Git / GitHub

Yanny's Computer 山﨑祐太

目的

本チュートリアルでは，以下の点について学んでいく

- Git / GitHubとは何か
- 基本的な使い方
- チーム開発におけるGitのTips
- Appendix

Reference

- Gitのコマンドと解説一覧(https://git-scm.com/docs)
- Gitの全容を知る(https://git-scm.com/book/en/v2)

Gitとは

- **分散型のバージョン管理ツール**
- 他にもバージョン管理ツールはあるが，現状**Gitの一択**
- ローカルとリモート(サーバー上)の両方にリポジトリをもつ
- ローカルで作業を行い変更履歴を更新し，リモートに変更を取り込む

GitHub(GitLab)とは

- GitHubは**Gitをより便利に利用できるwebサービス**の名前
- チーム開発やコードの差分チェックなどが便利に！
- 同様のwebサービスにGitLabなどもある

なぜGitやGitHubを使うか

- **いつ誰が何を変更したか**がすぐに確認できる
- **過去のある時点にすぐ戻ることができる**
- 分散型バージョン管理では，**複数人での開発がより便利に**なる

基本的な使い方

1. リモートリポジトリをローカルにclone
2. 作業用のbranchを切る
3. ローカルで作業を行い，変更をcommit
4. ある程度commitが溜まったら(別にcommitは1つでも良い)push
5. pushされたbranchを元のbranchにmerge

1.リモートリポジトリをローカルにclone

```
git clone -b develop https://github.com/yutayamazaki/Tutorials.git
```

- GitHubなどからリポジトリをローカルに取ってくる
- `-b develop` でdevelopブランチを取ってくるという意味(ブランチは後で解説)
- リモートから取ってきて，ローカルで作業，リモートに変更を取り込むと言う流れになる

2. 作業用ブランチを切る

```
git checkout -b feature/fix_yamazaki
```

- `git checkout -b ブランチ名` でブランチを作成する
- とりあえず `feature/変更や機能の名前` というブランチを作成すればok
- 作業をcommitした後にこのブランチを元のブランチにマージする

branch

- Gitで管理している履歴を枝分かれさせたもの
- 複数のブランチを作成してそれを本流に結合するという流れで開発する
- これにより複数人や複数チームが並行して別機能の開発を行える

3. ローカルで作業を行い，変更をcommit

```
git add yamazaki.md
git commit -m "add yamazaki.md"
```

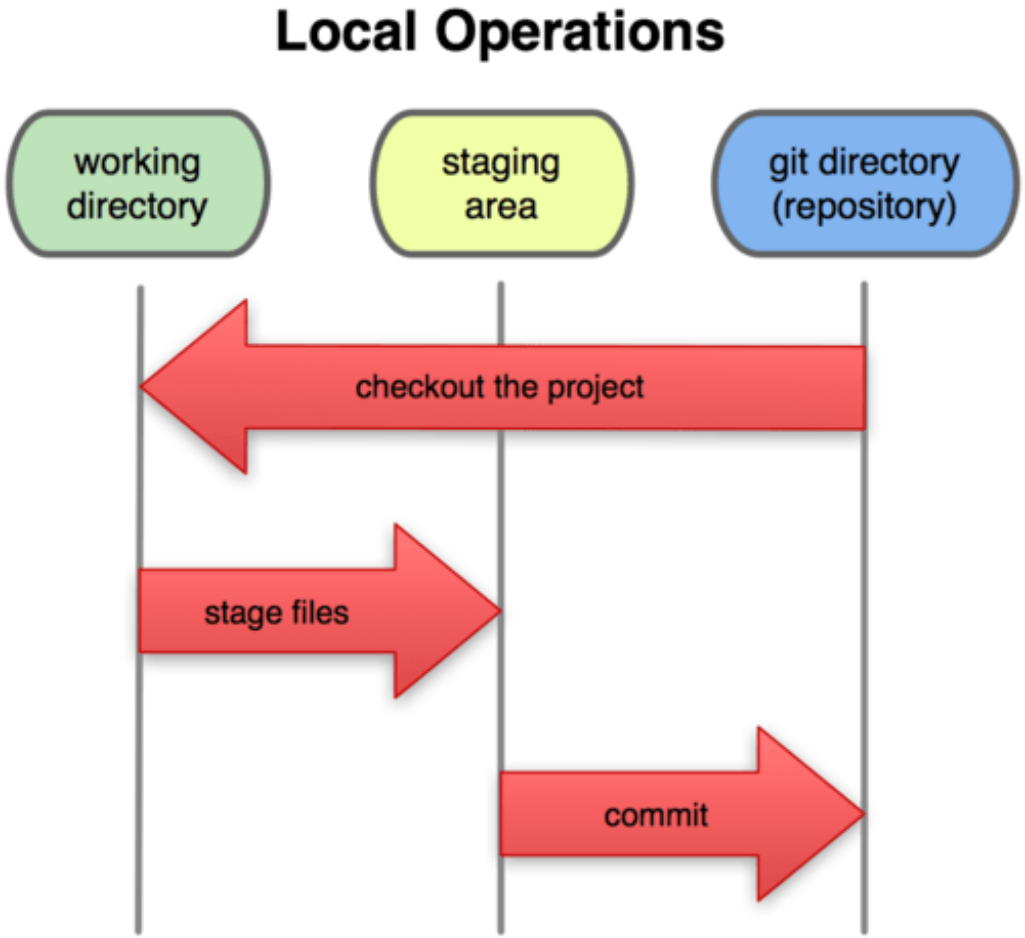
- `git add file名` でステージングエリアに上げる
- `git commit -m "コミットメッセージ"` で変更を記録する

commit

- Gitにおけるバージョン管理の単位
- ひとまとまりの作業を行うたびにcommitを行い，適宜変更を記録していく
- `git commit -m "コミットメッセージ"` でcommitする
- メッセージは `fix function` など「動詞+目的語」で何をどうしたかを書く
  - 参考記事：GitHubで使われている実用英語コメント集
- commitの単位はひとまずは大きすぎなければok

add

- Gitでcommit出来るのはステージングエリアにあるファイルだけ
- ステージングエリアがあることで，commitの単位を調整できる
- 以下の流れでcommitを行う
    i. ファイルに変更を加える
    ii. ステージングエリアに追加
    iii. commit



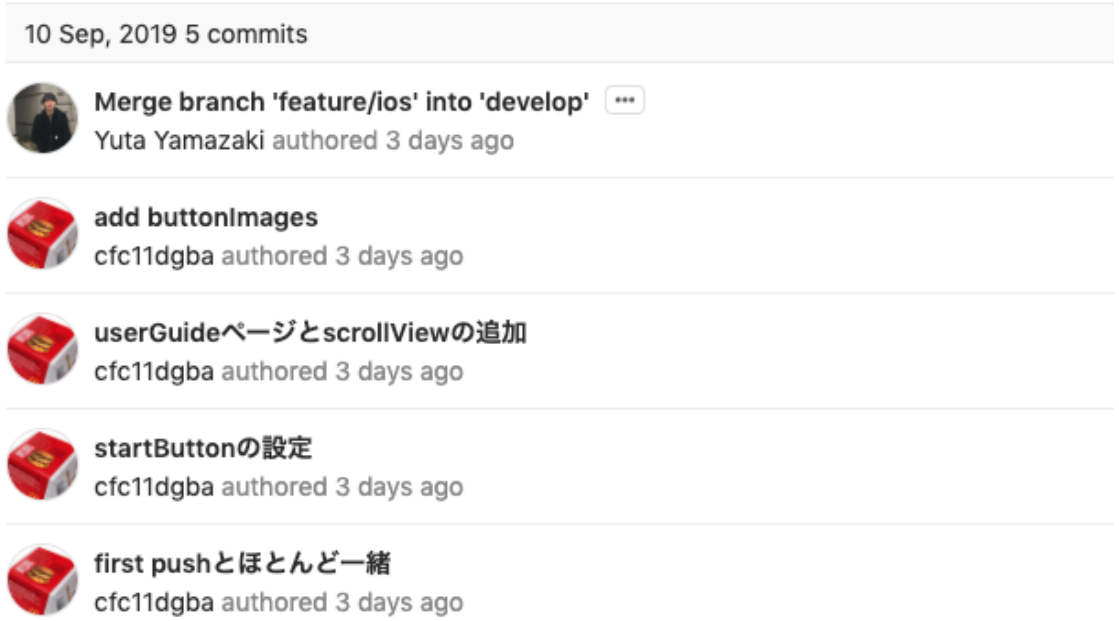
4. ある程度commitが溜まったらpush

```
git push origin feature/fix_yamazaki
```

- 自分の作業ブランチをリモートに送る
- その後GitHubやGitLabでプルリクエストやマージリクエストを作成
- コードレビューや修正を経て元のブランチにマージされる

GitのTips

commitの Tips



commitのTips

- commitはレビューの際に確認していく単位でもある
  - 大きすぎても小さすぎても面倒
  - **困ったときはより小さくまとめる(レビューしやすい)**
- **開発時にどんな変更が加えられたのかをコミットメッセージで確認**していく
  - メッセージは分かりやすく
  - commitが**意味のあるまとまり**だと理解しやすくなる！

Pull Request(Merge Request)のTips

- GitHubではPull Request，GitLabではMerge Requestと呼ばれる
- 作業ブランチを元のブランチに統合する処理のこと
- Descriptionに何の変更を施したかを書く
    - 変更された機能だけでなく，その開発の背景や目的などがあるとなお良い

頻出Gitコマンド集

確認系のコマンド

- **現状確認**

今いるブランチや変更されたファイルがステージングされているかなどを確認.

```
git status
```

- **直近数個のcommitのログを確認**

```
git log
```

- **ステージング前のファイルの変更の差分を確認**

追加した部分が緑，削除した分が赤で表示される.

```
git diff filename
```

作業取り消し系のコマンド

- addの取り消し

```
git reset filename
```

- commitの取り消し

```
git reset HEAD^
```

Gitの流れを体験する

やること

- https://github.com/yutayamazaki/Tutorials.git をforkしてローカルにcloneする
- 適当なファイルを作ってその変更をcommitする
- 自分のリモートリポジトリにpush
- 元のリモートリポジトリにPull Requestを送る

Appendix

Gitのアカウント設定

commitはアカウントに紐づけられているので，ユーザーの設定をする必要がある

- ユーザー名
- メールアドレス

の2つの情報が必要でこれが設定されていないとcommitが出来ない

設定の確認

Gitで管理されているディレクトリで以下のコマンドを打つと設定を確認できる.

```
git config -l
```

出力結果の例

```
user.email=tppymd@gmail.com
user.name=yutayamazaki
```

globalとlocalの設定

`global`と`local`の2通りの設定がある

- `global`：そのPC全体におけるデフォルトのユーザー設定(`~/.giconfig`)
- `local`：各ローカルリポジトリにおけるユーザー設定(`.git/config`)

それぞれのファイルの中身は以下のようになっており，これを書き換えることでユーザー設定を変更できる.

```
[user]
    email = tppymd@gmail.com
    name = yutayamazaki
```

configの書き換え

普通にvimなどのエディタで書き換えてもいいが，一応それ専用のコマンドもある

```
$ git config --global user.name "yutayamazaki"
$ git config --global user.email tppymd@gmail.com
```

`--global`を`--local`に変えることで，localの設定を変更できる

GitHubとGitLabの使い分け

- GitHubやGitLabに登録する際にユーザー名とメールアドレスを登録する
- このユーザー名とメールアドレスを用いて，GitHubとGitLabを同じPCで使い分けることが可能

GitHubやGitLabとの通信

- リモートリポジトリとローカルリポジトリの通信は`HTTPS`か`SSH`で行う
- GitHubの推奨通信プロトコルは'HTTPS'

HTTP と HTTPS

- `HTTP` と `HTTPS` の違いは，通信が暗号化されているか否か
- `HTTP` (Hyper Text Transfer Protocol)と `HTTPS` (Hypertext Transfer Protocol Secure)
- 通信が暗号化されていることで，第三者からは内容を理解できない

HTTPS と SSH

- `SSH` も `HTTPS` と同様にセキュアな通信を行うためのプロトコル
- `SSH` はSecure Shellの略で，リモートサーバーの操作やファイル転送などを行う

公開鍵認証

- GitHubの `SSH` のユーザー識別は公開鍵認証という暗号化の方式を用いている
    i. 「公開鍵」で暗号化し「秘密鍵」で復号する
    ii. 「公開鍵」を通信したい相手に渡して暗号化してもらう
    iii. 自分が所有する「秘密鍵」で復号する

GitHubと`SSH`で通信する

- 公開鍵と秘密鍵の生成

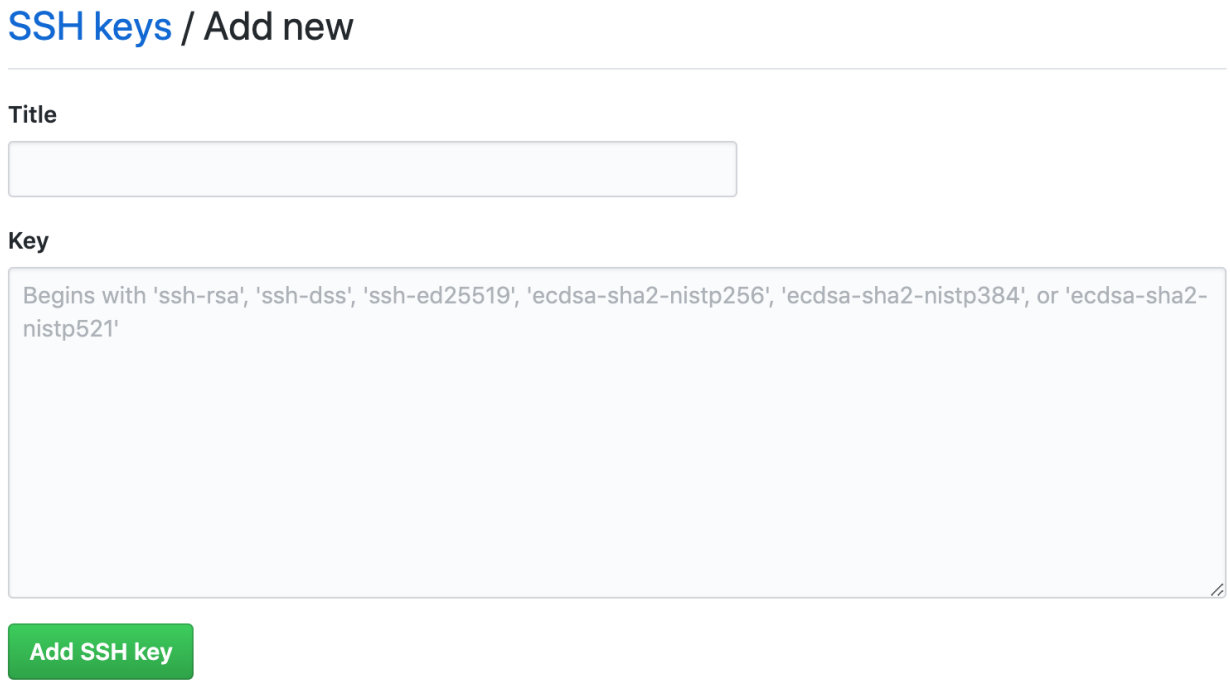
3回エンターを入力すると`id_rsa`と`id_rsa.pub`の2つのファイルが生成される

```
ssh-keygen -t rsa
```

- `id_rsa`
  - 秘密鍵で外には出さない
- `id_rsa.pub`
  - 公開鍵でこの内容をGitHubに登録する

GitHubと`SSH`で通信する

- https://github.com/settings/keys にアクセスして，`New SSH key`を押す
- Titleを適当にわかりやすい名前をつける
- Keyには`id_rsa.pub`の内容をコピペ



GitHubと `SSH` で通信する

```
ssh -T git@github.com
```

で上手く結果が表示されればok

Permittion deniedが出てるとどこかで設定がうまく言ってない