Mission2まとめ

[redacted]

Table of Contents

1. バージョン管理とは

作業ファイルやフォルダ内の変更を記録すること。

- ソフトフェアやシステムの開発の現場でエラー等が起きた際、起きる前の時点に復元できることは必須である。
- 作られるスクリプトやソースコードの作成・変更履歴を記録することで、簡単に更新内容の確認や変更前の状態に復元することができる。

2. Gitについて

2.1. Gitとは

分散型バージョン管理システムである。Gitには以下のようなメリット・特徴がある。

- 変更履歴を保存できる（いつ、誰が、どこを変更したか）
- 変更の差分をコード上に表示できる
- 管理できるファイル形式に制限はない
- ローカル上で作業するため、常にネットワークに接続されている必要がない
- チーム内で変更履歴を共有でき、同時に作業できる
- ブランチモデルでの運用が可能である
  - 機能追加・バグ修正ごとに開発者の作業を分担できる
  - レビュー箇所が明確なため作業ミスが減る
  - レビュー前や作業途中のコードが混じらないので、コードを綺麗に保つことが出来る

3. Gitの導入方法

3.1. Git for Windowsのインストール

https://git-scm.com/downloads にアクセスし、Git for Windowsをダウンロード、インストールする。

- 「Choosing the default editor…」の画面で、自分の使うeditorを選択しておくと便利

Git BashとGit GUIがインストールされる。Git BashでCUI操作、Git GUIでGUI操作が行える。

Figure 1 :git install

4. GitLabの設定

4.1. アカウント作成

https://gitlab.com/users/sign_in にアクセス、「Register」を押す。各項目入力してアカウントを作成する。

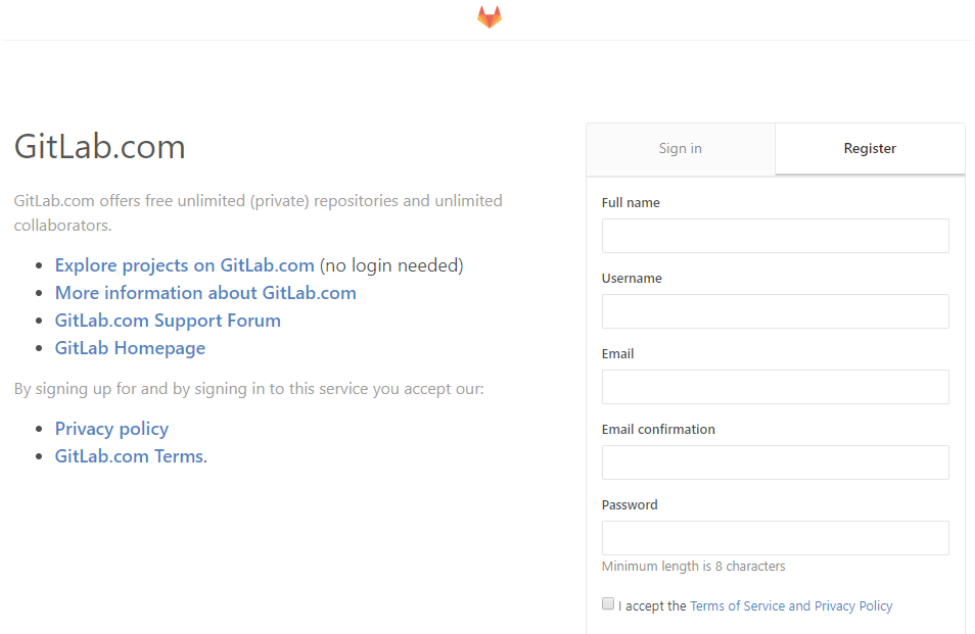

Figure 2 :gitlab

- サインインすると、メニューに「Project」「Group」などが表示される。チーム作業する際はグループを作成し、グループからプロジェクトを作成すると良い

4.2. GitLabとGit for Windowsの連携

Git Bashを起動し、初期設定を行う。

```
git config --global user.name "GitLabで登録したユーザー名"
git config --global user.email "GitLabで登録したEmailアドレス"
git config --global core.squotepath false
```

SSHの設定を行う。まず、SSHキーの出力を行う。

```
ssh-keygen -t rsa -C 'GitLabで登録したEmailアドレス'
```

生成されたSSHキーは通常以下のパスに保存されている。

```
C:\Users\username\.ssh\id_rsa.pub
```

このファイル（SSHキー）をエディタ（vscodeや秀丸など）で開き、全てコピーする。

GitLabを開き、アイコンを押下する。SettingsからSSH Keysを選択する。Keyにコピーしたsshキーをペーストし、Titleに適当な名前を付け、Add Keyを押下する。

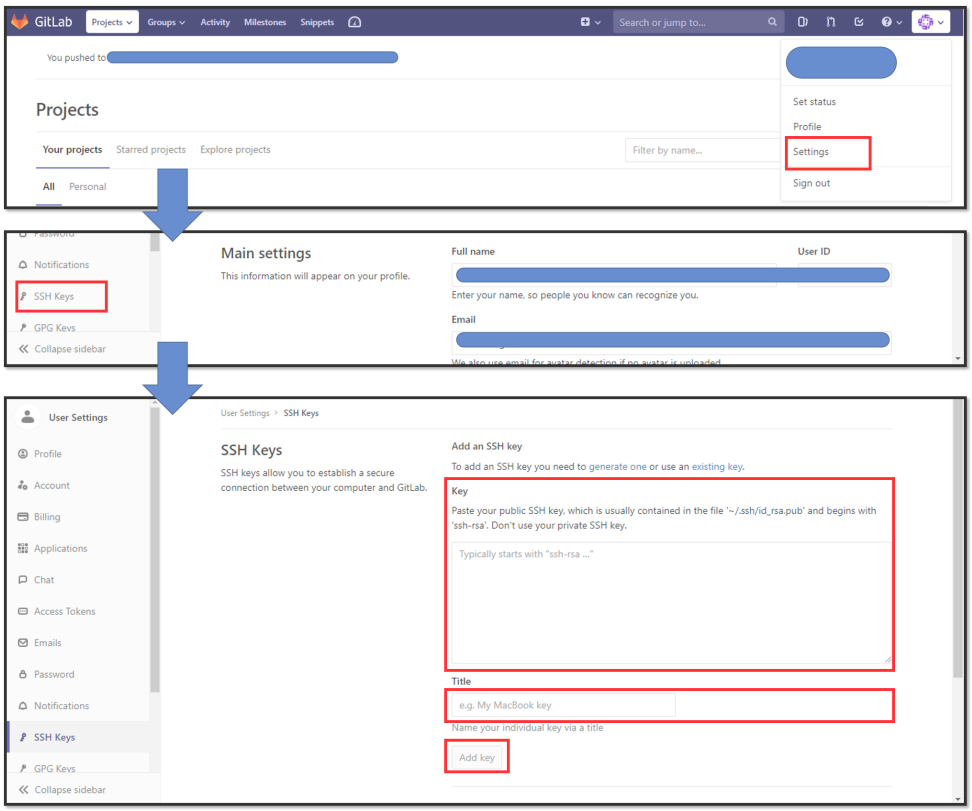

Figure 3 :ssh setting

- SSHの接続確認を行うことが出来る。

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

  接続ができている場合は以下のメッセージが表示される

  ```
  Welcome to GitLab, @ユーザー名!
  ```

  1. gitlabの紹介[amg]

- githubとの比較

5. gitのロジック

5.1. gitの要素説明・初期設定

- gitは大きく分けて五つエリアがあり、Remote Repository、Local Repository、Stage、Stash areaとWorking Directory。それらの概念に基づき、先ず説明する。

  1. Remote Repositoryはサーバー上にあり、サーバーで作業データを保存する。例としては、githubやgitlabなどのサイトはサーバーがある。
  2. Local Repositoryは自分のパソコンに.gitというフォルダの中にある。.gitフォルダはgitをinitialize時に自動的に出来る倉庫である。(後で補足説明)
  3. StageはLocal Repositoryと同じ、.gitフォルダの中にあり、自分が作業しているファイルをLocal Repositoryにアップデートする前に、必ずStageに送る必要がある。
  4. Stash areaはmergeとか衝突が発生し、解決したい場合、一時的にファイルを保存出来る場所。(4.2で補足説明)
  5. Working Directoryは自分のPCで作業しているフォルダである。

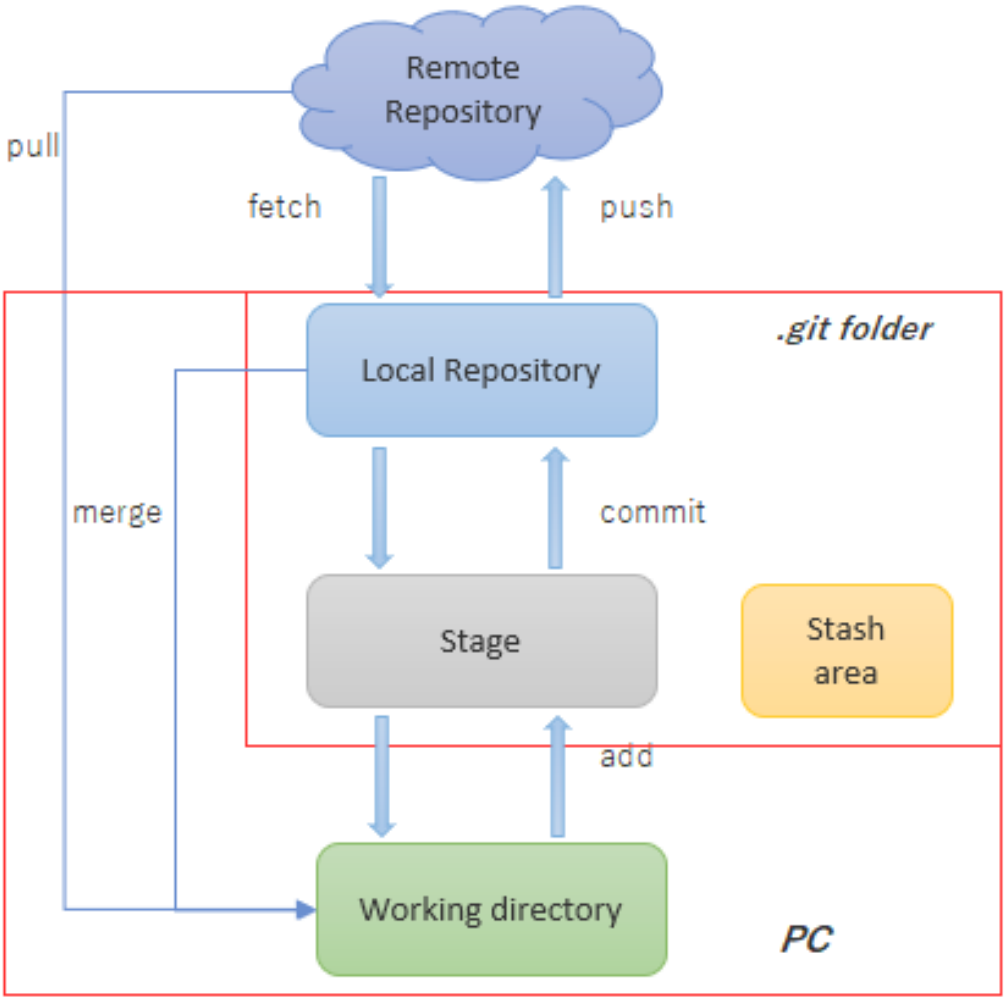

Figure 4:git workflow

- gitの使う初期設定としては、まず自分のPCあるところに作業フォルダを作って、Figure 2のような形である。

Figure 5:working folder

- そのフォルダに移動し(例"cd C:\ACN\AS-FS\bootcamp\gitとgithubFlow\repo_1st\Mission_2\world_team")、gitの作業倉庫を作り("git init")、そして、そのフォルダ中に.gitという隠されたフォルダが出て来る。このフォルダはgitの作業倉庫であり、その中にStage、Stash aeraやLocal Repositoryがある。Working Directoryはworld_teamフォルダであり、その中にファイルを修正したり、アップロードしたり、自分が仕事をしている場所と認識すればよいと思う。

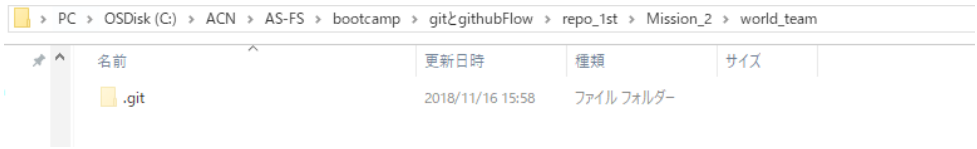

Figure 6:.git foler

- 準備出来た後に、Remoteと接続する必要があり("git remote add origin URL[自分のgitlabのURL])、もしRemoteからcloneする場合、このステップが不要となる。他人作ったプロジェットの中に.gitという倉庫も入ったので、Localでgit init必要もない。

5.2. gitのworking flow

- 要素説明と初期設定の紹介が完了となり、ここからgitのworking flowについて説明する。Figure 4はgitのlocalからremoteに作業結果をアップロードすることとremoteからlocalに更新された結果をダウンロードことの流れを描いている。

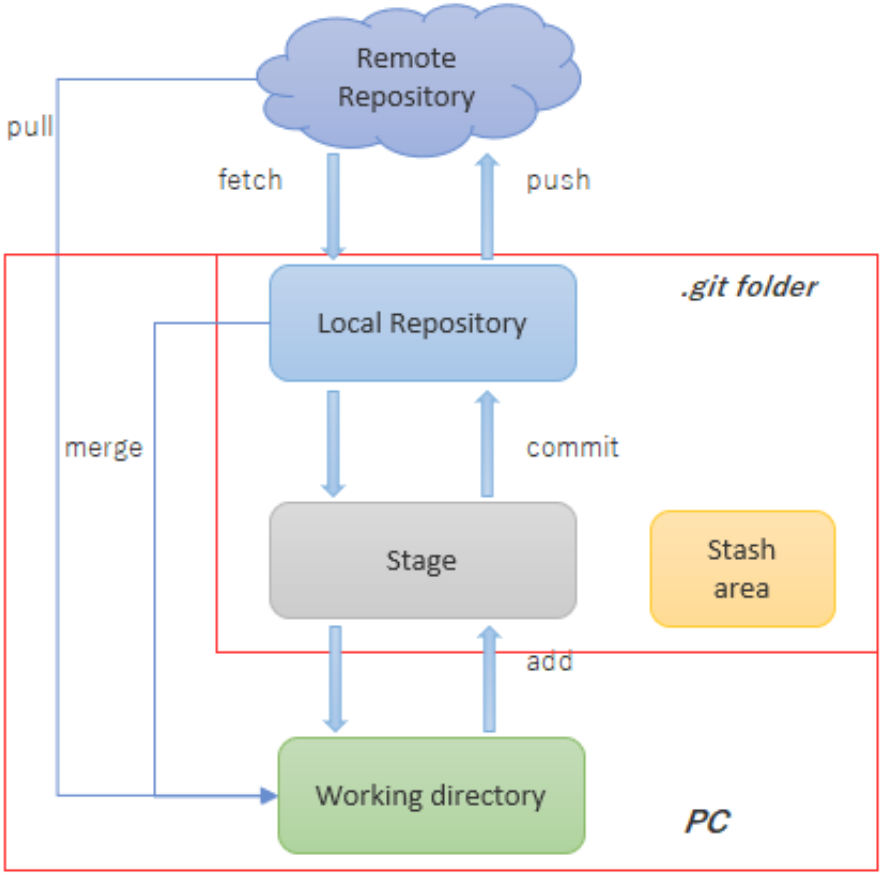

Figure 7:git workflow

- localをRemoteにアップロードする場合、まず、修正や新規のファイルをStageにaddする("*git add -A/-u")。次に、Stageのファイルを Local Repositoryにアップロードする(git commit -m "コメント")。そして、Local RepositoryからRemote Repositoryにpushする ("*git push -u origin master")。ここで、自分のWorking Directoryで作業しているファイルをRemoteに更新できた。

- localをRemoteからダウンロードする場合、二つ方法がある。一つ目は、pullである("git pull origin master")。二つ目は、fetchとmergeである("git fetch origin master","git merge")。pullはfetch+mergeと考えてよいと思う。fetchはRemote Repositoryから情報をLocal Repositoryに送り、mergeはLocal Repositoryから情報を自分のWorking Directoryに送る。pullはfetchとmergeを一緒にするコマンドとなり、使う時に気を付けなければならない。なぜなら、Local RepositoryからWorking Diretoryに情報を更新する時に、merge conflictがよく出る。(gitのconflictに補足)

5.3. gitのbranch

- gitのconflictに行く前に、branchに対して、説明する。gitの主なbranchはMasterと呼ばれ、masterから別のbranchを作る操作がcheckoutと呼ばれる。

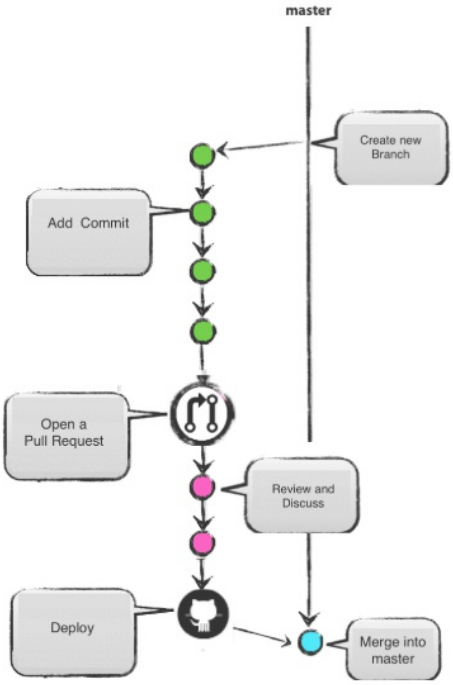

Figure 8:git branch

- gitはlocalとremoteのbranchの概念があり、それを理解した上で、これからの内容も分かりやすいと思う。LocalのbranchとRemoteのbranchは一対一に接続となる。例えば、Local MasterがRemote Masterにpushし、Local A(branch)がRemote A(branch)にpushする。Remote MasterからRemote Aをcheckoutことができ(gitlabで操作)、Local MasterからLocal Aをcheckoutこともできる("git checkout -b [branch name]")。LocalのbranchをRemoteにpushし、新しいRemote branchを作ることがRemoteでbranchを作るの一つ方法である("git push origin <local branch name >:<remote branch name>")。RemoteのA branchはRemoteのMaster branchにmergeすることがよくあると思う。LocalのA branchはLocalのMaster branchにmergeすることもできる。RemoteとLocalの操作がお互いに影響を与えないとなる。

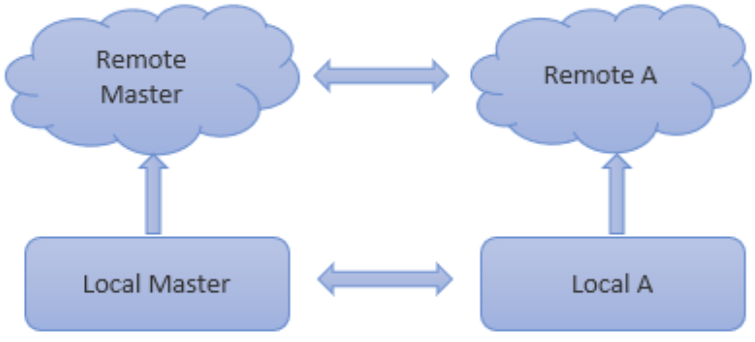

Figure 9:local branch and remote branch of git

5.4. gitのconflict

- 実際gitで仕事をする時に、先紹介したmerge conflictがよく出て来る。この理由について説明する。Figure 6のような形は自分は今topic branchで作業している。簡単にすると(localとremoteを考えないように)、topicに作業したことをmasterにmergeしたい。問題はmasterの状況も更新され、topic branchの状況は最新のmasterバージョンと一致しない。その場合は、Masterから更新する必要がある(pull)。

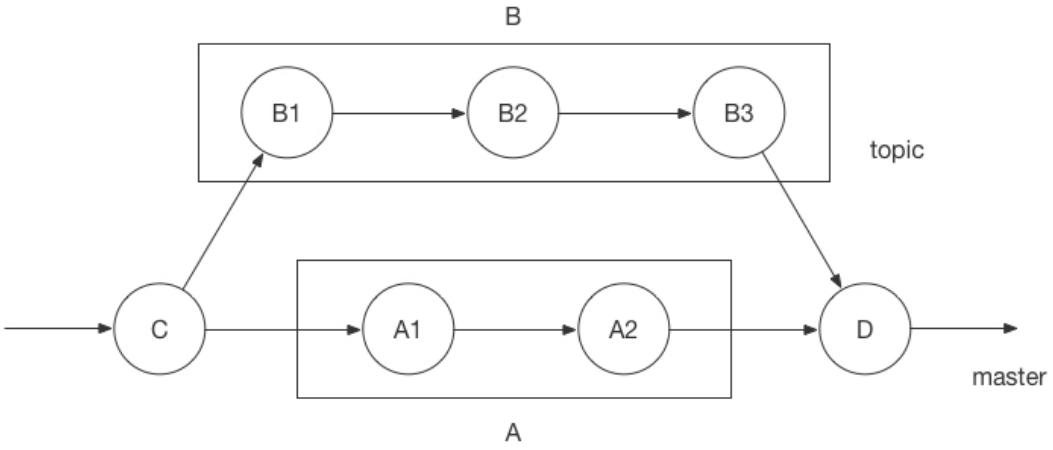

Figure 10:merge conflict

- Figure 11の通り、mergeができない。なぜなら、自分のWorking Directoryが最新のバージョンではないと、gitは衝突が自動的に処理出来ない。その場は沢山方法があると思うが、よく使うのがstash機能である。stash areaは一時的な保存場所であり、Working Diretoryのデータを一旦stash areaに送って、Remoteからpullし、そして、stashから元のことを引っ張て、比較した後、自分で直してRemoteにpushする。

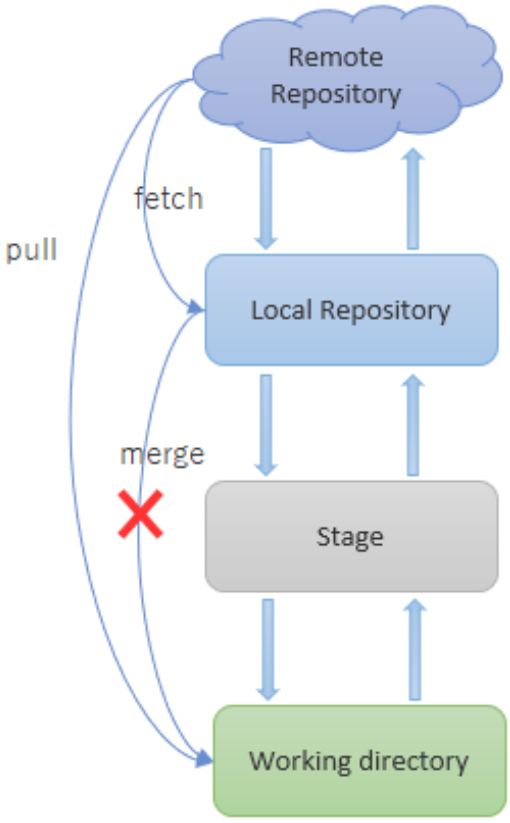

Figure 11:merge conflict

6. GitHub Flowについて

Git機能であるブランチを使って運用されるブランチモデルの一つである。Git Flowを簡略化させ、運用しやすいモデルとなっている。

6.1. ルール

1. Masterブランチは常にリリース可能な状態
2. かならずMasterブランチからTopicブランチを切る
3. 定期的にPushする
4. Masterブランチにマージする前にレビューを行う
5. コードレビューはマージ（プル）リクエストで依頼する
6. レビュー後は手を加えずマージする

6.2. 手順

1. 追加機能開発やバグ修正に対して、MasterブランチからTopicブランチを切る
2. 開発する
3. 開発が終わったら、マージリクエストを送る
4. リクエスト受信者はレビューを行い、
    - 問題があれば、修正を依頼する（2へ戻る）
    - 問題が無ければ、5へ
5. Masterブランチへマージする