【Git】基礎

目次

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、またはアップデート
- ▶ ☆ mac に Git をインストール
- ▶ ☆ Linux 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。

【Git】基礎

目次

- 環境編
  - 環境構築
- 基礎編
  - 単語
  - Gitの設定変更
  - ローカルリポジトリを作成
  - コミットの作成
  - 状態の復元
  - 状態の確認
  - ブランチに関する操作
  - リモートリポジトリとのやり取り
- 応用編
  - GitHub CLI
  - git-ftp
  - act

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、またはアップデート
- ▶ ☆ mac に Git をインストール
- ▶ ☆ Linux 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操作用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
- ステージングエリア
- HEAD
- リモートリポジトリ
- 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 `origin` 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、**複数人がそれぞれのローカルリポジトリで作業を行い**、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

- ※ 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操作用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　コミットを貯めてゆく場所
- 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**）
- ステージングエリア　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　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 `origin` 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、**複数人がそれぞれのローカルリポジトリで作業を行い**、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

- ▶ ☆ 設定の影響範囲（レベル、スコープなどと呼ばれる）について
- ▶ 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

- ▶ ※ 設定項目名には大文字小文字の区別なし。ゆえに `git config --list` の結果ではすべて小文字で表示される。
- ▶ 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
- ▶ 自動で改行文字を統一

■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 `.git` 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」「ステージングエリア」「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イルを編集すると **modified** の状態になり、ファイルを新規作成すると **untracked** の状態となる。（以降、それぞれの状態のファイルをM、Uと表記）

■Gitの設定変更

- ▶ ☆ 設定の影響範囲（レベル、スコープなどと呼ばれる）について
- ▶ 設定値を変更　$ git config ※[1] 設定項目 設定値
  ※[1]ここでレベルを指定 (`--system` か `--global` か `--local`)
- ▶ 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 ※[1] --unset 設定項目　※[1]ここでレベルを指定

主な設定項目

- ▶ ※ 設定項目名には大文字小文字の区別なし。ゆえに `git config --list` の結果ではすべて小文字で表示される。
- ▶ 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 --wait"` 。
- ▶ プロキシサーバURL　http.proxy
- ▶ デフォルトブランチ名　init.defaultBranch
- ▶ 自動で改行文字を統一　core.autoCRLF　※デフォは `true` なので困るなら `false` に

■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 `.git` 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　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 *repoURL*※　※HTTPSと限らない

■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」「ステージングエリア」「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イルを編集すると **modified** の状態になり、ファイルを新規作成すると **untracked** の状態となる。（以降、それぞれの状態のファイルをM、Uと表記）

- ※ ＭＵをステージングエリアに登録すると **staged** の状態となる。（以降、その状態のファイルをＡと表記）
- ＭＵをステエリに登録
- 全ＭＵを〃
- ワーツリのファ・ディ削除
- 全Ａをコミット
- 全ＭＵＡをコミット

■状態の復元

- ＭＵを直前コミの状態に戻す
- Ａのステエリ登録を取消し
- 全Ａの〃
- ※ 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。

- コミットせず変更中作業を退避
- Ａを除いて〃
- gitignore対象のファ込みで〃
- メッセージをつけて〃
- 退避した作業の一覧
- 退避した作業を削除
- すべての〃
- 退避した作業の対象ファ一覧
- 退避した作業の内容を確認
- 退避した作業をワーツリに復元
- 〃をワーツリとステエリに復元
- 〃を復元と同時に削除する

■状態の確認

- ローカル環境の状態を確認
- ワーツリ／ステエリの差分を確認
- ステエリ／ローカルリポの〃
- コミットの履歴を確認

- ※ ＭＵをステージングエリアに登録すると **staged** の状態となる。（以降、その状態のファイルをＡと表記）

- ＭＵをステエリに登録　$ git add *MUPath*
- 全ＭＵ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Ａをコミット　$ git commit -m "*commitMessage*"
- 全ＭＵＡをコミット　$ git commit -am "*commitMessage*"

■状態の復元

- ＭＵを直前コミの状態に戻す　$ git restore *MUPath*　か　$ git checkout -- 〃
- Ａのステエリ登録を取消し　$ git reset HEAD *APath*
- 全Ａの〃　$ git reset --mixed HEAD

- ※ 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。

- コミットせず変更中作業を退避　$ git stash -u　※ＭＵＡを退避する
- Ａを除いて〃　$ git stash -uk
- gitignore対象のファ込みで〃　$ git stash -a　※ `-a` は `--all` と同じ。
- メッセージをつけて〃　$ git stash -u save "*stashMessage*"
- 退避した作業の一覧　$ git stash list
- 退避した作業を削除　$ git stash drop stash@{*n*}
- すべての〃　$ git stash clear
- 退避した作業の対象ファ一覧　$ git stash show stash@{*n*}
- 退避した作業の内容を確認　$ git stash show stash@{*n*} -p
- 退避した作業をワーツリに復元　$ git stash apply stash@{*n*}
- 〃をワーツリとステエリに復元　$ git stash apply stash@{*n*} --index
- 〃を復元と同時に削除する　$ git stash pop stash@{*n*}　※ `--index` でステエリも。

■状態の確認

- ローカル環境の状態を確認　$ git status
- ワーツリ／ステエリの差分を確認　$ git diff
- ステエリ／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〃
- ▶ 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
- ▶ ブランチを切り替える
- ▶ （上記２つを一気に）
- ▶ ※ main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main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。
- ▶ 別のブランチとの差分を確認
- ▶ 現ブランチ**に**別の**を**マージ
- ▶ 現ブランチの名称確認
- ▶ 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

- ▶ ☆ 現ブランチにmainブランチの最新状態をとりこむ

■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 `origin` にすることが多い。
- ▶ ※ `git@github.com: Permission denied (publickey). fatal: Could not read from remote repository.` と出たら　→　GitHubとSSHで接続するための設定をしよう。
- ▶ 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
- ▶ リモとローカルの比較
- ▶ リモのも含めブランチの一覧

- ▶ 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 *branchName*
- ▶ ブランチを切り替える　$ git checkout *targetBranch*　か　$ git switch 〃
- ▶ （上記２つを一気に）　$ git checkout -b *branchName*
- ▶ ※ main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main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。
- ▶ 別のブランチとの差分を確認　$ git diff *anotherBranch*
- ▶ 現ブランチ**に**別の**を**マージ　$ git merge *anotherBranch*
- ▶ 現ブランチの名称確認　$ git branch --contains
- ▶ 現ブランチの名称変更　$ git branch -m *branchName*　※ `-M` は強制的変更
- ▶ あるブランチの名称変更　$ git branch -m *branch branchName*　※ 〃
- ▶ 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

- ▶ ☆ 現ブランチにmainブランチの最新状態をとりこむ

■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　※ `-v` をつけるとURL込みで表示
- ▶ リモを新たに登録　$ git remote add *rRipoName*※ *rRipoURL*　※好きな名
- ▶ ※ 最初のリモートリポジトリの名前は `origin` にすることが多い。
- ▶ ※ `git@github.com: Permission denied (publickey). fatal: Could not read from remote repository.` と出たら　→　GitHubとSSHで接続するための設定をしよう。
- ▶ 現ブランチを初プッシュ　$ git push -u *rRipo rBranchName*※[1]　※[1] 普通は同名
- ▶ 〃を２回目以降のプッシュ　$ git push *rRipo rBranch*
- ▶ リモ上のブランチを削除　$ git push --delete *rRipo rBranch*
- ▶ フェッチ　$ git fetch *rRipo*
- ▶ 現ブランチにプル　$ git pull *rRipo rBranch*
- ▶ リモとローカルの比較　$ git fetch　$ git diff *rRipo/rBranch lBranch*
- ▶ リモのも含めブランチの一覧　$ git branch -a

応用編

■GitHub CLI

- ※ GitHub CLI は、GitHub（＝あるGitHubアカウントが有するGitHubリソース）を操作できるツール。（GitHubリソースのみならずローカルも操作できるコマンドもある）
- ☆ インストール
- ※ GitHub CLI のコマンドは長いのでコマンド補完の設定を行っておくとよい。
- リポを作成
- シークレットを登録
- 現ブランチからプルリク作成

■git-ftp

- ☆ インストール
- ※ (Windowsにおいて) `$ curl --version` の結果示される使用可能なプロトコル一覧のなかに `sftp` があるのに使えない場合　→　別途、sftp対応のcurlコマンドをインストール
- ※ 公式ドキュメントはコチラ。
- ☆ 初めてのアップロード（別プログラム等によってアップロード済みであっても必要）
- (2回目以降の) 差分のアップロード
- *action*の様子を最大限に冗長に出力したい
- ☆ テスト環境、開発環境で分ける

■act

- ☆ インストール
- ※ act はコミットやプルなどをせずに、GitHub Actions で実行されるようすをローカルで テストするためのソフトウェアである。

応用編

■GitHub CLI

- ※ GitHub CLI は、GitHub（＝あるGitHubアカウントが有するGitHubリソース）を操作できるツール。（GitHubリソースのみならずローカルも操作できるコマンドもある）
- ☆ インストール
- ※ GitHub CLI のコマンドは長いのでコマンド補完の設定を行っておくとよい。
- リポを作成　$ gh repo create *ripoName* ※[1]　※[1]ここにオプ
- シークレットを登録　$ gh secret set *secretName* ※[1]　※[1]ここにオプ
- 現ブランチからプルリク作成　$ gh pr create ※[1]　※[1]ここにオプ

■git-ftp

- ☆ インストール
- ※ (Windowsにおいて) `$ curl --version` の結果示される使用可能なプロトコル一覧のなかに `sftp` があるのに使えない場合　→　別途、sftp対応のcurlコマンドをインストール
- ※ 公式ドキュメントはコチラ。
- ☆ 初めてのアップロード（別プログラム等によってアップロード済みであっても必要）
- (2回目以降の) 差分のアップロード　$ git ftp push
- *action*の様子を最大限に冗長に出力したい　$ git ftp *action* -vv

- ☆ テスト環境、開発環境で分ける

■act

- ☆ インストール
- ※ act はコミットやプルなどをせずに、GitHub Actions で実行されるようすをローカルで テストするためのソフトウェアである。