

# 【Git & GitHub】基礎

## 環境編

### ■環境構築

- ▶ バージョンを確認
- ▶ ☆ Windows に Git をインストール
- ▶ ☆ mac に Git をインストール
- ▶ ※ Windows や mac に上記の方法で Git をインストールすると、新たに Git Bash、Git GUI といったアプリケーションが現れる。Git Bash はCUIでGitクライアントを立ち上げられるもので、Git GUI はGUIでGitクライアントを立ち上げられるものである。
- ▶ ※ Windowsならコマンドプロンプトでも、macならターミナルでもCUIクライアントを立ち上げられる。ただ、WindowsのコマンドプロンプトはLinuxコマンドを使えないので Git Bash を使ったほうが簡単。
- ▶ ※ GUIクライアントを立ち上げるアプリは Git GUI 以外にもあり、無料でインストールできる。
- ▶ ※ 各種IDEでGit操作のプラグインがあるので、それを使うのもアリ。

## 基礎編

### ■単語

- ▶ コミット
- ▶ ※ コミットの単位やタイミングは自由だが、意図的に行うことで履歴を追いやすくなる。
- ▶ リポジトリ
- ▶ ブランチ
- ▶ マージ
- ▶ ローカルリポジトリ
- ▶ ワークツリー
- ▶ インデックス
- ▶ HEAD
- ▶ リモートリポジトリ

# 【Git & GitHub】基礎

## 環境編

### ■環境構築

- ▶ バージョンを確認      \$ git --version
- ▶ ☆ Windows に Git をインストール
- ▶ ☆ mac に Git をインストール
- ▶ ※ Windows や mac に上記の方法で Git をインストールすると、新たに Git Bash、Git GUI といったアプリケーションが現れる。Git Bash はCUIでGitクライアントを立ち上げられるもので、Git GUI はGUIでGitクライアントを立ち上げられるものである。
- ▶ ※ Windowsならコマンドプロンプトでも、macならターミナルでもCUIクライアントを立ち上げられる。ただ、WindowsのコマンドプロンプトはLinuxコマンドを使えないので Git Bash を使ったほうが簡単。
- ▶ ※ GUIクライアントを立ち上げるアプリは Git GUI 以外にもあり、無料でインストールできる。
- ▶ ※ 各種IDEでGit操作のプラグインがあるので、それを使うのもアリ。

## 基礎編

### ■単語

- ▶ コミット                      管理対象の全ファの現状を記録する操作。またその記録
- ▶ ※ コミットの単位やタイミングは自由だが、意図的に行うことで履歴を追いやすくなる。
- ▶ リポジトリ                      コミットを貯めておく場所
- ▶ ブランチ                          コミット履歴の流れを分岐させてできるそれぞれの流れのこと
- ▶ マージ                              あるブランチでなされた変更点を別のブランチに取り込むこと
- ▶ ローカルリポジトリ              利用者の手元のコンピュータに作成したリポジトリ
- ▶ ワークツリー                      利用者が実際に作業するディのこと（=作業ディレクトリ）
- ▶ インデックス                      コミットを実行する前に更新内容を一時的に保存する場所
- ▶ HEAD                                現ブランチにおける最後のコミットを指すポインタ
- ▶ リモートリポジトリ              ネットワーク上に存在するリポジトリ

▶ フォーク	
▶ クローン	
▶ ※ クローンするとクローン元のリモートに <code>origin</code> という名前が自動的に付けられる。	
▶ プッシュ	
▶ プルリクエスト	
▶ フェッチ	
▶ プル	
▶ ※ 多くの場合、1つのプロジェクトにローカルリポジトリ、リモートリポジトリがそれぞれ1つ以上存在する。共同作業する際は、 <b>複数人がそれぞれのローカルリポジトリで作業を行い</b> 、リモートリポジトリに反映させる。そして、別の作業者がそれをまた自分のローカルリポジトリに取得して作業を続ける。この繰り返しで進めていくのが一般的な共同作業の流れである。	
▶ 上流ブランチ	

#### ■Gitの設定変更

▶ 設定値を変更	
▶ 設定値の一覧を確認	
▶ 特定の設定値を確認	
▶ ある設定項目を削除	

#### 主な設定項目

▶ ユーザ名	
▶ メールアドレス	
▶ 既定のエディタ	
▶ プロキシサーバURL	
▶ デフォルトブランチ名	

#### ■ローカルリポジトリを作成

▶ ※ Gitで管理対象となるのは1つのフォルダー（中身を含める）である。あるフォルダーをGitで管理すると決めたら、原則としてそこに含まれるものも全て管理対象となる。	
▶ ローカルリポを作成	
▶ ※ 上記コマンドを実行すればディレクトリのなかに <code>.git</code> という名前の隠しディレクトリが作られる。これは「Gitディレクトリ」とも呼ばれ、これこそがローカルリポジトリにあたる。	

▶ フォーク	別ユーザのリモートを自分用のリモートとして複製すること
▶ クローン	リモートをローカル環境に複製すること
▶ ※ クローンするとクローン元のリモートに <code>origin</code> という名前が自動的に付けられる。	
▶ プッシュ	あるブランチを（現状態で）リモートに複製すること
▶ プルリクエスト	プッシュしたブランチをリモートの別ブランチにマージしてもいいか、他の開発者に確認すること
▶ フェッチ	リモの（その時点の）全ブランチをローカルリポに取り込むこと
▶ プル	フェッチしたうえ、あるブランチをワークツリーに反映すること
▶ ※ 多くの場合、1つのプロジェクトにローカルリポジトリ、リモートリポジトリがそれぞれ1つ以上存在する。共同作業する際は、 <b>複数人がそれぞれのローカルリポジトリで作業を行い</b> 、リモートリポジトリに反映させる。そして、別の作業者がそれをまた自分のローカルリポジトリに取得して作業を続ける。この繰り返しで進めていくのが一般的な共同作業の流れである。	
▶ 上流ブランチ	ローカルブランチが更新を追うリモートブランチ（普通は同名）

#### ■Gitの設定変更

▶ 設定値を変更	\$ git config --global 設定項目 設定値
▶ 設定値の一覧を確認	\$ git config --list
▶ 特定の設定値を確認	\$ git config 設定項目
▶ ある設定項目を削除	\$ git config --global --unset 設定項目

#### 主な設定項目

▶ ユーザ名	user.name
▶ メールアドレス	user.email
▶ 既定のエディタ	core.editor ※VSCodeの場合、設定値は <code>"code --wait"</code> 。
▶ プロキシサーバURL	http.proxy
▶ デフォルトブランチ名	init.defaultBranch

#### ■ローカルリポジトリを作成

▶ ※ Gitで管理対象となるのは1つのフォルダー（中身を含める）である。あるフォルダーをGitで管理すると決めたら、原則としてそこに含まれるものも全て管理対象となる。	
▶ ローカルリポを作成	\$ cd 管理対象となるディのパス \$ git init
▶ ※ 上記コマンドを実行すればディレクトリのなかに <code>.git</code> という名前の隠しディレクトリが作られる。これは「Gitディレクトリ」とも呼ばれ、これこそがローカルリポジトリにあたる。	

▶ 管理しないファディを設定	
▶ クローン	
■コミットの作成	
▶ ※ ローカルリポジトリでコミットを行う際、「ワークツリー」「インデックス」「Git ディレクトリ」と呼ばれる3つのエリアを使う。	
▶ ※ ワークツリーでファやディを編集すると <b>modified</b> の状態になり、ファやディを新規作成して編集すると <b>untracked</b> の状態となる。こういったファやディをここでは節約のために MCUC (Modified Contents, Untracked Contents) と呼びたい。	
▶ ※ MCUCをインデックスに登録すると <b>staged</b> の状態となる。このようなファやディをここでは節約のために SC (Staged Contents) と呼びたい。	
▶ MCUCをインデに登録	
▶ 全MCUCを "	
▶ ワーツリのファ・ディ削除	
▶ 全SCをコミット	
▶ 全MCUC,SCをコミット	
■状態の復元	
▶ MCUCを直前コミの状態に戻す	
▶ SCのインデ登録を取消し	
▶ ※ コミット自体を取り消すコマンドも存在するが、様々な問題を引き起こすおそれがあるので避けるべき。それをするなら、「取り消すためのコミット」を追加するのがよい。	
■状態の確認	
▶ ローカル環境の状態を確認	
▶ ワーツリ／インデの差分を確認	
▶ インデ／ローカルリポの "	
▶ コミットの履歴を確認	
▶ 差分も含めて "	
▶ ブランチの一覧	
■ブランチに関する操作	
▶ 現ブランチにブランチを生やす	

▶ 管理しないファディを設定

そのようなファディが存在するディ（その先祖でもいい）に .gitignore というファを作り、中にそれらのパスを列挙して書く

▶ クローン

\$ cd parentDirPath

\$ git clone SHHKey※

※GitHubで取得

■コミットの作成

▶ ※ ローカルリポジトリでコミットを行う際、「ワークツリー」「インデックス」「Git ディレクトリ」と呼ばれる3つのエリアを使う。

▶ ※ ワークツリーでファやディを編集すると **modified** の状態になり、ファやディを新規作成して編集すると **untracked** の状態となる。こういったファやディをここでは節約のために MCUC (Modified Contents, Untracked Contents) と呼びたい。

▶ ※ MCUCをインデックスに登録すると **staged** の状態となる。このようなファやディをここでは節約のために SC (Staged Contents) と呼びたい。

▶ MCUCをインデに登録

\$ git add MCUCPath

▶ 全MCUCを "

\$ git add -A

▶ ワーツリのファ・ディ削除

\$ git rm filePath

・ \$ git rm -r dirPath

※ **staged** になる

▶ 全SCをコミット

・ \$ git commit -m "一行のコミットメッセージ"

・ \$ git commit

※直後に開かれるエディタでコミメを書く

▶ 全MCUC,SCをコミット

\$ git commit -a ..

※ **-m** つけるなら **-am** でOK

■状態の復元

▶ MCUCを直前コミの状態に戻す

\$ git restore MCUCPath

か \$ git checkout -- "

▶ SCのインデ登録を取消し

\$ git reset HEAD SCPath

▶ ※ コミット自体を取り消すコマンドも存在するが、様々な問題を引き起こすおそれがあるので避けるべき。それをするなら、「取り消すためのコミット」を追加するのがよい。

■状態の確認

▶ ローカル環境の状態を確認

\$ git status

▶ ワーツリ／インデの差分を確認

\$ git diff

▶ インデ／ローカルリポの "

\$ git diff --cached

▶ コミットの履歴を確認

\$ git log

▶ 差分も含めて "

\$ git log -p

▶ ブランチの一覧

\$ git branch

■ブランチに関する操作

▶ 現ブランチにブランチを生やす

\$ git branch 新規ブランチ名

▶ ブランチを切り替える	
▶ （上記2つを一気に）	
▶ ※ master ブランチから新たにブランチを作って分岐させたいなら、いったん master ブランチに切り替えなおしてから、ブランチを生やす。	
▶ ※ 新規ブランチに切り替えたら、普通に編集しコミットもする。	
▶ 他のブランチとの差分を確認	
▶ 現ブランチに別のをマージ	
▶ 現ブランチの名称変更	
▶ あるブランチの名称変更	
▶ ローカルリポ上の ブランチを削除	

#### ■リモートリポジトリとのやり取り

▶ 登録済みのリモの一覧	
▶ リモを新たに登録	
▶ ※ 最初のリモートリポジトリの名前は <code>origin</code> にすることが多い。	
▶ 現ブランチを初プッシュ	
▶ " を2回目以降のプッシュ	
▶ リモ上のブランチを削除	
▶ フェッチ	
▶ 現ブランチにプル	

▶ ブランチを切り替える	\$ git checkout <i>targetBranch</i> か \$ git switch "
▶ （上記2つを一気に）	\$ git checkout -b 新規ブランチ名
▶ ※ master ブランチから新たにブランチを作って分岐させたいなら、いったん master ブランチに切り替えなおしてから、ブランチを生やす。	
▶ ※ 新規ブランチに切り替えたら、普通に編集しコミットもする。	
▶ 他のブランチとの差分を確認	\$ git diff 比較対象のブランチ
▶ 現ブランチに別のをマージ	\$ git merge 別ブランチ
▶ 現ブランチの名称変更	\$ git branch -m 新規ブランチ名 ※ <code>-M</code> は強制的変更
▶ あるブランチの名称変更	\$ git branch -m ブランチ 新規ブランチ名 ※ "
▶ ローカルリポ上の ブランチを削除	\$ git branch -D ブランチ か \$ git branch --delete マージ済みブランチ

#### ■リモートリポジトリとのやり取り

▶ 登録済みのリモの一覧	\$ git remote ※ <code>-v</code> をつけるとURL込みで表示
▶ リモを新たに登録	\$ git remote add 好きなリモ名 リモのURL
▶ ※ 最初のリモートリポジトリの名前は <code>origin</code> にすることが多い。	
▶ 現ブランチを初プッシュ	\$ git push -u リモ ブランチ名※ <sup>1</sup> ※ <sup>1</sup> 普通は同名 ※リモに新規ブランチ作成&それを上流に設定&push
▶ " を2回目以降のプッシュ	\$ git push
▶ リモ上のブランチを削除	\$ git push --delete リモ ブランチ
▶ フェッチ	\$ git fetch リモ
▶ 現ブランチにプル	\$ git pull ※上流ブランチが設定されている前提