

# Git勉強会

# バージョン管理入門

## なぜバージョン管理をするのか

- 変更を追跡できる: 誰が、いつ、何を変更したかを記録し、過去のバージョンに戻ることができる。
- バックアップとしての機能: 誤ってファイルを削除したり、壊したりしても、以前のバージョンに簡単に戻すことができる。
- ブランチを利用した開発: 複数の開発者が独立して作業を進め、必要に応じてマージできる。これにより、機能ごとに独立した開発が可能になる。

## バージョン管理すべきでないもの

例えば、パスワードといった機密情報はバージョン管理してはいけません。  
代わりにGitHub Secrets、AWS SSM Parameter Storeなどの目的に特化した場所に保管しましょう。

これはリポジトリの可視性に依りません。  
プライベートなりポジトリであっても、ソースコード流出時の被害を最小限に抑える必要があります。

被害事例: <https://www.bbc.com/news/technology-42075306>

# Git入門

# Gitコンポーネント

- Gitサーバー
  - Gitリポジトリのホスティングプラットフォーム
    - ex. GitHub, GitLab
- Gitクライアント
  - Gitコマンドライン
    - 今回のメインコンテンツ
    - 単にGitと言う場合これを指す
  - Git GUIツール
    - ex. Sourcetree

## バージョンの確認

```
$ git --version  
git version 2.45.2
```

## ブランチの操作

- ブランチを切り替える

```
$ git switch develop  
# または  
$ git checkout develop
```

- 新しくブランチを切る

```
$ git switch -c fix-bug develop  
# または  
$ git checkout -b fix-bug develop
```



## checkoutとswitchのどちらを使うべきか

ブランチ操作を扱う分にはどちらでもいいです。

ただしswitchは実験的なコマンドであるため、その動作は今後変わる可能性があることに注意しましょう。

<https://git-scm.com/docs/git-switch/2.45.2>

## 変更をステージする

ステージングとは、コミットによってスナップショットを撮る準備段階のこと。

- ファイル単位

```
$ git add outline.md
```

- ディレクトリ単位

```
$ git add images/
```

## `git add .` は使わない

`git add .` はカレントディレクトリ配下の変更をすべてステージします。これに慣れてしまうと何をステージ・コミットするかを意識しなくなるため、1つのコミットが大きくなったり、機密情報をコミットするリスクが高まります。

代わりに前述の `git add outline.md` のように特定ファイルのみをステージする癖を付けましょう。

## 変更を元に戻す

- 作業ディレクトリの変更を破棄する

```
$ git restore --worktree outline.md  
# または  
$ git restore outline.md  
# または  
$ git checkout outline.md
```

- ステージを取り消す

```
$ git restore --staged outline.md  
# または  
$ git checkout outline.md
```

## checkoutとrestoreのどちらを使うべきか

作業状態の復元をする分にはどちらでもいいです。

ただしrestoreは実験的なコマンドであるため、その動作は今後変わる可能性があることに注意しましょう。

<https://git-scm.com/docs/git-restore/2.45.2>

## コミット

```
# 最も基本的な使い方
$ git commit -m "awesome commit message"
# コミットメッセージは複数行書いても良い
# デフォルトではvimが採用される
$ git commit
```

## 変更をリモートリポジトリに反映させる

```
# 最も基本的な使い方  
# リモートリポジトリに add-awesome-feature ブランチがあれば更新  
# なければ作成する  
$ git push origin add-awesome-feature
```

- 通常リモートリポジトリの参照は `git clone` 時に `origin` として追加されている
  - リモートリポジトリの参照名は `git remote` で確認できる

## リモートリポジトリの変更をダウンロードする

リモート追跡ブランチ(※)の一覧を `git fetch` の前後で確認する。

※事前にGitHub上でブランチ `fetch-test` を作成しておく。

```
$ git branch -r
  origin/main
$ git fetch
From https://github.com/wonda-tea-coffee/git-study
* [new branch]      fetch-test -> origin/fetch-test
$ git branch -r
  origin/fetch-test
  origin/main
```

※リモート追跡ブランチ・・・リモートの変更を追跡するためのブランチ



# 作業状態を退避させる

## 退避

```
$ git stash push  
# または単に  
$ git stash
```

## 復元

```
$ git stash pop
```

- `git stash list` でstashした一覧を確認したり、stash push時にメッセージを追加できますが、複数stashすること自体あまり推奨しません。
- 複数stashしないといけないくらい並行して作業すること自体効率的ではありませんし、一旦コミットしてしまえばそもそも使う必要がありません。

## 変更を部分的に取り込む

```
$ git cherry-pick [commit]
```

- 別のブランチのコミット単体を取り込むことができます。
- メインのブランチとは別に、大掛かりな変更を別ブランチで行う際に使うことがあります。
  - ex. フレームワークのアップデート

## 変更履歴を探す

```
$ git log
# 変更されたコードも出力する
$ git log -p
# 特定のファイルパターンの履歴に絞る
$ git log -- README.md
# 逆順にもできる
$ git log --reverse
```

## 変更を取り消す

```
$ git revert [commit]
```

## コミットをまとめる

```
# HEADから3つのコミットをまとめる  
$ git rebase -i HEAD~3
```

## コンフリクトの解消ハンズオン

```
[main]$ echo "version: 1" > README.md
[main]$ git add README.md
[main]$ git commit -m "add README.md(in main)"

[feat]$ git switch -c feat main
[feat]$ echo "version: 2" > README.md
[feat]$ git commit -am "fix README.md(in feat)"

[main]$ git switch main
[main]$ echo "version: 3" > README.md
[main]$ git commit -am "fix README.md(in main)"

[feat]$ git switch feat
[feat]$ git merge main
# コンフリクトの解消
[feat]$ git add README.md
[feat]$ git merge --continue
```

## 快適にGitを使うためのTips

## Gitの操作に困ったら

- ドキュメントを読む
  - <https://git-scm.com/docs/git>
  - `git help -a`
  - `git [command] -h`
- 「git [command] 使い方」でググる

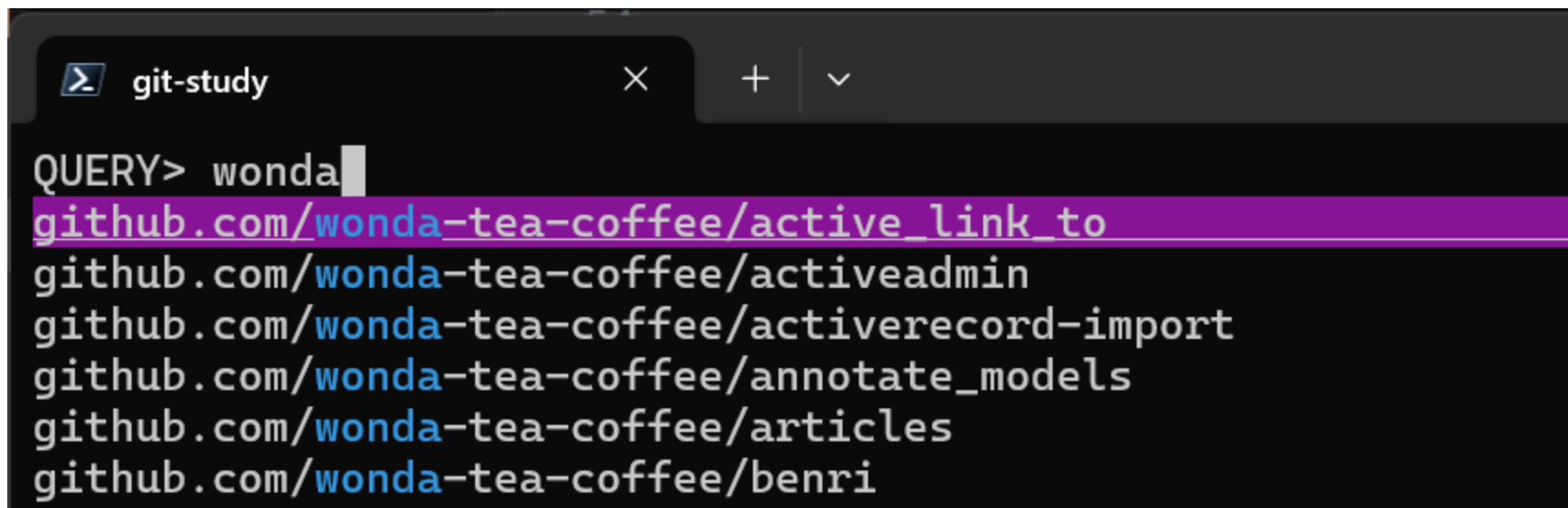


## ターミナルにリポジトリ名・ブランチ名を表示する

tips1

参考: <https://qiita.com/caad1229/items/6d71d84933c8a87af0c4>

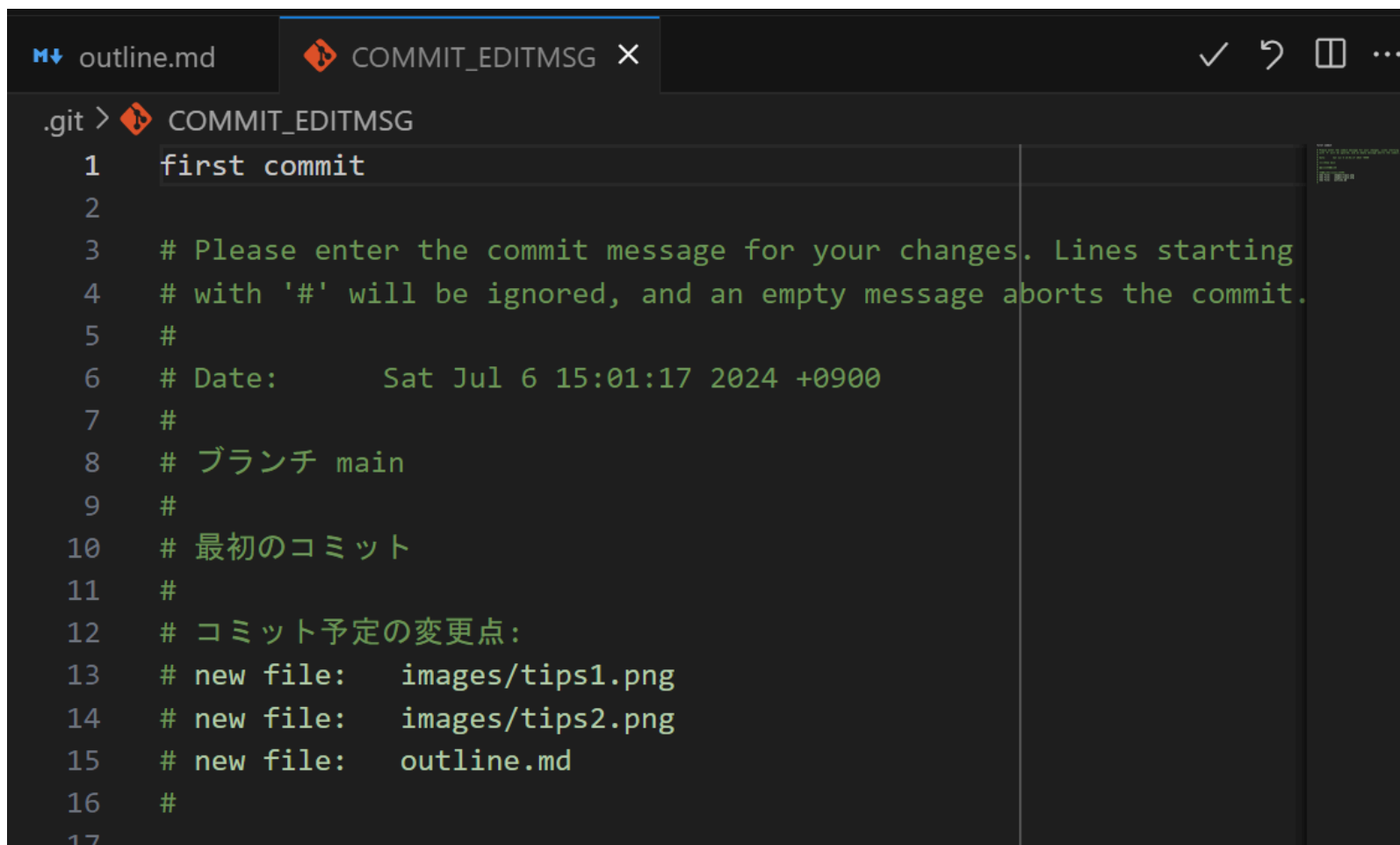
## ローカルリポジトリを検索・移動する



```
git-study
QUERY> wonda
github.com/wonda-tea-coffee/active_link_to
github.com/wonda-tea-coffee/activeadmin
github.com/wonda-tea-coffee/activerecord-import
github.com/wonda-tea-coffee/annotate_models
github.com/wonda-tea-coffee/articles
github.com/wonda-tea-coffee/benri
```

参考: <https://zenn.dev/obregonia1/articles/e82868e8f66793>

# 好きなエディターでコミットメッセージを書く



The screenshot shows a code editor with a dark theme. The top bar has a tab labeled 'COMMIT\_EDITMSG' with a close button. Below the tab, the text '.git > COMMIT\_EDITMSG' is visible. The main editing area contains a commit message template with line numbers 1 through 17 on the left. The template text is as follows:

```
1 first commit
2
3 # Please enter the commit message for your changes. Lines starting
4 # with '#' will be ignored, and an empty message aborts the commit.
5 #
6 # Date:      Sat Jul 6 15:01:17 2024 +0900
7 #
8 # ブランチ main
9 #
10 # 最初のコミット
11 #
12 # コミット予定の変更点:
13 # new file:  images/tips1.png
14 # new file:  images/tips2.png
15 # new file:  outline.md
16 #
17
```

参考: <https://zenn.dev/obregonia1/articles/e82868e8f66793>

## 熟達のヒント

- コマンドのヘルプを読む
- git configの設定
- ソースコードを読む
  - <https://github.com/git/git>
- システムコールを追いかける
  - ex. strace

**質問タイム**

## 小ネタ

- main ? master ?

