



Git入門！

SourceTreeを使って1人でGit運用してGitに慣れてみる

Git の必要性を理解する

Web サイト制作・アプリケーション開発をしていると、ソースコードの追記・修正をするなんていうことは日常茶飯事にあります。修正といっても、



- バグ対処のための修正
- 制作物の仕様の変更による修正
- 新機能のリリースのための追記・修正

など、様々な状況のソースコード修正が存在します。

Git の必要性を理解する - git でできること -

様々な状況が想定される制作作業において、ファイルの変更管理を簡単に便利に行うことができるのが、**バージョン管理システムである Git** です。



- **ファイルの変更履歴を管理することが出来る。**

- **変更履歴を管理することが出来る**

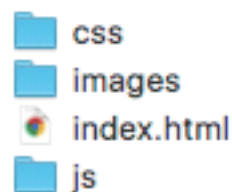
→ 過去の状態へファイル履歴を戻すことが出来る。

- **変更履歴を管理することが出来る**

→ ソースコードのバックアップとして活用することも出来る。

- **他の作業者とソースコードの共有・共同管理が出来る**

Git の必要性を理解する - Git がない場合？ -



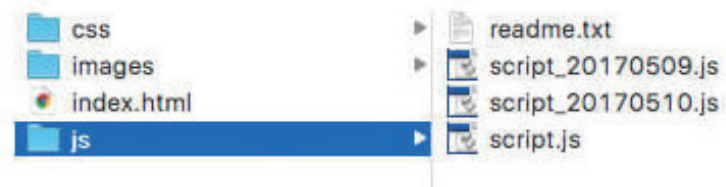
例えば、左記のようなシンプルプロジェクトを **1人で作成している** としてお話を進めてみましょう。

制作・開発が進んで、制作物の仕様追加・変更などがあったりした時に、js フォルダの中に入っている「script.js」という JavaScript ファイルに新しいプログラムを追記していったり修正をしていった場合、

「プログラムのいつ何を変更・追加・削除をしたのか（誰が作業したのか）」

を記録する時は、どのような記録方法が考えられるでしょうか？

Git の必要性を理解する - Git がない場合？ -



- readme.txt に「誰がいつ何を作業（変更・追記・修正など）をしたのか」をメモ。あるいはソースコードにコメントを書いていく。

- 日付ごとにファイルを都度保存していく。あとでバージョンを戻したくなった時は、readme.txt に書いたメモを参照し、ファイル名を変更してバージョンを戻す。

上記のような方法があげられるでしょう。

- readme.txt に「誰がいつ何を作業（変更・追記・修正など）をしたのか」をメモ。あるいはソースコードにコメントを書いていく。

- 日付ごとにファイルを都度保存していく。あとでバージョンを戻したくなった時は、readme.txt に書いたメモを参照し、ファイル名を変更してバージョンを戻す。

例)

5/10のバージョンに戻したい場合

- 現在のscript.jsを削除
- script_20170510.jsのファイル名をscript.jsに変更

面倒&手間がかかる !!!

Git の必要性を理解する - Git がない場合？ -

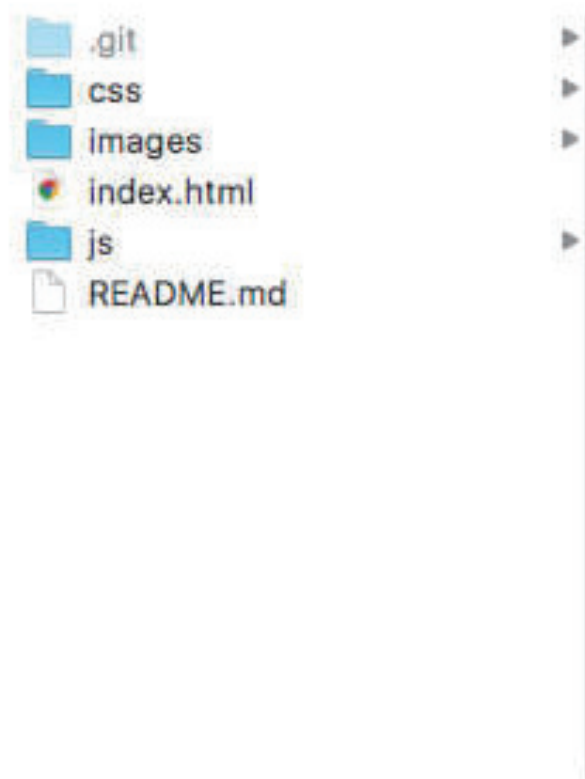


ファイル単体の変更ならまだしも、Web サイト全体をリニューアルとはいかないまでも、

結構なソースコードの量の変更があった時は、
大量のファイルをコピー→保存しなければならず、無駄なファイル容量が増えるばかり・・・

Git の出番だ !!!

Git でファイル変更記録管理をする



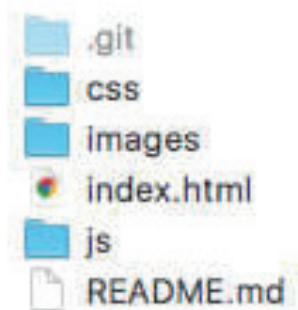
git を使用すると、作業フォルダ内に「**リポジトリ**」というものが作られます。この場合、

- ・ 日付ごとにファイル保存の必要がない。
- ・ フォルダ内に「**リポジトリ**」というものが作成され、「**変更履歴→各種ファイル & 情報**」は全てここに保存される。

上記の特徴があります。

先ほどの事例でいえば、JS ファイルを日付ごとに保存しておく必要もなくなるのです！

2つのリポジトリ「ローカルリポジトリ」



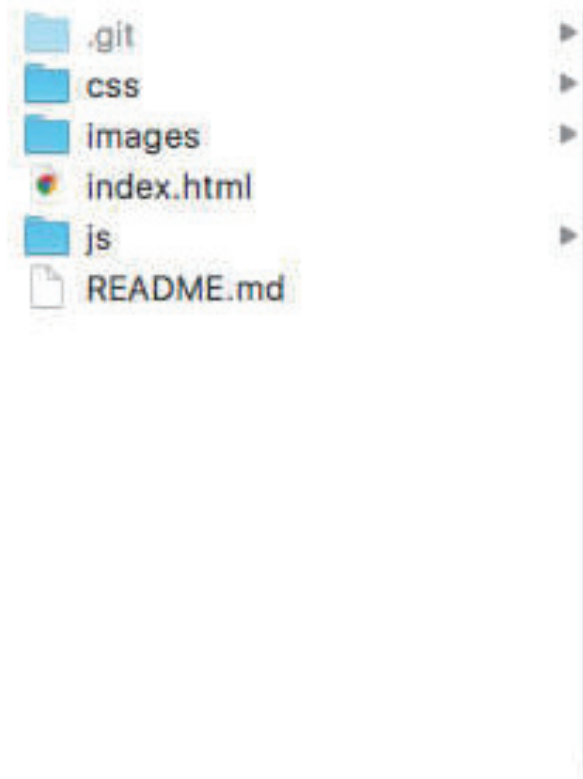
■ローカルリポジトリ

自分の PC 内の作業フォルダに作成されるリポジトリが「**ローカルリポジトリ**」と思ってもらって OK です。

PC 内で制作・開発作業していくだけでも各種ソースコード・ファイルの変更・追記・修正は常々発生していくものですよね！

そういった「ソースコードの変更・追記・修正」は、ローカルリポジトリで適宜管理することができます。

2つのリポジトリ「リモートリポジトリ」



■リモートリポジトリ

共同作業者とソースコードを共有しあったり、
自分の制作したプログラムコードを公開したり
・・・インターネット上に作成しておくリ
ポジトリのことと思ってもらって OK です！

※代表例

GitHub

Bitbucket

ローカルリポジトリでバージョン管理

Source Tree の使い方・超基本！



■デスクトップ上に移動

Source Tree のダウンロードを済ませた状態から話を進めていきますので、まだの方は Source Tree のダウンロードをしましょう。

ダウンロードが終わり、アプリケーションを開くと、左記の画面が出てきます。

DeskTop 上に今日作業したデータのフォルダを置いて、赤枠部分にドラッグ&ドロップします。

ローカルリポジトリの作成



■ローカルリポジトリ作成

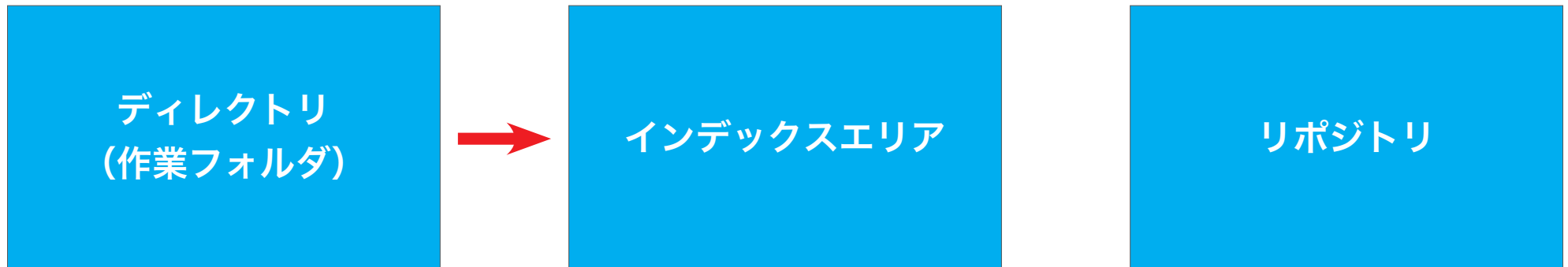
ローカルリポジトリを作成し、タイプを Git にし、保存先のパスが先ほど Desktop 上に置いたフォルダになっていることを確認して作成ボタンを押します。

うまく行くと、元の画面に戻り、左記のようになっていると思いますので、新しくできたリポジトリというものを選択してみましょう。

変更履歴を記録するまでの流れ



変更履歴を記録するまでの流れ -add-



※ターミナル操作の場合

`git add` ファイル名・・・何のファイルをインデックスエリアに登録するのかを指定

変更履歴を記録するまでの流れ -commit-

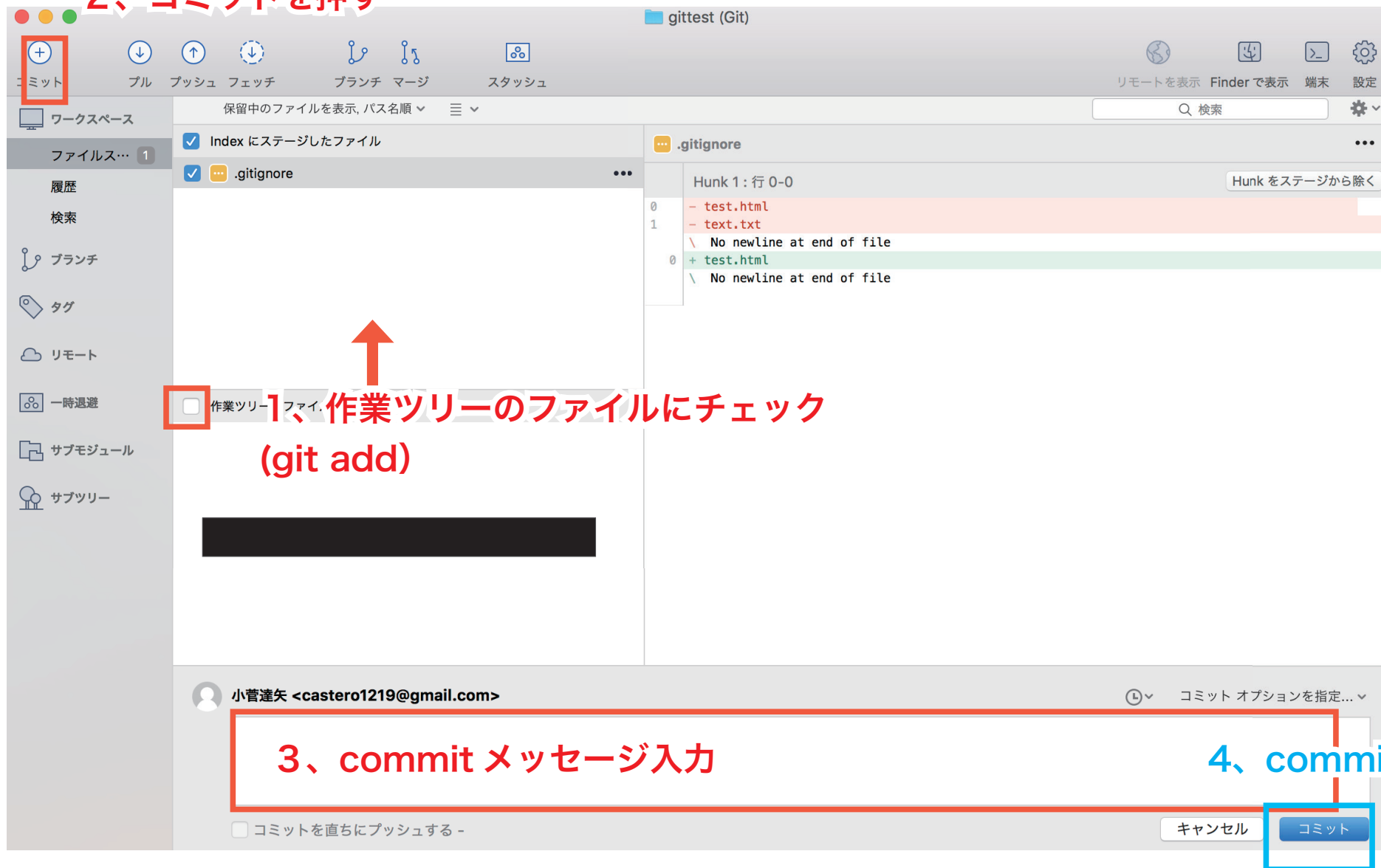


※ターミナル操作の場合

`git commit -m "メッセージ内容"`・・・何のための commit なのかをメッセージ内容に記入した上で、実際にリポジトリに変更履歴を記録する。

変更履歴を記録するまでの流れ on SourceTree

2、コミットを押す



演習



ひたすらファイルを編集してみて、何度も
何度も commit をしてみましょう！

リモートリポジトリにプッシュ

リモートリポジトリへプッシュ



1人で作業していて、情報を記録することができるようになったら、次はリモートリポジトリへのプッシュをしてみましょう。

とある Web サイトを、AさんとBさんが作ることになり、**今回はリモートリポジトリの置き場として GitHub を活用することにした**、という状況を想定し、お話を進めていきたいと思います！

ちなみに今回のようなケースの場合、Gitを紹介している書籍などでは、GitHubなどに作成するリモートリポジトリを「**中央リポジトリ**」と表現していたりもしますね！

リモート リポジトリ

・ GitHub、BitBucketなど



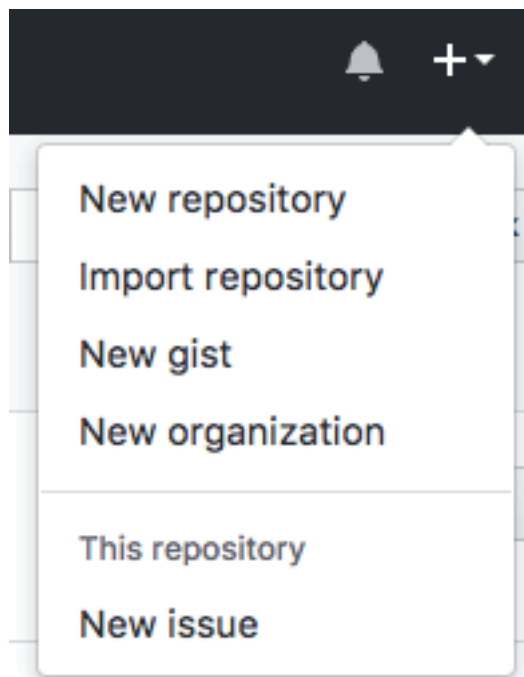
Aさん

AさんとBさんは同じWebサイトを
共同で制作することになった!



Bさん

リモートリポジトリの作成



GitHub アカウントを取得すると、ページの右上あたりに「+マーク」があるかと思います。


「+マーク」をクリックすると、「**New Repository**」というところがあるので、こちらをクリックします！

リモートリポジトリの作成

Create a new repository

A repository contains all the files for your project, including the revision history.

Owner

 castero1219 ▾

Repository name

リポジトリ名を入力

Great repository names are short and memorable. Need inspiration? How about **reimagined-dollop**.

Description (optional)

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

リモートリポジトリ名を公開にするか、
非公開にするか。月額7ドルで無制限の
非公開リポジトリが作成可能。

☐ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: **None** ▾

Add a license: **None** ▾



■ gitignore・・・Gitで管理したくない
ファイルの拡張子やフォルダを指定。

■ license・・・ソースコードの三次利用
など関連の定義。(MIT Licenseなど)


Create repository

※ initialize read me のチェックは、
今回は外してください。

試しに、**githubtest** というリモート
リポジトリを作成したいと思います。

リモートリポジトリの設定

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

 Set up in Desktop or **HTTPS** **SSH** `https://github.com/castero1219/cheeseacademy.git`

We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).

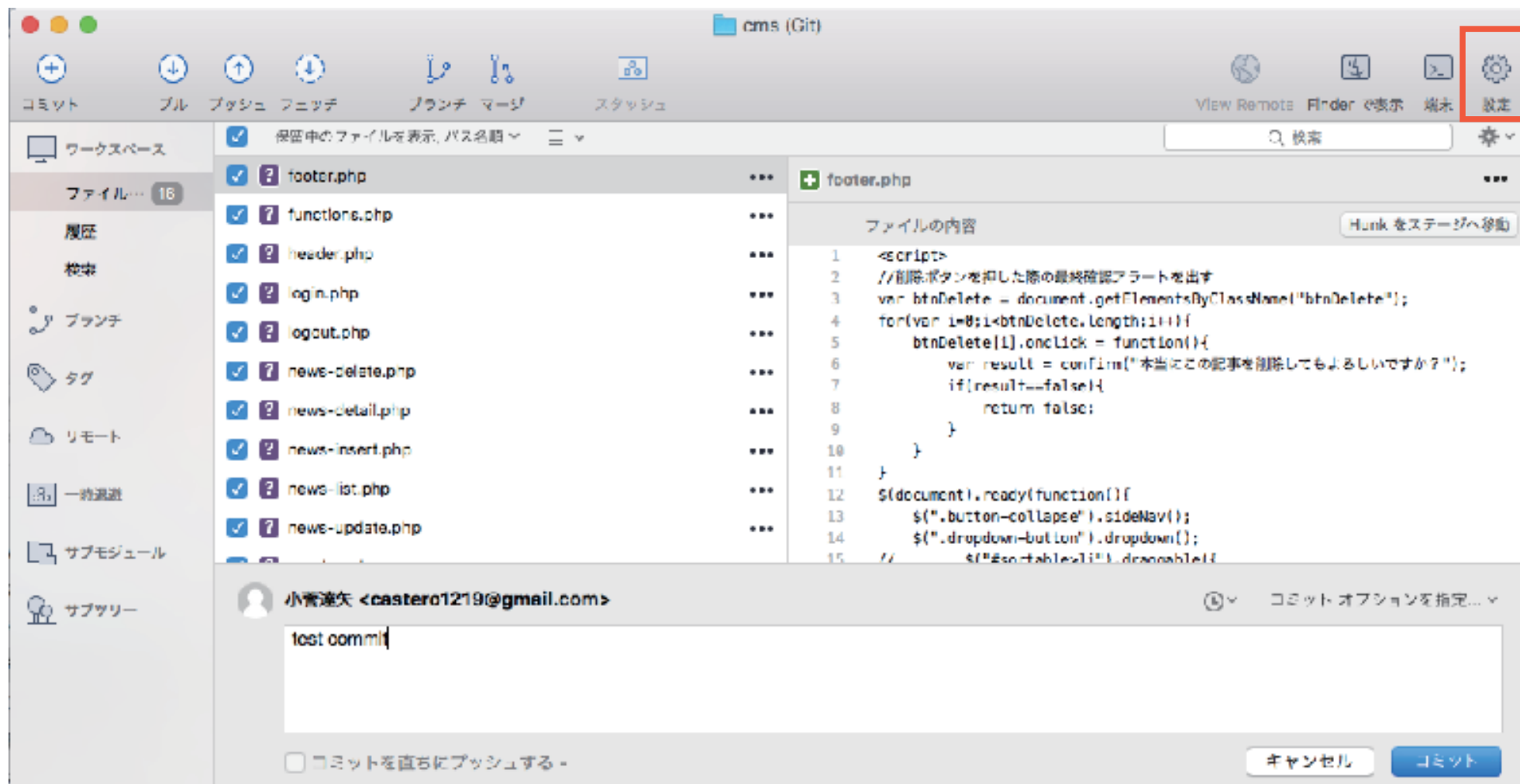
ローカルリポジトリの内容を、先ほど作成した **githubtest** というリモートリポジトリにプッシュしたいと思います。

※「プッシュ」

「リモートリポジトリにソースコードの変更記録を反映させること」になります。

複数人で作業する時に、リモートリポジトリにソースコード(コミットした内容)をプッシュしていくことによって、自分以外の他の作業者にもソースコード及びソースコードの差分・ソースコードの変更履歴を共有出来るようになります。

リモートリポジトリの設定



リモートリポジトリの設定

The screenshot shows a dialog box titled "リモートリポジトリの設定" (Remote Repository Settings). The top bar contains four tabs: "コミットテンプレート" (Commit Template), "リモート" (Remote), "セキュリティ" (Security), and "高度な設定" (Advanced Settings). The "リモート" tab is selected and highlighted with a red box. Below the tabs, the text "リモートリポジトリのパス:" (Remote Repository Path:) is displayed. A table with two columns, "名前" (Name) and "パス" (Path), is shown. Below the table, there are three buttons: "追加" (Add), "編集" (Edit), and "削除" (Delete). The "追加" button is highlighted with a red box. At the bottom of the dialog, there are three buttons: "Config ファイルを編集..." (Edit Config File...), "キャンセル" (Cancel), and "OK".

名前	パス
----	----

追加 編集 削除

Config ファイルを編集... キャンセル OK

必須情報

リモートの名前:

URL / パス:



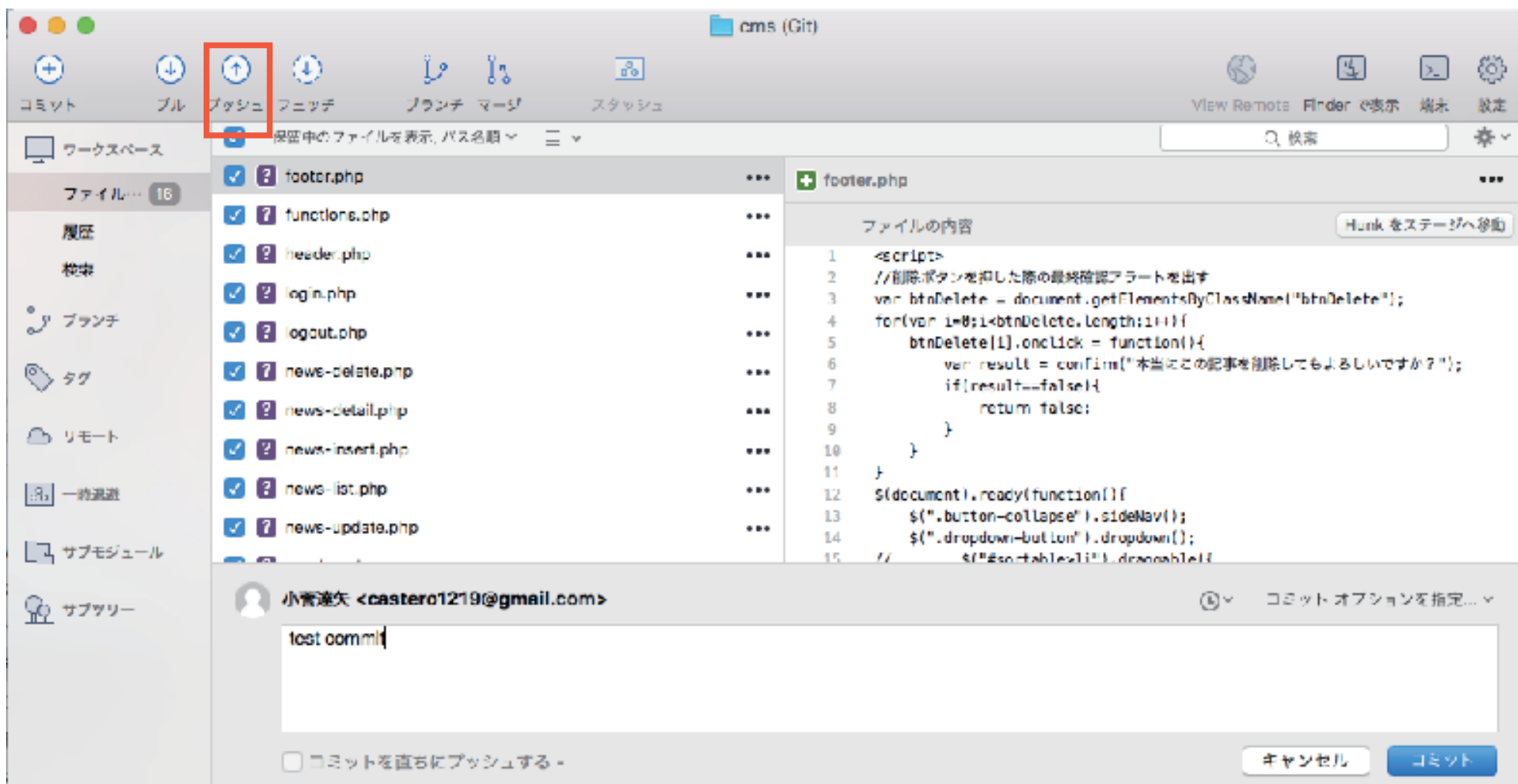
オプションの拡張統合

GitHub で先ほど作ったリポジトリとアドレスを入れます。
今回は HTTPS を選択した時のアドレスを入れておきましょう。
できたら、OK ボタンを押します。

拡張統合は、Bitbucket のようなホスティングプロバイダと、クローンを使用したりリンクをチェックアウトしたり、あるいはプルリクエストを作成したりする際に既存のクローンを指定するといったような、より密接な統合を可能にします。

キャンセル

OK



リモートリポジトリに記録を追加しよう！

プッシュ先のリポジトリ:

プッシュするブランチ

プッシュ...	ローカルブランチ	リモートブランチ	追跡...
<input checked="" type="checkbox"/>	master	master	<input checked="" type="checkbox"/>

Master にチェックし OK ボタン

☒ すべて選択

☒ すべてのタグをプッシュする

gitignore

gitignore



ここまで紹介した方法においては、git 管理下のファイルを基本的には全てリポジトリに記録する形になります、

しかしファイルの中には、**リポジトリに記録したり、リモートリポジトリに公開してはまずい性質のファイルがあったりします。**

(例：何かのアカウント & パスワード情報)

こういった際に、特定のファイルやフォルダ下のデータをリポジトリに記録しないようにするのが **gitignore** と言われる仕組みです。

gitignore における指定方法



ファイル名・・・該当ファイルを無視

(例：test.html・・・test.html を無視)

(この書き方では、同名ファイルは全て無視されます。)

フォルダ名/・・・指定フォルダ以下を無視

(例：db/・・・db フォルダ内を無視)

*.拡張子・・・指定拡張子ファイルを無視

(例：*.exe・・・.exe ファイルを無視)

他にもたくさん！！！！

gitignore における指定方法

Commit テンプレート リモート セキュリティ 高度な設定

リポジトリ限定無視リスト

/Users/TatsuyaKosuge/Desktop/gittest/.gitignore **編集**

ユーザー情報

☒ グローバルユーザー設定を使う

名前: 小菅達矢

メールアドレス: castero1219@gmail.com

コミット文字列の置換

追加 編集 削除

その他

☒ 自動更新 (無効にする場合はこのリポジトリを手動で更新しなくてはなりません)

☒ リモートステータスの更新をバックグラウンドで実行

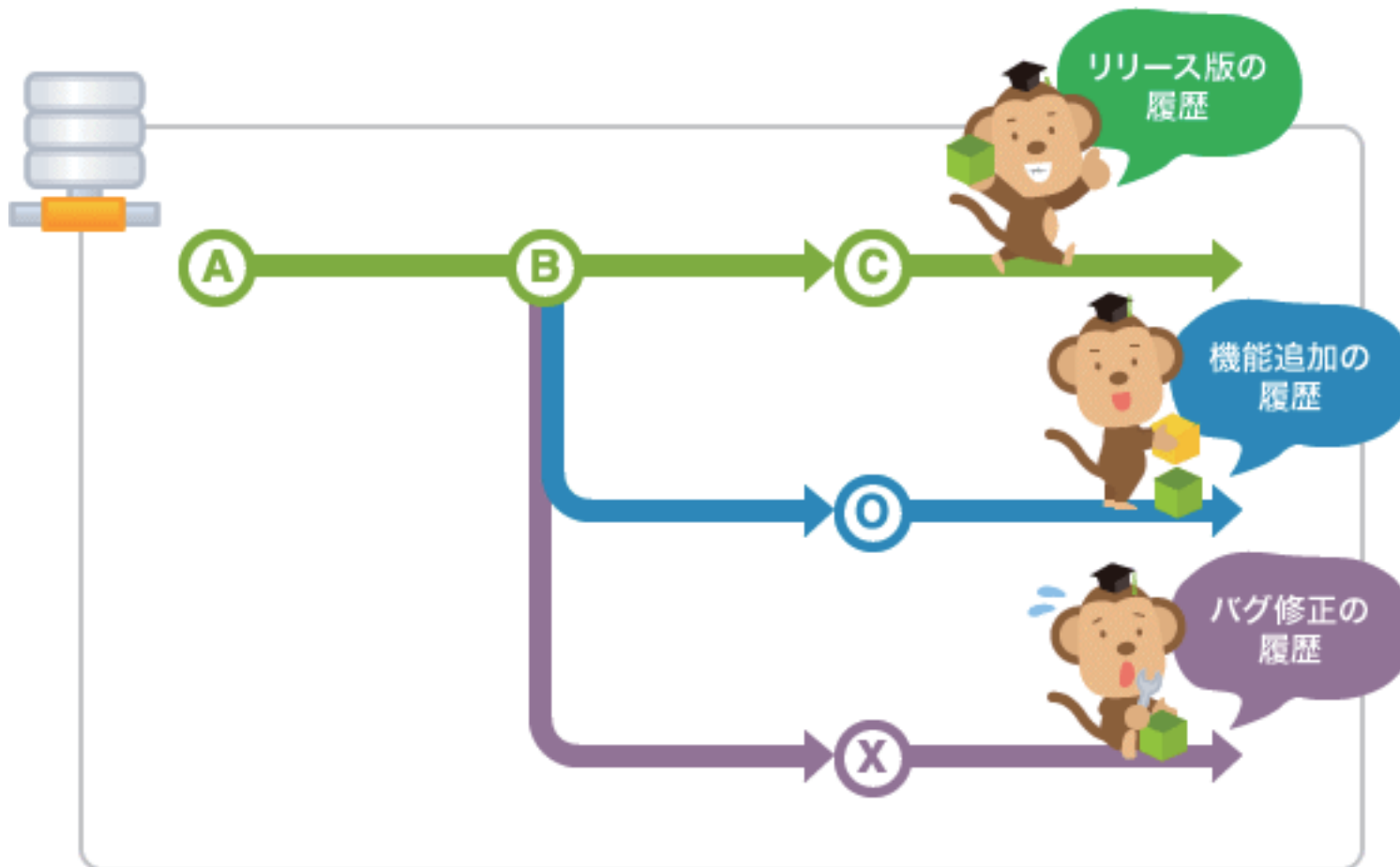
☐ サブモジュールに再帰的な操作をしない

Config ファイルを編集... キャンセル OK

Source Tree の場合は、
「設定」→高度な設定→左記部分を
編集して、実際に ignore する内容
を入力していきます。

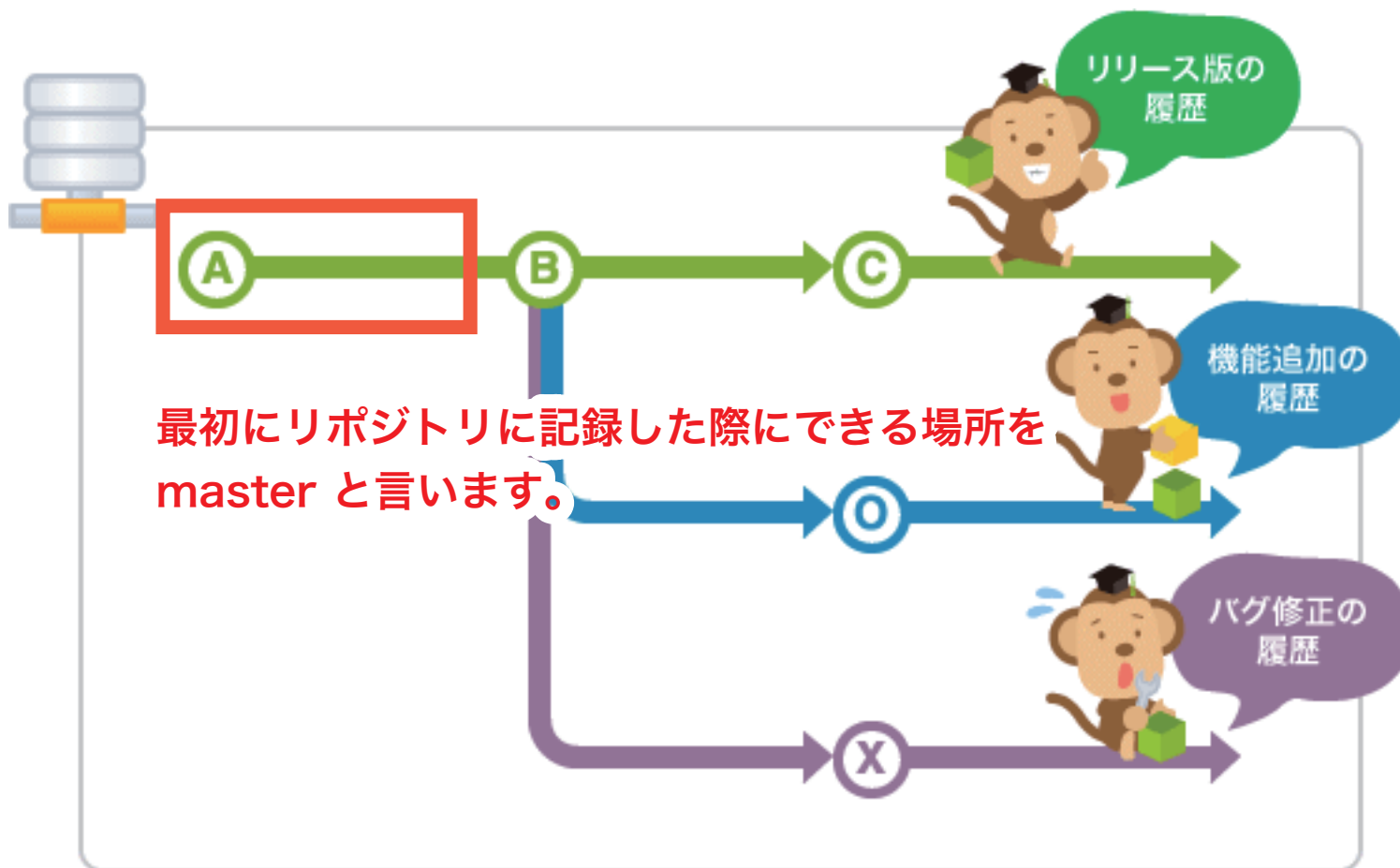
ブランチ

ブランチとは??



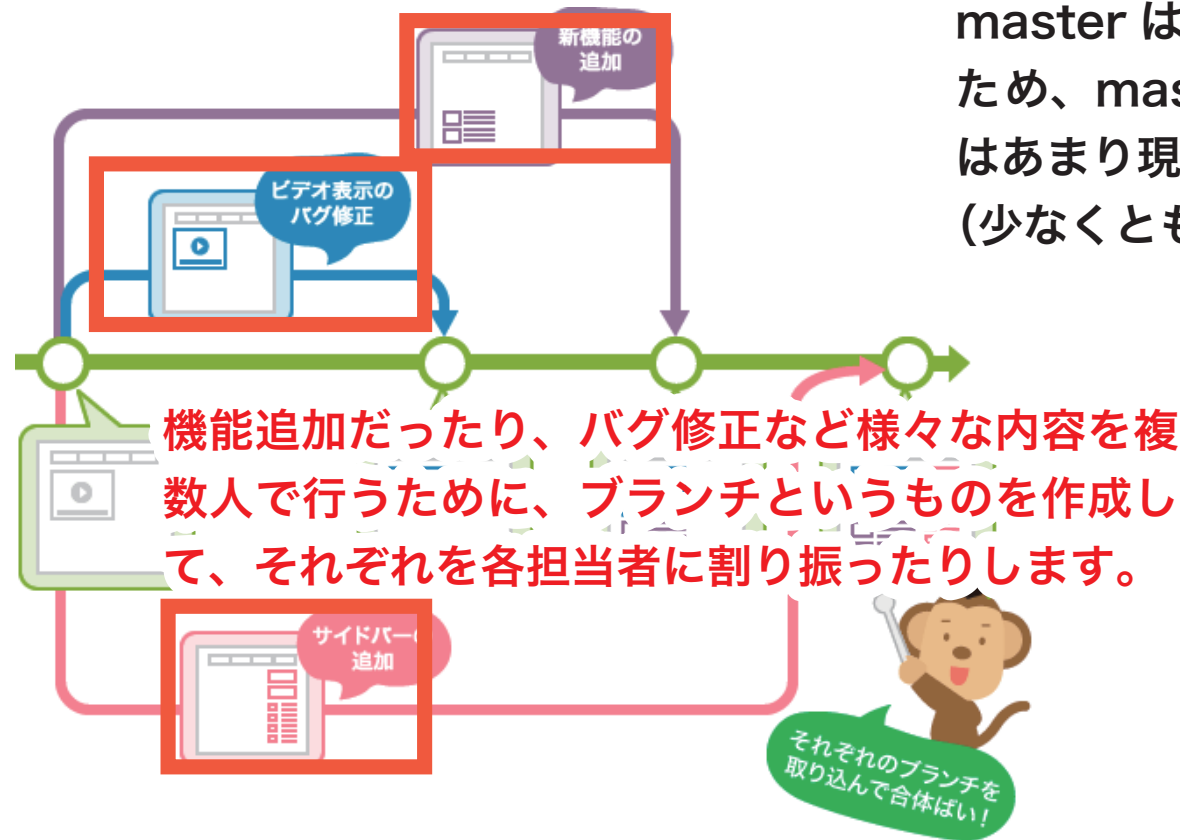
引用) <https://www.backlog.jp/git-guide/> より

ブランチとは?? -master -



引用) <https://www.backlog.jp/git-guide/> より

ブランチとは?? - ブランチとmasterの扱い -



master は、安定した状態を常に保つようにするため、master に直接 commit するようなことはあまり現場ではありません。
(少なくとも僕が関与しているところは)

引用) <https://www.backlog.jp/git-guide/> より

Source Tree でのブランチ作成方法



現在の commit からブランチを枝分かれして作成したい場合は「作業コピーの親」
特定のコミットからブランチを作りたい時は「指定のコミット」を選択した上で、該当コミットを選択。

「新規ブランチをチェックアウト」

→これから作成するブランチにコミットを積み重ねていきます、という意味。
master → ブランチへの切り替えを指します。

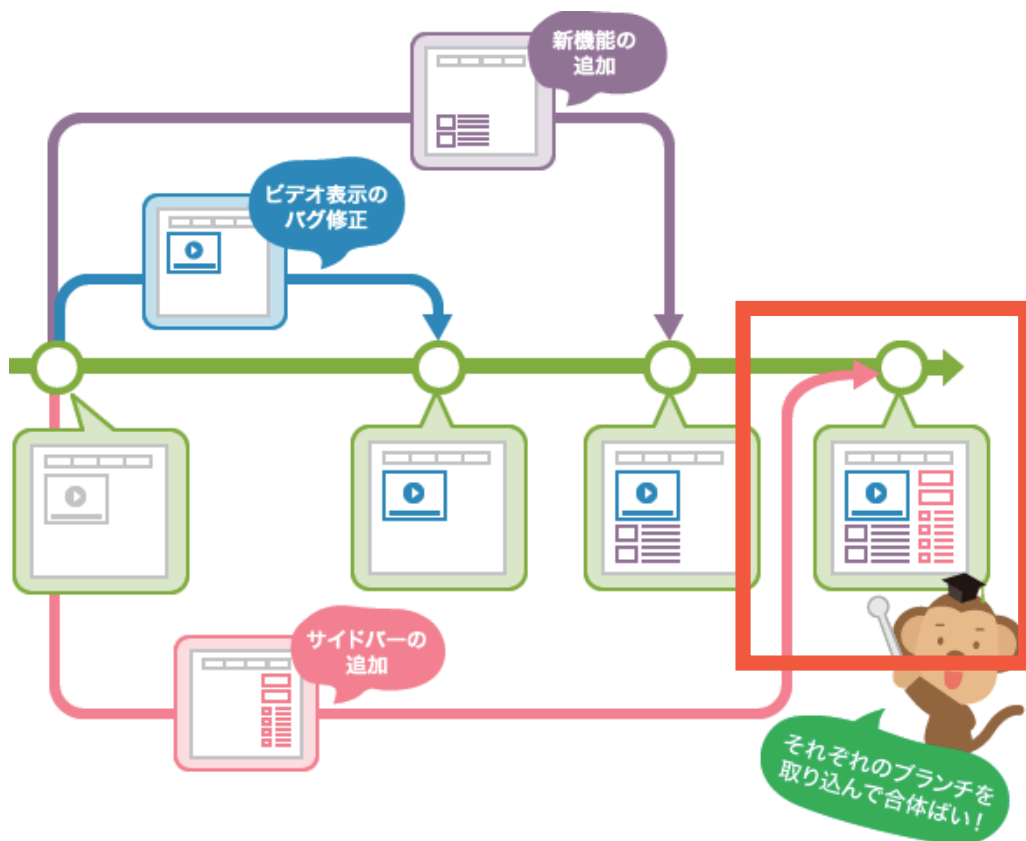
Source Tree でのブランチ作成方法



複数ブランチがある場合、今いるブランチ（あるいは master）の場所に○がつきます。ブランチ名をクリックすることで、チェックアウトつまり切り替えができます。

マージ

マージ

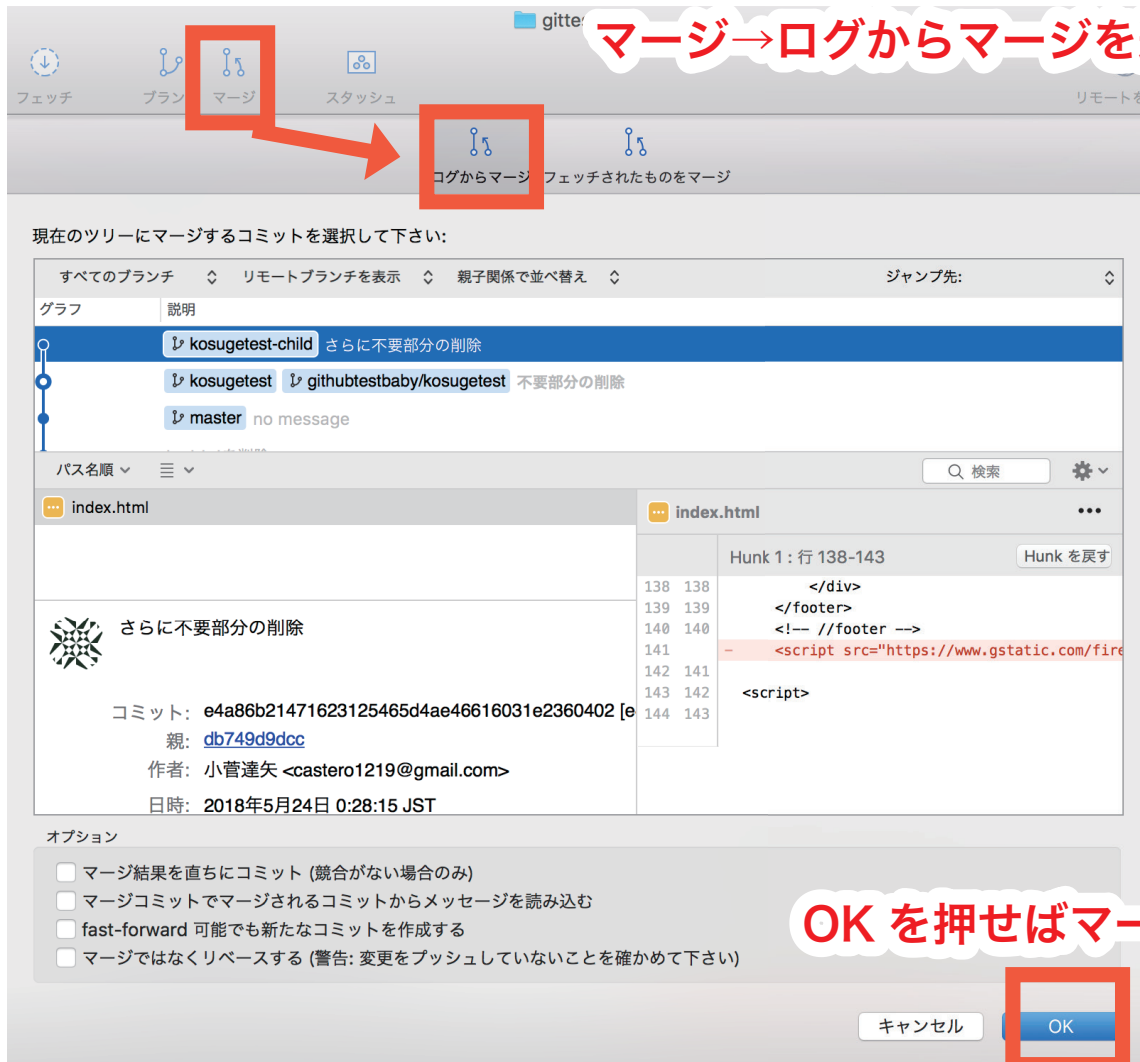


ブランチで作業完了したものは、最終的には merge (マージ) します。

マージすることで、`master` にブランチで作業した内容を反映させます。

マージの方法

マージ→ログからマージを選択します。



現在のツリーにマージするコミットを選択して下さい:

すべてのブランチ リモートブランチを表示 親子関係で並べ替え ジャンプ先:

グラフ 説明

- ↳ kosugetest-child さらに不要部分の削除
- ↳ kosugetest ↳ githubtestbaby/kosugetest 不要部分の削除
- ↳ master no message

パス名順 検索

index.html

さらに不要部分の削除

コミット: e4a86b21471623125465d4ae46616031e2360402 [e
親: db749d9dcc
作者: 小菅達矢 <castero1219@gmail.com>
日時: 2018年5月24日 0:28:15 JST

Hunk 1: 行 138-143 Hunk を戻す

```
138 138     </div>
139 139     </footer>
140 140     <!-- //footer -->
141 141     - <script src="https://www.gstatic.com/fire
142 141     <script>
143 142
144 143
```

オプション

- ☐ マージ結果を直ちにコミット (競合がない場合のみ)
- ☐ マージコミットでマージされるコミットからメッセージを読み込む
- ☐ fast-forward 可能でも新たなコミットを作成する
- ☐ マージではなくリベースする (警告: 変更をプッシュしていないことを確かめて下さい)

キャンセル OK

OK を押せばマージ完了です。

参考 URL

■ 猿でもわかる git 入門

<https://www.backlog.jp/git-guide/>

■ Source Tree という GUI を使った Git 入門シリーズ

<https://arrown-blog.com/git-begin1/>