

# Git基礎編

citrus88

[citrus.mikan88@gmail.com](mailto:citrus.mikan88@gmail.com)

2025-07-27

# はじめに

---

- 想定読者
  - バージョン管理を知らない人
  - これからGitを使い始める人
- ゴール
  - バージョン管理のメリットが分かる
  - Gitの特徴が分かる
  - Gitを使って個人の開発環境でバージョン管理を行うことができる

# この資料で述べないこと

---

- GUIアプリ(VSCodeやTortoiseGit)などの操作方法
- リモートリポジトリの使い方
- チーム開発でのGitの利用方法
- Gitの応用的な使い方

# 本資料の概要

---

- バージョン管理の意義
- Gitの重要な概念
- Gitの操作例
- Gitの基本的なコマンド

# バージョン管理の意義

---

- 変更履歴をいつでも遡ることができる
  - だれが・いつ・なにを・なぜ変更したのかが分かる
  - いつでも自由に過去のバージョンに変更することができる
- 複数のバージョンを並行して管理できる
  - 開発版と安定版を同時に維持できる
  - 実験的な変更を容易に試すことができる
- 共同作業の効率化
  - 各人がそれぞれの環境で作業し、その成果を簡単に統合することができる

# Gitの特徴

---

- 分散型
  - 各開発者が自身のローカル環境でバージョン管理を行うことが可能
- 高いパフォーマンス
  - あらゆる作業を非常に高速で実行可能
- オープンソースソフトウェア
  - ライセンスは, [GNU General Public License version 2.0](#)

# 補足: なぜGitを使うのか

---

- バージョン管理のデファクトスタンダードであるから。
  - 情報が非常に多い。
- GitLab/GitHubといった便利なホスティングサービスがあるから。
  - Gitリポジトリをリモートで共有できるサービス。
  - 共有以外にも開発に役立つ様々な機能(イシュー, マージリクエストなど)を持つ。

# Gitの重要な用語

---

- リポジトリ(repository)
  - 変更履歴を保存するための領域、貯蔵庫。
- コミット(commit)
  - 変更履歴をリポジトリに保存すること
  - 全てのコミットにはコミットID(SHA-1ハッシュ)が付与され一意に定まる。
- ブランチ(branch)
  - 開発における経路の一つ
  - ブランチを作り作業し、mainブランチへ統合することで開発を進める。

# リポジトリに変更履歴を保存するまでの流れ

- ・ ワークツリー: 現在作業している場所.
- ・ ステージングエリア: コミットする前の変更内容を一時的に保存する場所.
- ・ リポジトリ: 変更履歴を保存する場所.



# Gitをインストールする(Windows)

---

- 公式サイトより、ダウンロードする。
- インストール時に設定を選ぶ必要がある。基本的にはRecommendedにして、以下は変更すると良い。
  - Adjusting the name of the initial branch in new repositories項目について
    - Override the default branch name for new repositories を選択し mainに変更する
    - 理由: 2020年10月1日に、GitHubのデフォルトブランチ名がmasterからmainに変更になり、それに合わせるため。

# Gitをインストールする(Linux)

---

- [公式サイト](#)にしたがって、コマンドでインストールする。
- その後パスを通す。

# 初期設定を行う

最初にGitに自分の名前とメールアドレスを設定する。

これらの情報はコミット履歴にユーザ情報を載せるためだけであり、リポジトリ内の情報以外で使われることはない。

```
git config --global user.name "あなたの名前"  
git config --global user.email "あなたのメールアドレス"
```

Gitの設定を確認するコマンドは以下になる。

```
git config --list
```

# ヘルプを見る

---

以下のコマンドでヘルプを見ることができる。  
オフラインでも使用可能である。

```
git コマンド名 --help  
git help コマンド名
```

# Gitを使った個人開発におけるバージョン管理の流れ

1. (初回のみ)リポジトリを作成する( `git init` )
2. (初回のみ)適当にファイルを作りコミットする(初回はコミットしないと、ブランチが作れないため)
3. 作業ブランチを作成する( `git branch` )
4. ファイルを編集する
5. 履歴を保存する( `git add` -> `git commit` )
6. 4と5を繰り返し、開発を進める。
7. (今回のブランチで作業が完了すると) `main` ブランチにマージする( `git merge` )
8. 3から再度実施し、開発を進める

# Gitの実例0

---

これから5ページ間、 実際のGitのコマンド操作方法を紹介する。

WindowsならGit Bashを、 Linuxならターミナルを開き、 実際にコマンドで操作してみてほしい。

始める前に以下の事前準備が完了しているかを確認すること。

- Gitをインストールしている
- Gitの初期設定を実施している
- 作業ディレクトリを作る

# Gitの実例1

---

1. ターミナル上で作業ディレクトリに移動する
2. リポジトリを作成する。プロジェクト名は git-tour とする
  - i. `git init git-tour`
3. プロジェクトに入る
  - i. `cd git-tour`
4. プロジェクトの中身を確認する。`.git` ディレクトリがあるはずである。
  - i. `ls -al`

# Gitの実例2

---

5. 適当なファイルを作る

- i. echo Hello > sample.txt
- ii. cat sample.txt

6. ファイルをコミットする

- i. git add sample.txt
- ii. git commit -m "first commit"

7. ログを確認する

- i. git log

# Gitの実例3

---

8. devブランチを作成する

i. `git branch dev`

9. devブランチへ変更する

i. `git switch dev`

10. ファイルを編集する

i. `echo World >> sample.txt`

11. 変更をコミットする

i. `git add sample.txt`

ii. `git commit -m "add World"`

# Gitの実例4

---

12. 変更履歴を確認する

i. git log

13. mainブランチに戻る

i. git switch main

14. mainブランチにおけるファイルの情報を確認する

i. cat sample.txt

# Gitの実例5

---

17. devブランチの変更をmainブランチにマージする

i. `git merge dev`

18. 変更履歴を確認する( `--all` をつけることで全てのブランチを表示する)

i. `git graph --all`

以降のページでは、今回用いたコマンドを説明する。

# リポジトリを作成する

---

以下のコマンドを実行することでリポジトリを作成できる。

出来たフォルダ内に `.git` フォルダができ、その中に履歴情報が格納されている。

```
git init フォルダ名
```

# 変更履歴を保存する

まず、変更情報をステージングエリアに追加する。

```
git add ファイル名
```

その後、コミットする。

その時、変更内容の説明に何を変更したか分かりやすく記述する。

```
git commit -m "変更内容の説明"
```

# 変更箇所を確認する

次のコマンドで変更箇所を確認できる。

- `git diff`
- `git diff --staged`



# 変更履歴を確認する

```
git log # デフォルト  
git log --oneline # 変更履歴を簡潔に表示する
```

オプションを設定することで表示する情報を変更できる。

オプションの詳細は[Git Documentation](#)が参考になる。

# ブランチを作成する

ブランチを作成するには、以下のコマンドを用いる。

```
git branch ブランチ名
```

作成したブランチに移動するには、以下のコマンドを用いる。

```
git switch ブランチ名
```

# マージする

---

マージとはブランチ同士の変更履歴を統合することである。  
マージ後はコミットと同様に、メッセージを書く。

```
git merge マージ元のブランチ
```

マージ元とマージ先のブランチにおいて同一箇所が変更された場合、競合(conflict)が起こり、手動での解決が必要になる。

# 変更履歴を遡る

---

過去の変更履歴を復元したい場合に用いる。

```
git checkout コミットID
```

コミットIDは、`git log` を使用することで調べることができる。

# まとめ

---

- Gitを使うことで簡単にバージョン管理ができる.
- コミットすることでリポジトリに変更内容を保存する.
- ブランチを作成することで独立した作業を行うことができる.

# おすすめの書籍

---

- [Git Pro](#)
- [独習Git](#)