棄 タグ

Git Flow リモート ターミナル Explorer 設定

樹形図 (リポジトリの履歴)

コミットした人の名前

コミットした内容

コミット時のメッセージ

コミットしたファイル

コミットした人の名前

コミットした内容

コミット時のメッセージ

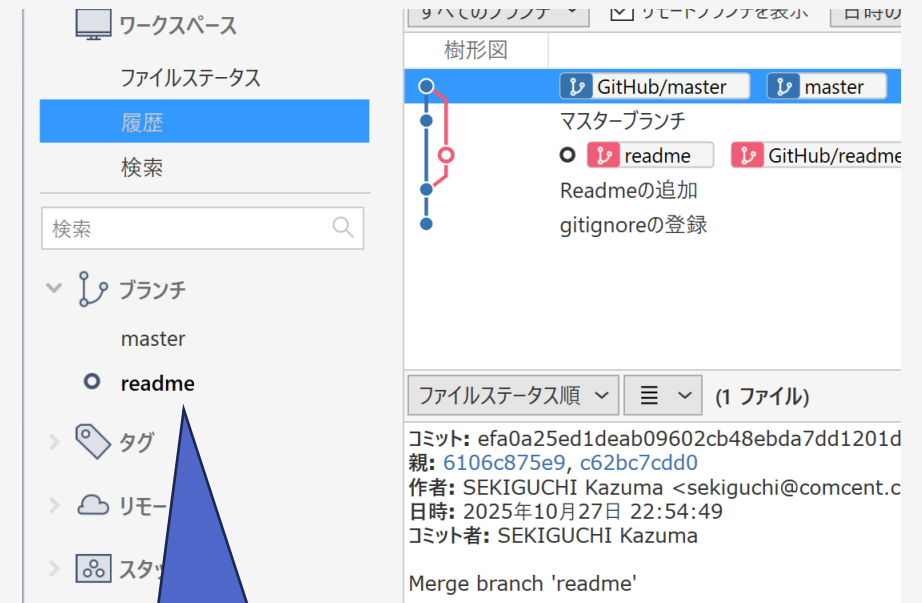
コミットしたファイル

ブランチ

- 通常、最初のコミットはmain (master) ブランチに対しておこなう
 - ブランチ = 作業の分岐
 - 他の人の作業やメインのコードを壊すことなく機能追加をおこなうことが可能
 - 通常、開発はブランチを作成してそこでおこない、ある程度の段階でマスターブランチにマージ（統合）する
- ブランチボタンを押すことでブランチを作成することが可能
 - チェックアウトすることで、ブランチを切り替えることができる
 - ローカルファイルも併せて変更される

ブランチ

- ブランチはリモートリポジトリにも存在する
 - プッシュすることで、共有される
- 自分が現在どこのブランチで作業をしているかは常に意識しておく
 - 同時並行で複数のブランチが走っているのが一般的なので、間違えて他の人のブランチを触っていると後が面倒になる



readmeブランチで作業中

マージ

- ブランチを他のブランチに統合すること
 - 大体はmainブランチにマージするが、他のブランチにマージすることもある



コンフリクト

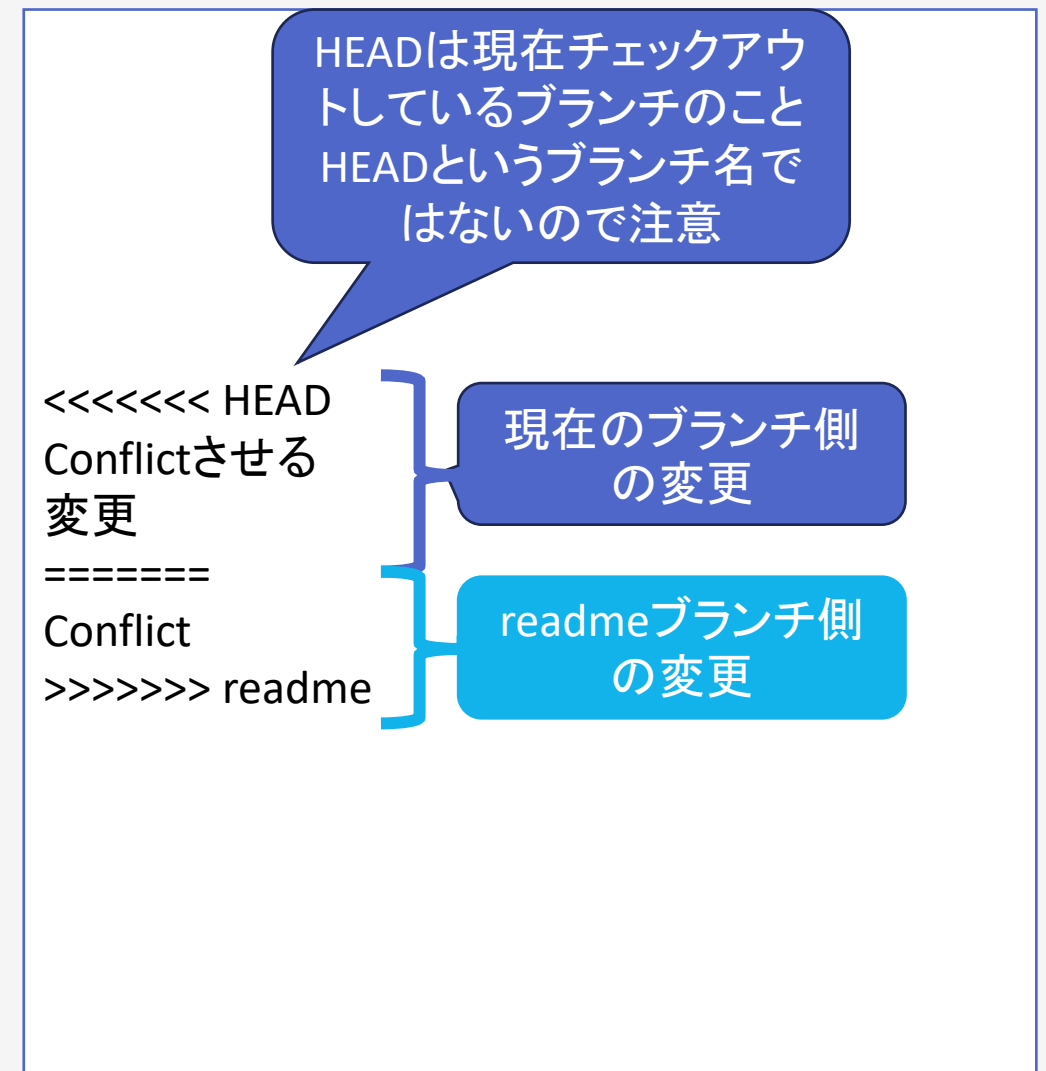
- マージした際に発生する
 - コンフリクト＝衝突
 - 同じファイルの同じ箇所を複数人が変更したときに発生
 - コンフリクト自体は解消する必要がある
 - Git自体は自動的にコンフリクトを解消することはできないが、どちらを採用するか？などの質問をしてくるので、答えればOK
 - どの変更を採用するか、または別の方法を採用かは自由
 - 相手の変更内容を採用する、自分の変更内容を採用するなど

コンフリクト

- 複数人で開発していれば必ず発生する
 - 別に失敗している訳では無い
 - どうしても同じファイルを編集することはある
- どちらを採用するか？をSourceTreeが聞いてくるので、答えればOK
 - 両方とも残したり、改変するなら、外部のマージツールを利用して修正を加える
- できるだけ小まめにコミット、Pullをすることで、コンフリクト自体を減らせる

コンフリクト

- いくつかの方法が存在
- 「自分の変更内容で解決」 や 「相手の変更内容で解決」 を 選択する
 - 中身を確認し、変更内容をよくチェックしてから利用する
 - あとは自動的にGitが処理してくれる



コンフリクト

- 手動でおこなう方法
 - マージツールを利用して手動でコンフリクトを解消することも可能
 - 同じ箇所の変更を見比べて、どれを残しつつコードを活かすかを判断する
 - 外部のマージツールを起動は設定しないと起動しない
 - ファイルを開いて、手動で解消し、「解決とマーク」する



コミットを打ち消す

- 一度コミットしたことを無かったことにすることが可能
 - 打ち消したいコミットを右クリックして、「このコミットを打ち消し」を選択する
 - ただし！ SourcTreeの3.4.26はバグっているのか機能しない
 - Gitコマンドで打ち消すことは可能
- 打ち消しができたら、プッシュをしておく、変更が元に戻った状態になる
 - 戻しました、という記録が残る
- 他にresetもあるが、使い方に注意
 - 履歴を壊す可能性があるので、できるだけ利用しない
 - 特に共同開発においては使わない

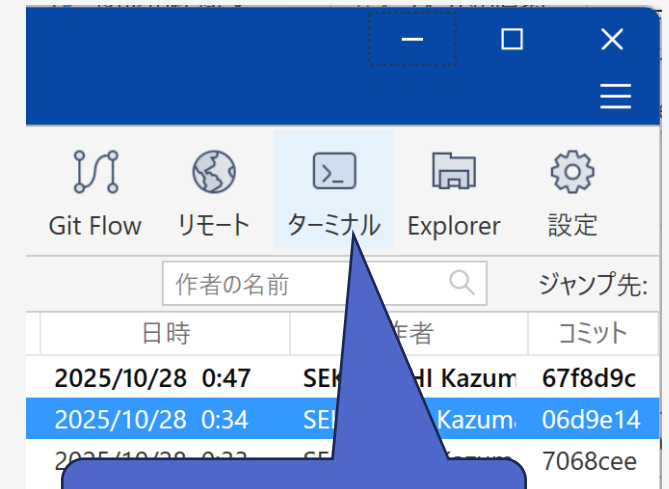
コミットをコマンドで打ち消す

- SourceTreeの右上の「ターミナル」をクリックし

```
git revert --no-edit 06d9e14
```

打ち消したいコミットの番号を
指定する

のように入力



ターミナルをクリック

```
MINGW32:/c/Users/kazum/Desktop/testrepository

kazum@HPKAZUMA MINGW32 ~/Desktop/testrepository (master)
$ git revert --no-edit 06d9e14
[master 67f8d9c] Revert "ファイルを追加"
Date: Tue Oct 28 00:47:45 2025 +0900
1 file changed, 0 insertions(+), 0 deletions(-)
delete mode 100644 sss.txt
```

GitHubとSourceTreeを連携させる

- GitHubはサーバー側、SourceTreeはクライアント側なので、連携させる場合は、認証が必要になってくる
 1. GitHubにログイン後、アカウントのSettingをクリック
 2. サイドメニューの一番下にある、Developer Settingsをクリック
 3. Personal access tokensを開き、Tokens(Classic) を選択
 4. Generate new tokenからGenerate new token(classic)を選択
 5. Noteに適当な名前を入れて、Expirationは有効期限なので、とりあえず30日を選択しておく
 6. repoとuserにチェックを入れてGenerate tokenをクリック
 7. トークンが生成されるので、コピーボタンを押してコピーしておく

GitHubとSourceTreeを連携させる

- GitHubではFine-grained Tokenの使用が推奨されているが、SourceTreeは現時点でFine-grainedに非対応のため、Classicを利用する
- Tokenは必要な権限だけに限定し、有効期限を設定して安全に運用するのがセキュリティ的にも安全
 - 当然Githubのアカウントが乗っ取られたら意味が無いので、2FAなども設定しておいた方が良い
- Tokenは絶対に公開しないこと

GitHubとSourceTreeを連携させる

- SourceTreeでリモートリポジトリの設定をしておく
 - ツール→「オプション」→「認証」で追加ボタンをクリック
 - ホスティングサービスでGitHubを選択、認証でPersonal Access Tokenを選択し、Personal Access Tokenを再読み込みをクリック

The screenshot shows a dialog box titled "ホスティングアカウントを設定" (Set up hosting account) with a close button (X) in the top right corner. The dialog is divided into two main sections: "Host" and "Credentials".

Host section:

- ホスティングサービス:** A dropdown menu with "GitHub" selected.
- ホスト URL:** A text input field containing "https://github.com/".
- 優先するプロトコル:** A dropdown menu with "HTTPS" selected.

Credentials section:

- 認証:** A dropdown menu with "Personal Access Token" selected.
- ユーザー名:** An empty text input field.
- Personal Access Token を再読み込み:** A button to refresh the token.
- Need help logging into your account?:** A blue hyperlink.

At the bottom right of the dialog are two buttons: "OK" and "キャンセル" (Cancel).

GitHubとSourceTreeを連携させる

- ユーザー名でGithubでの名前を入力、パスワードに先ほどコピーしたPersonal Access tokenを指定する
- 成功すれば、「認証に成功」と出てくる

Windows セキュリティ

Sourcetree Personal Access Token request

Enter your PAT as the password for https://github.com/.

ユーザー名
ユーザー名

パスワード
パスワード

OK キャンセル

こちらには
Tokenを入力

ホスティングアカウントを設定

Host

ホスティングサービス: GitHub

ホスト URL: https://github.com/

優先するプロトコル: HTTPS

Credentials

認証: Personal Access Token

ユーザー名: SKAZUMA87

Personal Access Token を再読み込み

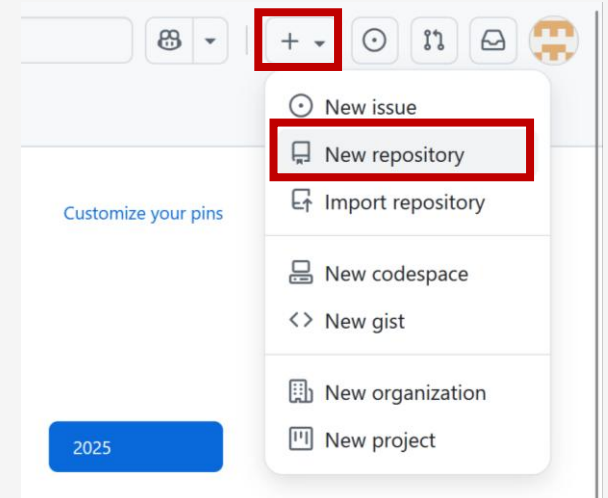
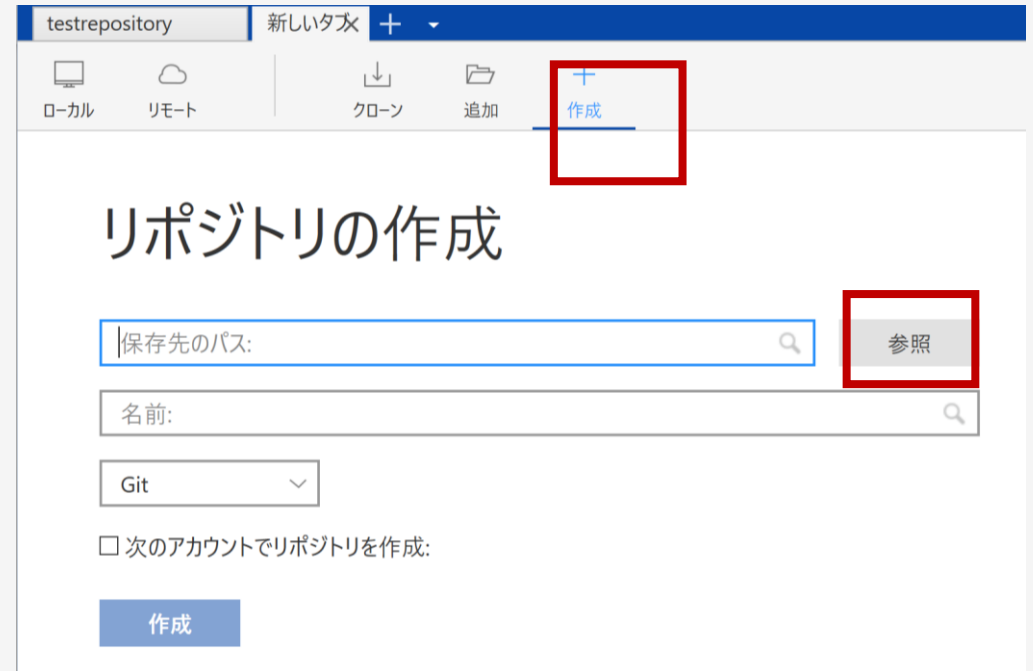
[Need help logging into your account?](#)

✓ 認証に成功

OK キャンセル

手順

1. 開発用のフォルダーを作成しておく
2. SourceTreeで「作成」から作成したフォルダーを指定してローカルリポジトリを作成する
3. 他の人と共有するなら、GitHub上でリポジトリを作成しておく



手順


4. 必要事項を記入して、リモートリポジトリを作成

- 必要に応じて、Privateリポジトリにすることも可能
- Publicリポジトリは誰でも閲覧可能

Create a new repository

Repositories contain a project's files and version history. Have a project elsewhere? [Import a repository](#).
Required fields are marked with an asterisk (*).

1 **General**

Owner *  SKAZUMA87 / Repository name *

✔ t2 is available.

Great repository names are short and memorable. How about [musical-octo-broccoli](#)?

Description

0 / 350 characters

2 **Configuration**

Choose visibility *

Choose who can see and commit to this repository

Add README ☐ Off

READMEs can be used as longer descriptions. [About READMEs](#)

Add .gitignore

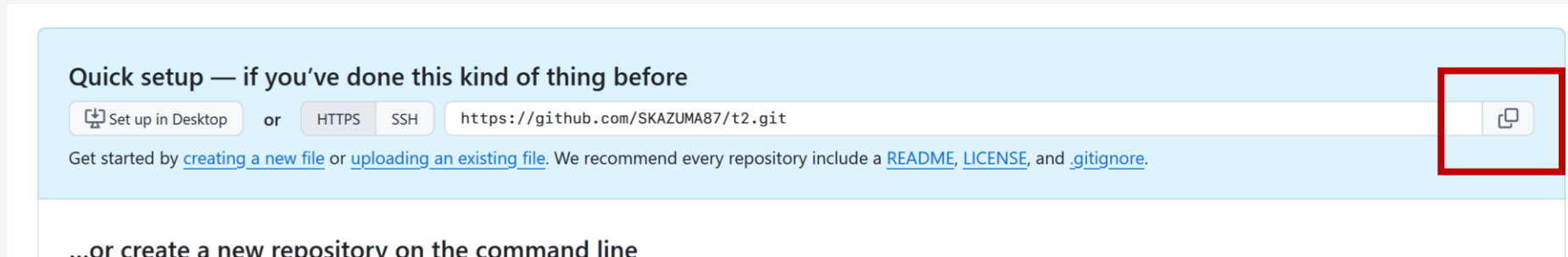
.gitignore tells git which files not to track. [About ignoring files](#)

Add license

Licenses explain how others can use your code. [About licenses](#)

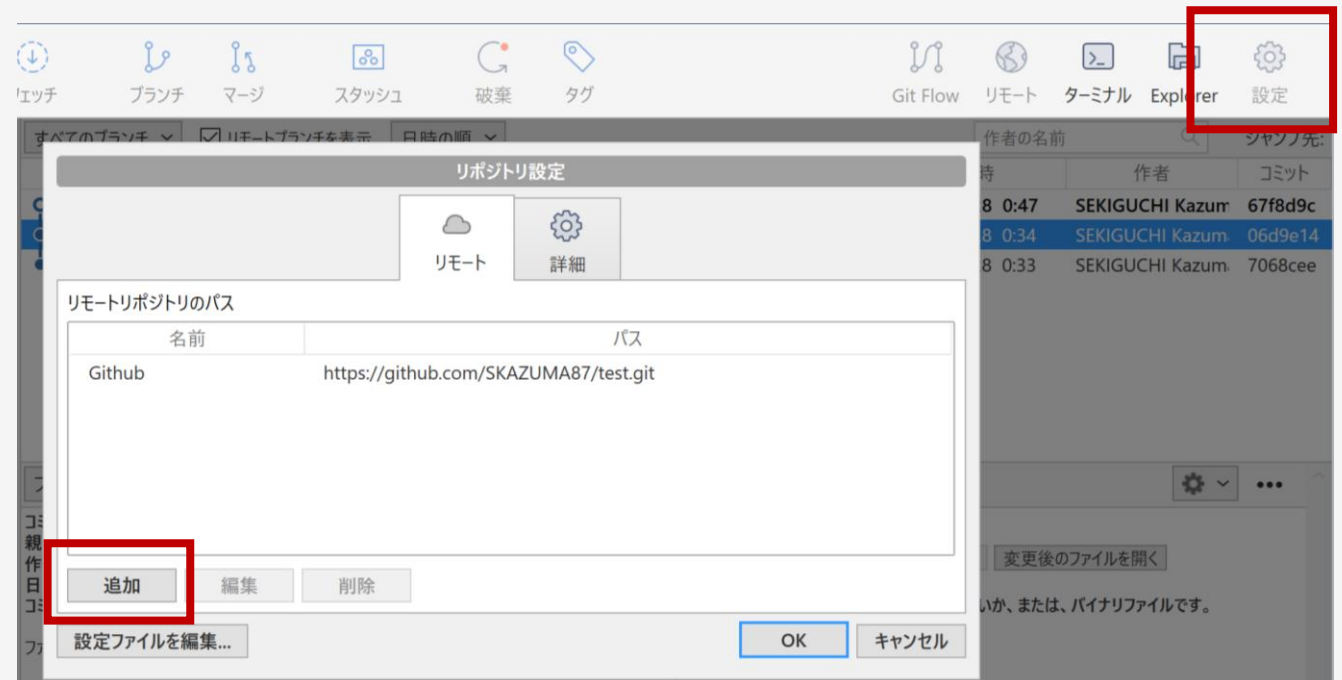
手順

5. URLをコピーしておく



6. SourceTreeの設定からリモートを選択し、追加でコピーしたURLを貼り付けておく

- リモート名は適宜付ける

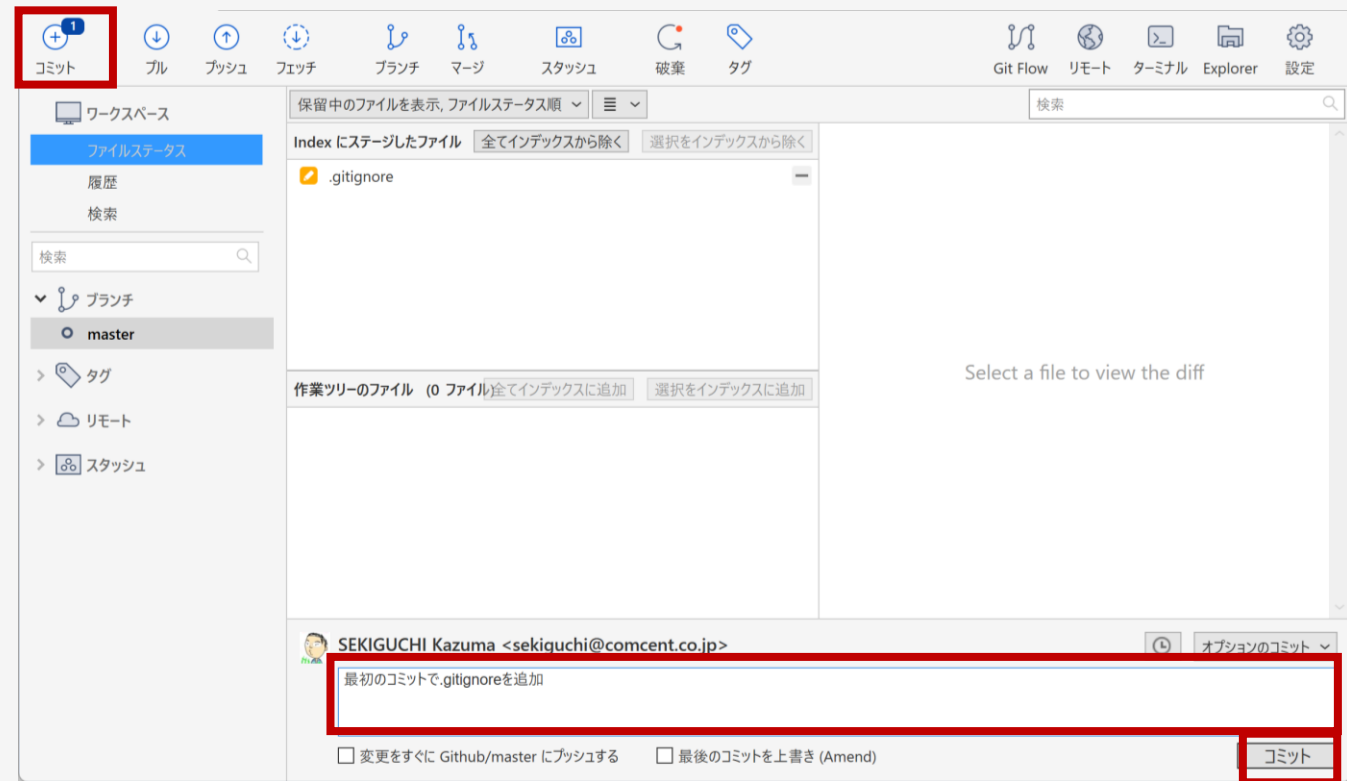


手順

7. これでローカルリポジトリとリモートリポジトリが繋がっている状態になるので、あとはファイルを作業フォルダーに作成する

- 基本的には.gitignoreを作成

8. 作成したら、コミットステージングしてコミットメッセージを記入後、コミットし、プッシュしておく

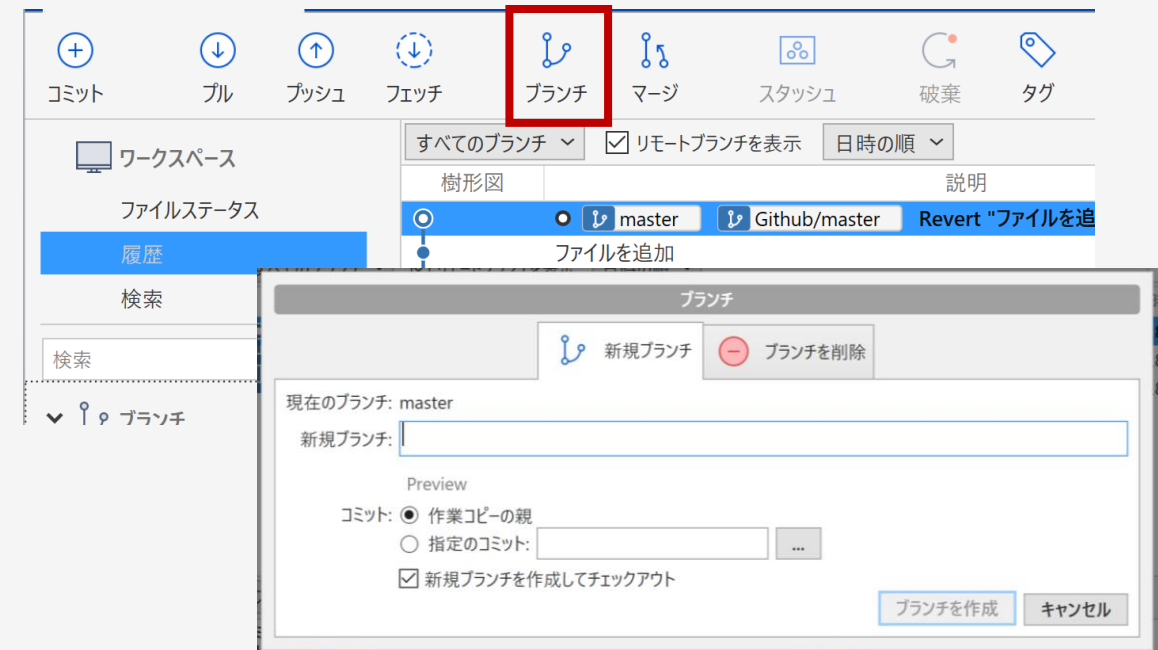
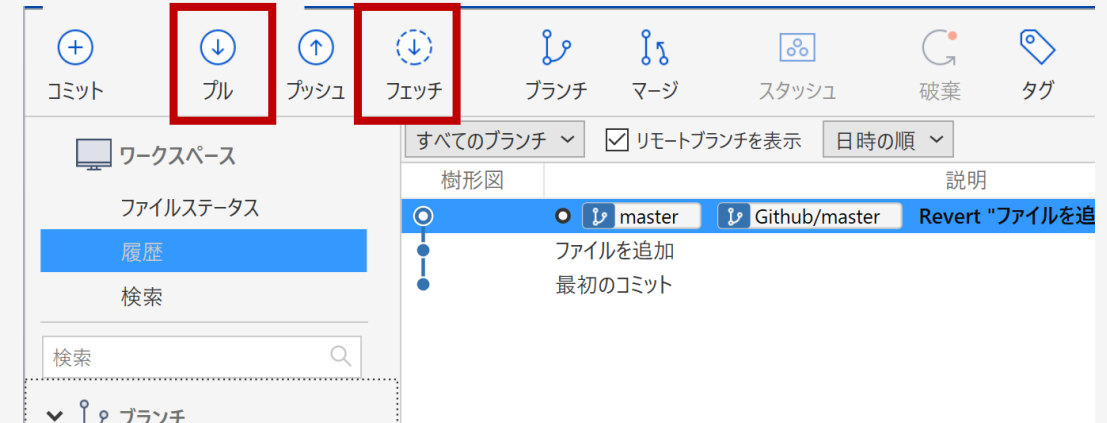


手順

9. 必要に応じてFetch,Pullを利用する

10.開発時はブランチを作成して、作業はそちらでおこない、完了したらマージをおこなう

11.コンフリクトを起こしたら、Gitの相手側または自分側で解決や自分で解決をおこなう



ありがとうございました。

Gitは今の開発に欠かせないツールです。

比較的慣れるまで言葉や操作に悩むところもあると思いますが、周りに聞くなり、調べるなりしていくことで、慣れていきます