

Git操作マニュアル

Introduction

概要

Gitとは、プログラムのコードの変更・差分を記録し、バージョン管理を補助するシステムのことである。これを使いやすくしたサービスとして、GitHubやGitlabなどが存在する。

なぜGitを使うのか

Gitを利用することで、主に以下のような利点を得られる。
マスタファイル、修正中ファイルの区別が明確にできる * 誰がどのような修正を行ったか追跡できる *
修正の統合・差し戻しが容易に行える

環境構築

Windowsのローカル環境において、Gitを操作する環境を構築する手順を説明する。

SourceTreeのインストール

ローカルでのGUI(Graphical User Interface) 操作を可能にするため、「SourceTree」を導入する。

1. <https://ja.atlassian.com/software/sourcetree> よりインストーラをダウンロードする
2. ダウンロードしたインストーラを実行する
 - a. .Net Frameworkが古い場合は警告が出るため、指示に従ってインストールする
 - b. SourceTreeを指示に従ってインストールする（設定の変更等は不要）
 - c. Gitが存在していないと警告が出た場合は、「Gitをダウンロードする」を選択してインストールする

sshキーの登録

リモートとの通信はhttpsとsshが利用できる。httpsを用いる場合特に設定は不要である。ここではsshで通信するための設定方法を記す。

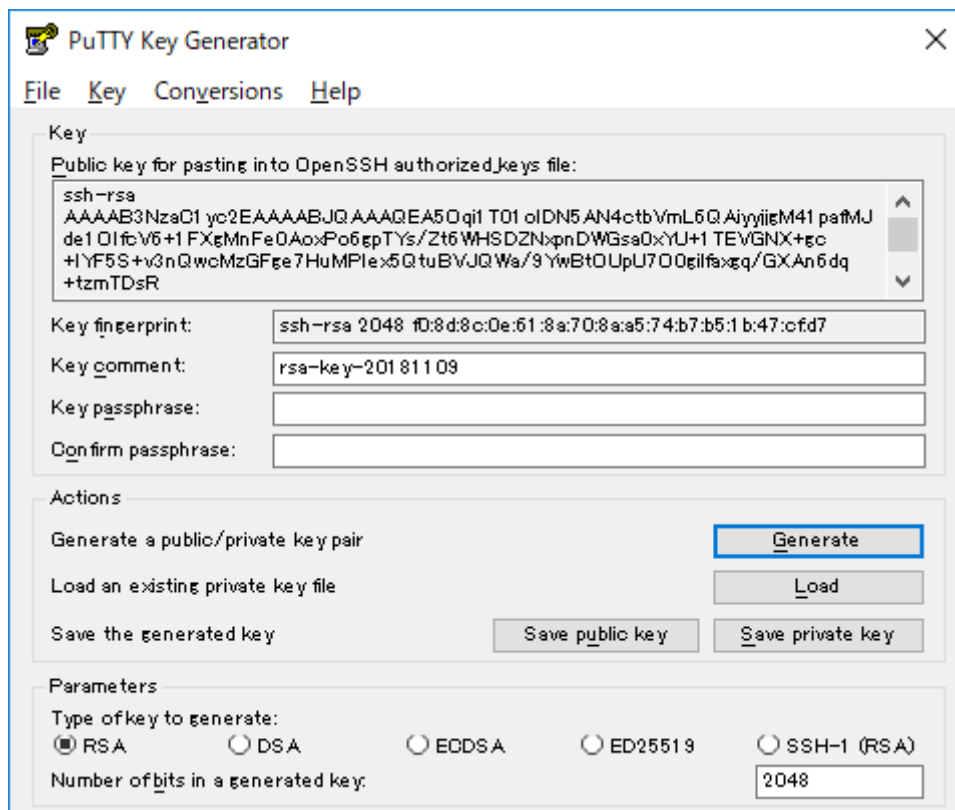
sshキーの生成

1. SourceTreeを起動する
2. 「ツール」→「SSHキーの作成／インポート」を押す
3. 「Generate」ボタンを押す
4. 画面上部の緑ゲージが右端に到達するまで、マウスをゲージ下空白部分で適当に動かす

5. 「Save public key」 ボタンを押し、任意の場所・ファイル名で保存する
6. 「Save private key」 ボタンを押し、任意の場所・ファイル名で保存する
7. 「Public key for pasting...」の内容をコピーする

sshキーのGitlabへの登録

1. Gitlabにログインする
2. 右上のアイコンを押し、「Settings」を選ぶ
3. 左側のリストから「SSH Keys」を選択する
4. 「Key」欄にコピーしておいた「Key fingerprint」の内容を貼り付け、「Add Keys」ボタンを押す



PuTTY Key Generator

File Key Conversions Help

Key

Public key for pasting into OpenSSH authorized_keys file:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAAQEA5Oqi1T01oIDN5AN4ctbVmL6QAiyyijgM41pafMJ
de1OIfcV6+1FXgMnFe0AoxPo6gpTYs/Zt6WHSdZNxpNDWGsa0xYU+1TEVGNX+gc
+IYF5S+v3nQwcmZGFge7HuMPlex5QtuBVJQWw/9YwBtOUpU7O0giffaxgg/GXAn6dq
+tzmTDsR
```

Key fingerprint: ssh-rsa 2048 f0:8d:8c:0e:61:8a:70:8a:a5:74:b7:b5:1b:47:cf:d7

Key comment: rsa-key-20181109

Key passphrase:

Confirm passphrase:

Actions

Generate a public/private key pair **Generate**

Load an existing private key file **Load**

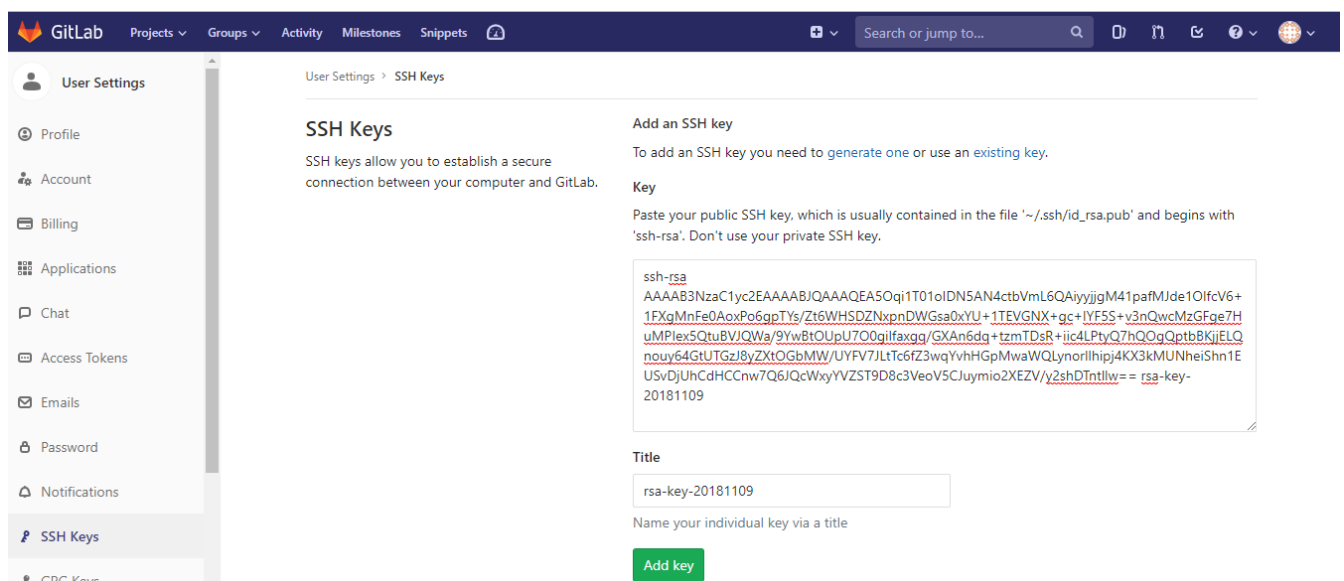
Save the generated key **Save public key** **Save private key**

Parameters

Type of key to generate:

☒ RSA ☐ DSA ☐ ECDSA ☐ ED25519 ☐ SSH-1 (RSA)

Number of bits in a generated key: 2048



GitLab Projects Groups Activity Milestones Snippets

User Settings > SSH Keys

SSH Keys

SSH keys allow you to establish a secure connection between your computer and GitLab.

Add an SSH key

To add an SSH key you need to [generate one](#) or use an [existing key](#).

Key

Paste your public SSH key, which is usually contained in the file '~/.ssh/id_rsa.pub' and begins with 'ssh-rsa'. Don't use your private SSH key.

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAAQEA5Oqi1T01oIDN5AN4ctbVmL6QAiyyijgM41pafMJde1OIfcV6+
1FXgMnFe0AoxPo6gpTYs/Zt6WHSdZNxpNDWGsa0xYU+1TEVGNX+gc+IYF5S+v3nQwcmZGFge7H
uMPlex5QtuBVJQWw/9YwBtOUpU7O0giffaxgg/GXAn6dq+tzmTDsR+iiic4LPtyQ7hQOgQptb8KjELQ
nouy64GtUTGzJ8yZxtOGbMW/UyFV7JLtTc6fZ3wqYvhHGpMwaWQlynorlhipj4KX3kMUNheiShn1E
USvDjUhcHCnw7Q6JQcWxyVZST9D8c3VeoV5CJuymio2XEZV/y2shDTntllw== rsa-key-
20181109
```

Title

rsa-key-20181109

Name your individual key via a title

Add key

sshキーのSourceTreeへの登録

1. SourceTreeを起動する
2. 「ツール」→「オプション」を押す
3. 「SSH クライアントの設定」の「SSH キー」に、保存した「Save private key」のファイルを指定する
4. 「ツール」→「SSHエージェントを起動」を押す
5. Windows右下のインジケータから、「pageant」をダブルクリックする
6. 「Add Key」で保存した「Save private key」のファイルを指定する

The screenshot shows the 'Options' dialog box in SourceTree. The 'SSH Client Settings' section is active. It includes fields for 'SSH Key' (set to a path in the user's Documents folder), 'SSH Client' (set to 'PuTTY / Plink'), and a checked option 'SourceTree 起動時に SSH エージェントを起動します'. Below this is the 'Repo Settings' section with fields for 'Project Folder', 'Language' (set to '自動'), and 'Default Text Encoding' (set to 'utf-8'). There are several checkboxes for repository behavior, such as 'Backup before destructive operations' and 'Update automatically when files change'. The 'OK' button is at the bottom right.

オプション

全般 Updates Diff Git Mercurial カスタム操作 認証 ネットワーク

☒ SourceTree に Git と Mercurial のグローバル設定ファイルの変更を許可する
☒ Use this version of SourceTree for URI association
☒ 将来的にブックマークを作成するように依頼

デフォルトユーザー情報

フルネーム:
メールアドレス:

SSH クライアントの設定

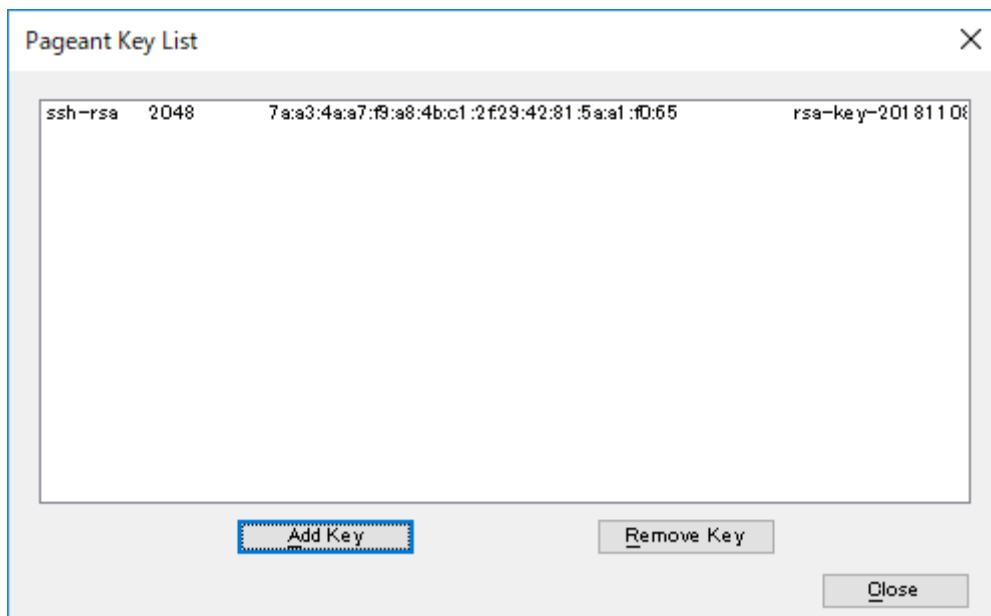
SSH キー: ...
SSH クライアント: (Git でのみ有効です。Windows 上の Mercurial では常に Plink を使います)
☒ SourceTree 起動時に SSH エージェントを起動します

Repo Settings

プロジェクトフォルダ: ...
言語: (再起動が必要) [Help translate Sourcetree!](#)
デフォルトの文字コード:

☒ 破壊的操作の前にバックアップを取る
☒ ファイルに変更が加わった時自動的に更新する
☐ アプリケーションにフォーカスがないときに更新する
☒ デフォルトのリモートの変更を調べる間隔 分
☒ 起動時にリポジトリタブを開き直す
☐ 常に全ての出力をコンソールに表示する

OK



GitLabの登録

https://gitlab.com/users/sign_in#register-pane にアクセス

画面右側のテキストボックスに必要事項を記入し、チェックボックスにチェックを入れ、[Register]を押下

登録したメールアドレスにGitLabからメールが届くので、[Confirm account]と書かれたリンクを押下

your

登録完了。次回からは https://gitlab.com/users/sign_in からログインする

Source Treeの操作方法

樹形図の見方



- ・ origin/master: リモート上にある幹となるマスタ
- ・ origin/HEAD: コミットされている地点
- ・ master: ローカル上にあるマスタ
- ・ origin[任意のブランチ名]: リモート上にあるブランチ

- ・ [任意のブランチ名]:ローカル上にあるマスタ

クローンの作成

クローンを作りたい元となるプロジェクトを指定する

■GitLabの画面操作

- 1、GitLabにアクセスし、クローンを作りたい対象のプロジェクトの画面にアクセス
- 2、[SSH]を[HTTPS]に変更し、URLをコピー

クローンを作成する

■SourceTreeの画面操作

- 1、左上のファイルタブから[新規/クローンを作成する]をクリック
- 2、上記でコピーしたURLをテキストボックス[元のパス/URL]にペースト
- 3、フォーカスアウトすると各テキストボックスに必要な情報が自動で入力されるので、[クローン]を押下

ブランチの作成

- 1、[ブランチ]をクリック
- 2、[新規ブランチ]に任意のブランチ名を入力
- 3、[新規ブランチを作成してチェックアウト]にチェックを入れる（こうすることで、新規作成したブランチが作業対象になる）
- 4、[ブランチを作成]を押下

ブランチを編集し、反映させる

- 1、画面下の[ファイルステータス]を押下
- 2、[エクスプローラで開く]を押下し、編集したいファイルを開く
- 3、編集後保存し、ファイルを閉じる
- 4、作業ツリーのファイルからコミットしたい対象のファイルを選択し、[選択をインデックスに追加]を押下
作業ツリーにあるファイルすべてをコミットする場合、[全てインデックスに追加]を押下する
- 5、画面下にあるテキストエリアに、任意のコメントを入力
ここで入力したコメントが、履歴として残る
- 6、[変更をすぐに「branch名」にプッシュする]にチェックを入れる
- 7、[コミット]を押下

ブランチをマスタにマージする

- 1、画面左の[ブランチ]タブで、[master]を選択
- 2、画面上の[マージ]を押下
- 3、マージしたい対象のブランチを選択し、[OK]を押下

Gitとは？

利用の流れ

1. サーバー上で、修正したい元のデータのコピーを作成する（branchの作成）

2. branchのデータを手元の環境（ローカル）にダウンロードする
3. ローカルのbranchに修正を加える
4. 修正したローカルbranchの内容を、リモートに反映（push）する
5. 修正が反映されたbranchを、修正元に統合したいという依頼を出す（merge request）
6. 統合依頼を受けた責任者が修正内容を確認し、問題がなければ統合を行う（merge）

用語の説明

- リポジトリ
ブランチ、差分情報を含む、ファイルデータ全体のこと。
- リモートリポジトリ
サーバー上のリポジトリのこと。マスタ管理は基本的にこちらで行う。
- ローカルリポジトリ
個人PC内に存在するリポジトリのこと。プログラムの修正はこちらで行う。
- マージリクエスト
修正した差分を、大元のデータに統合する依頼を出すこと。権限者による承認がなければ統合できないため、危険な修正が反映されることを防止できる。

コマンドの説明

以下、特に断りがない限りローカルリポジトリに対する操作である。

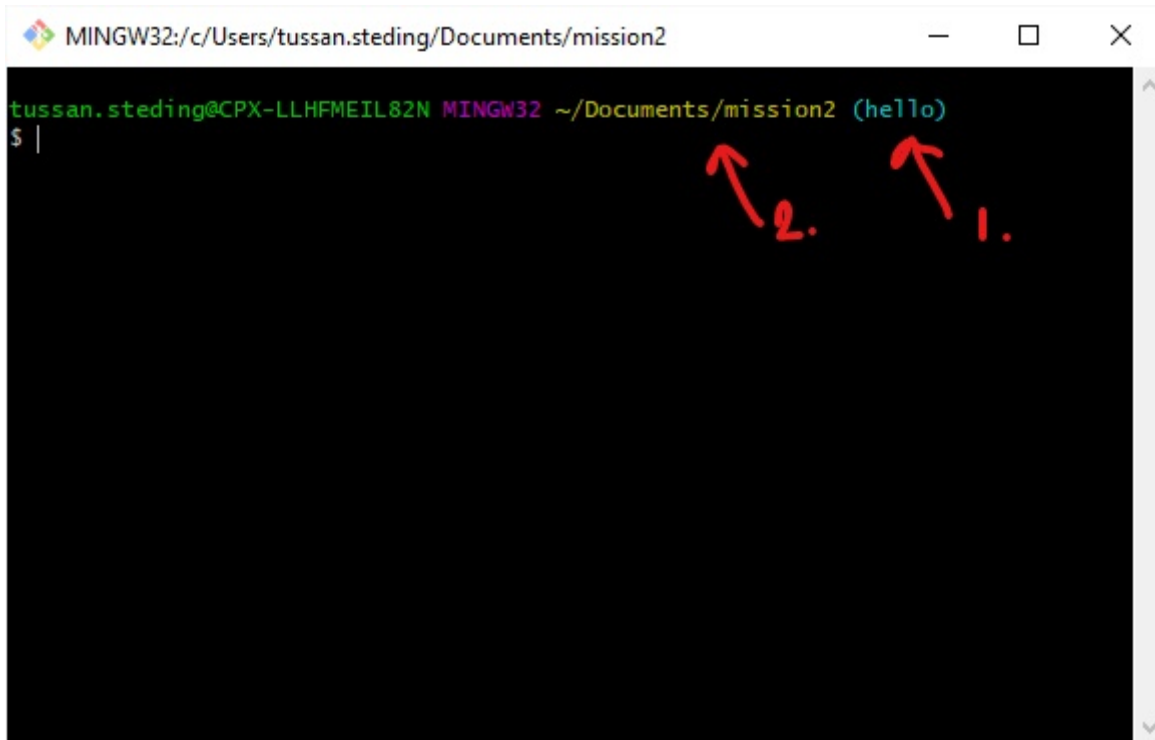
- **clone**
ローカル上に、リモートリポジトリとmasterブランチをダウンロードする。
- **branch**
現在のブランチを複製する。複製したブランチのプログラムに修正を加えると、複製元とは独立して更新される。
- **checkout**
現在操作しているブランチを変更する。
- **fetch**
リモート上で更新が進んでいる情報を取得する。
- **pull**
リモート上の更新情報を取得し、それを現在のローカルブランチに反映する。つまりローカルブランチを、リモートの状態と同じに更新する
- **init**
これを実行すると、現在のフォルダがリポジトリになり、git管理ができるようになる。
- **commit**
ファイルを更新した内容を確定する。これを実行して初めて、実際のファイルだけでなく、git管理上でも更新が反映される。
- **add**
更新したファイルのうち、commitしたいファイルを選ぶ。
- **merge**
他のブランチの更新内容を、現在のブランチに反映させる。

- **push**

ローカルブランチの更新内容を、リモートブランチに反映する。

CUI (Command User Interface)

Terminal-Windowsのコマンドプロンプトと同一し、SourceTreeのGUIでできる操作を全てコマンドを使って、実行可能。Windowsのコマンドプロンプトも使用可能だが、設定が必要となるため、Terminalをおすすめする。
SourceTree上でTerminalをクリックすると、入力画面が表示される。



1. 現在いるブランチ名
2. リポジトリのディレクトリ

一般のコマンド説明と使用法は以下の通り。

留意点：コマンドを実行してから、実行確認メッセージ等が出ない場合があるため、必ず確認のコマンドを実装すること。

- **Clone**

- MasterをCloneするには、リポジトリを指定する必要がある。ウェブ上でGitlabにログインし、リポジトリのページに、リポジトリURLをコピーする。
注意：SSHではなく、HTMLを選択してからリンクをコピーすること。

M **Mission2** Public Add license

Project ID: 9254180

0 Star 0 Fork HTTPS <https://gitlab.com/AcBox>   + Global

Red arrows point to the 'HTTPS' dropdown (labeled 1) and the repository URL (labeled 2).

Files (246 KB) Commits (17) Branches (3) Tags (0) Security Dashboard

Add Readme

Add Changelog

Add Contribution guide

Enable Auto DevOps

Add Kubernetes cluster

Set up CI/CD

\$ git clone リポジトリのアドレス

例 \$ git clone <https://gitlab.com/AcBootCamp/mission2.git>

• Branch

- Branchを作るには以下のコマンドを実行する。

\$ git branch ブランチの名前

例 \$ git branch Hello

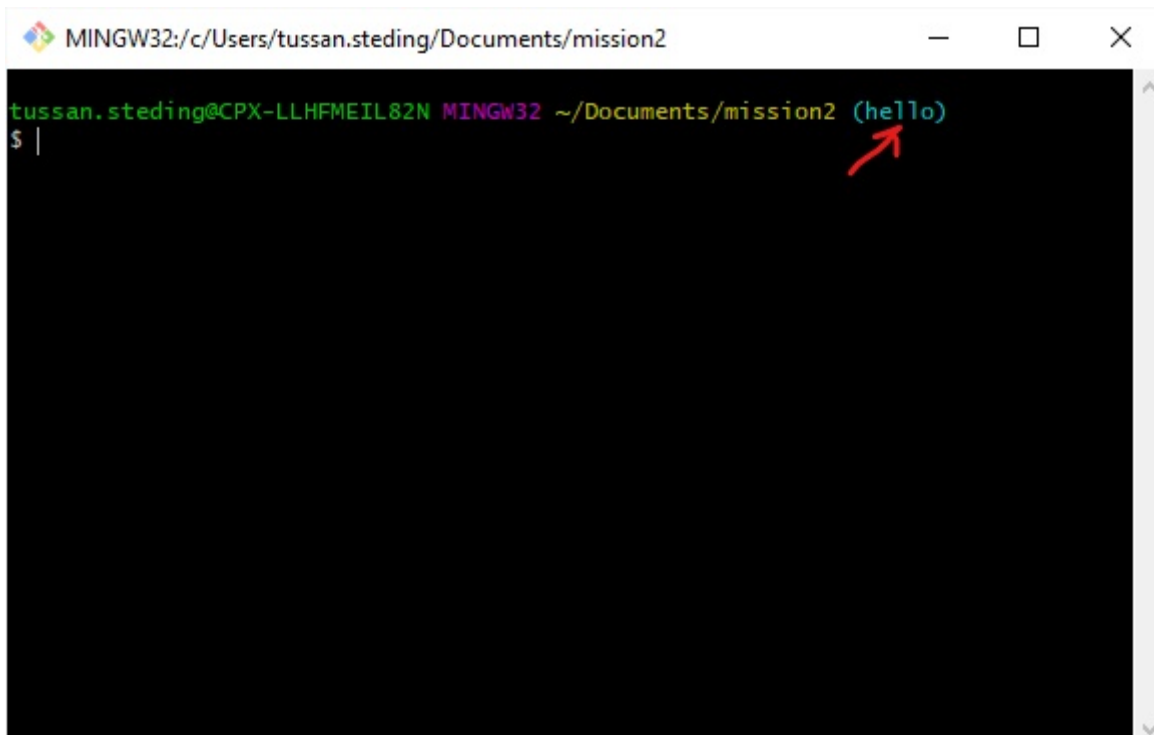
• Checkout

- Branchを切り替える場合はCheckoutを使う。

\$ git checkout 行きたいブランチの名前

例 git checkout Hello

切り替えたブランチが成功したら、Terminalにある青色の文字がブランチに変わる。



```
MINGW32:/c/Users/tussan.steding/Documents/mission2
```

```
tussan.steding@CPX-LLHFMEIL82N MINGW32 ~/Documents/mission2 (hello)
```

```
$ |
```

A red arrow points to the word "(hello)" in the terminal output, indicating the current branch name.

- **Fetch**

- リモートリポジトリと更新状況

```
$ git fetch origin
```

- **Pull**

- リモートリポジトリから修正されたファイルをローカルと同期するには以下のコマンドを使用する。

```
$ git pull origin Pullしたいブランチ名  
例 $ git pull origin Hello
```

留意点：ログイン画面が出る場合、Gitlabのログイン情報を入力し、OKを押すとPullされる。

- **Init**

- 作業ディレクトリをGitにするためのコマンド。

```
$ git init
```

- **Add**

- ローカルでファイル修正を行った場合、Commitする前にAddでCommitしたいファイルを指定しないといけない。以下のコマンドを使用すると、修正したすべてのファイルが選択される。

```
$ git add .
```

注意：addの後には必ずスペースとピリオドを入れること。

Commitするファイルを指定したい場合；

```
$ git add ファイル名  
例 $ git add textfile.txt
```

- **Commit**

- Addに追加したファイルをCommitする。

```
$ git commit -m コミットメッセージ  
例 $ git commit -m 2行目修正終了。
```

コミットメッセージ - 更新内容のコメント。

- **Merge**

- 修正したファイルをローカル又はリモートのブランチとMergeする。

```
$ git merge ブランチ名  
例 $ git merge Hello
```

リモートのブランチとMergeしたい場合；

```
$ git merge origin ブランチ名  
例 $ git merge origin Hello
```

- **Push**

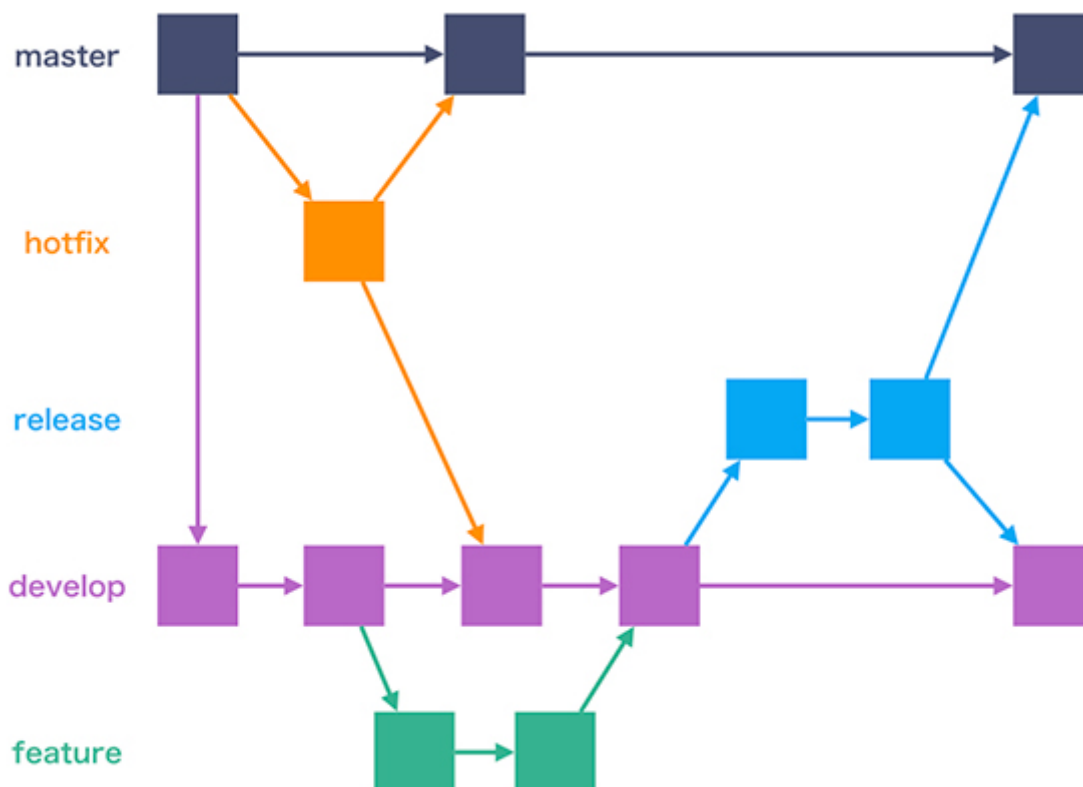
- 現在のローカルブランチをリモートのブランチと同期する。

```
$ git push origin ブランチ名  
例 git push origin Hello
```

Git Flow & GitHubFlow

Gitを使用する時の開発フローは主に2つ、GitFlowとGitHubFlow。

GitFlow



メインブランチ

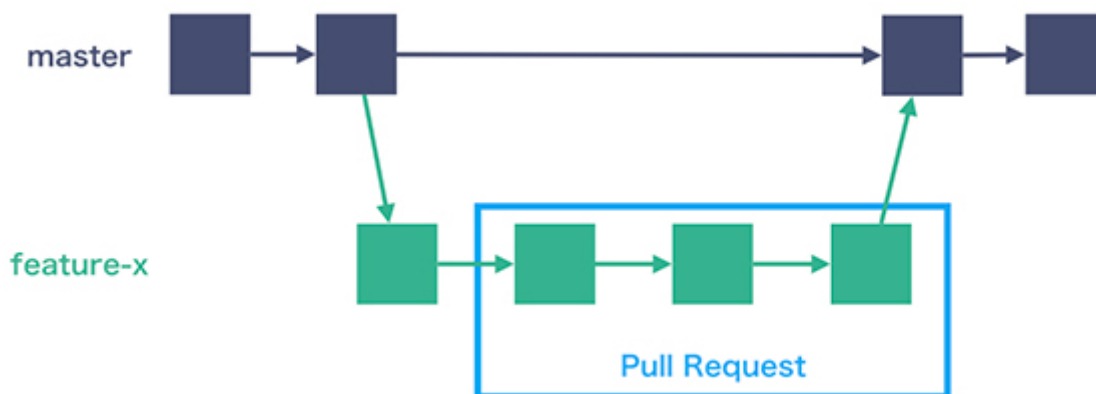
Git flowをモデルした開発では、メインブランチとそれ以外のSubブランチを使用する。メインブランチはMasterとDevelopブランチとなる。

- **Master**
 - リリースに使用するコードで、いつでもリリースできる状況。
- **Develop**
 - リリースに近い状況のコードが持っているブランチ。最終的にMasterとマージする。

Subブランチ

- **Hotfix**
 - Hotfixというのは、リリース後に緊急なバグが発生した時、すぐに行われなければならない修正のこと。Developにもっていく余裕がないため、別のブランチを使用し、修正後はすぐにMasterとマージする。
- **Release**
 - リリース用のコードが終了し、資料更新等のために使用するブランチ。
- **Feature**
 - 開発のそれぞれの部分がここで作成・修正する。修正終わったコードはDevelopにマージする。

GitHubFlow



GitHubFlowのモデルの場合、Gitflowと特に違いがないが、GitFlowより単純なモデルで、MasterとFeatureしかないため一日で複数回デプロイを行うWebアプリケーション等のような開発に使用する。

- **Master**
 - 定期的に更新される。
- **Feature**
 - 作業用ブランチ。必ずMasterから直接作成する。Developブランチはない。