Git勉強会

Gitに慣れよう

SHIBANO Toshiki
2023/06/08

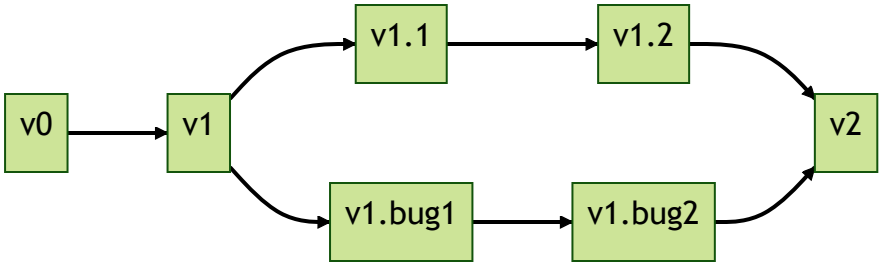
Contents

バージョン管理の概要

日々の業務でバージョンを管理する

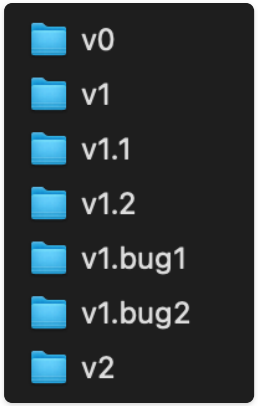
- 日々変更されるプログラム
  - 差分(変更箇所)を知りたい
  - 過去のバージョンに戻りたい
  - 並行して複数のバージョンを扱いたい
  - 分岐したバージョンを簡単に結合したい

Fig. バージョンの変遷



バージョンを管理する方法1

フォルダによるバージョン管理



- メリット
  - 特殊なソフトが不要
- デメリット
  - バージョン数だけフォルダが出来る
  - 差分の確認が難しく，変更箇所の統合も難しくなる

バージョンを管理する方法

バージョン管理ツールを用いる

- よく使われるツール
  - **Git**
    - 多くの会社で採用されている
    - GitHubの存在
  - Mercurial
  - Subversion
- メリット
  - 効率よくバージョン管理が可能
  - 色々な場面で導入されている
- デメリット
  - ツールを導入する必要がある
  - 学習コストがかかる

バージョン管理の用語

リポジトリ(repository)

- ファイルやディレトクリの状態を保存するための領域
- 変更履歴がすべて格納されている

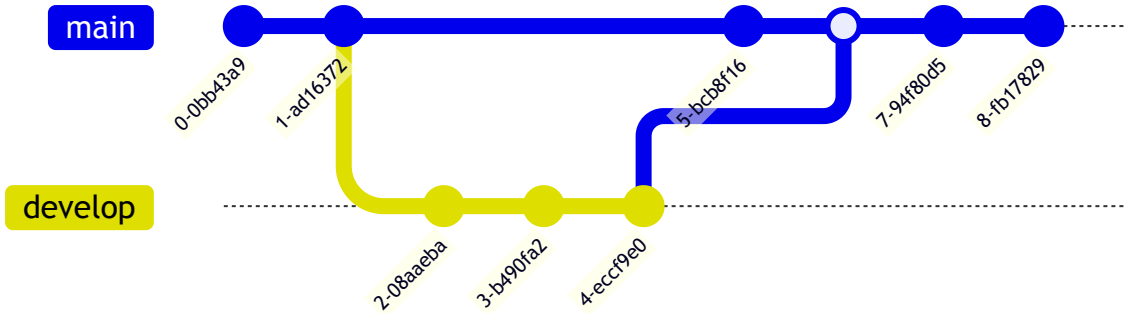
コミット(commit)

- リポジトリへ変更を保存すること
- **こまめにコミットしましょう**

ブランチ(branch)

- 開発における経路の1つ
- 新たに実装する場合は，ブランチを作って作業する

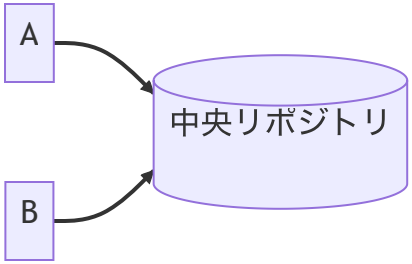
Fig. コミットとブランチ



バージョン管理ツールの分類

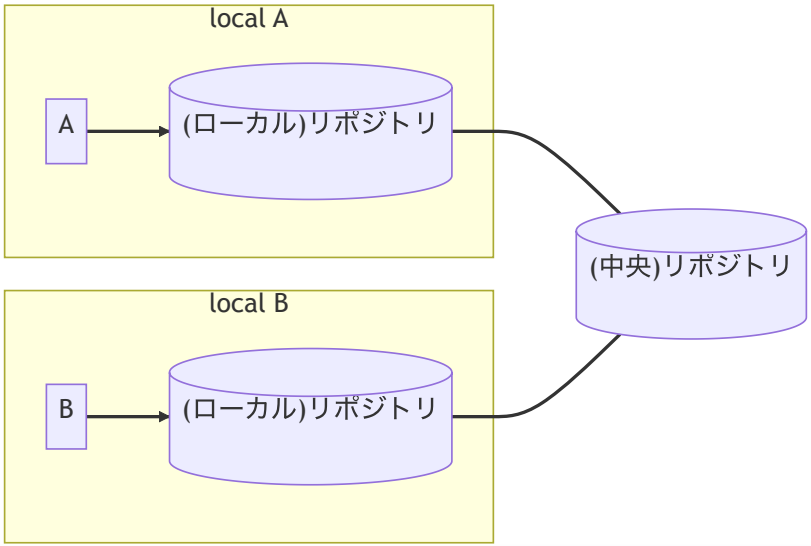
集中型バージョン管理

- サーバに唯一の中央リポジトリを持つ
- 開発者は中央リポジトリに接続してバージョン管理を行う



分散型

- 各自がローカルにリポジトリを持つ
- 開発者は自身のローカルリポジトリでバージョン管理を行う



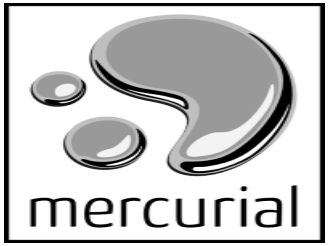
主なバージョン管理ツール

Git

- 分散型バージョン管理
- **デファクトスタンダート**
- Linus TorvaldsによってLinux開発のために開発された
- 使用例: Linuxカーネル

Mercurial

- 分散型バージョン管理
- 「履歴は永久的で神聖である」という哲学
- Pythonで作られており，Pythonで機能拡張ができる
- 使用例: Mozilla Firefox



なぜGitを使うのか

1. ネットでの情報量が多い
2. GitHub等Gitホスティングサービスの存在

Fig. Google Trendにおける， gitとmercurialの人気度比較 (2023/06/02実施)

Gitのインストール

Windows(1)

Gitのインストール

1. 公式サイトのダウンロード画面から自身のPCの環境に合わせたインストーラをダウンロードする
   - Git Download for Windows
2. 指示に従って，インストールを進める
   - デフォルトエディタの設定は，Nanoがおすすめ(Vimは使えるなら)
   - デフォルトブランチの名前の設定は，Overrideにして，mainにしていおく
     - 近年，Gitのデフォルトブランチ名をmasterからmainに変更された

Windows(2)

TortoiseGitのインストール

TortoiseGitは，GitをGUIで操作できるツールである． TortoiseGitは，エクスプローラに組み込まれているため，エクスプローラから作業する．

1. 公式サイトのダウンロード画面から自身のPCの環境に合わせたインストーラをダウンロードする
   - TortoiseGit Download
2. Language Packsもダウンロードする
3. インストーラを実行し，指示に従ってインストールする
4. インストールが完了後，日本語パックを実行し，インストールする
5. エクスプローラで右クリックをし，TortoiseGitの設定を開く
6. General -> TortoiseGit -> Language において，日本語を選択する

Linux

1. 公式サイトの指示に従ってダウンロードする
    - Git Download for Linux and Unix

Gitツアー

CUIでの操作を説明します.
Windowsの方はGit Bashを, Linuxの方はターミナルを開いてください.
GUIは各自で調べてください.

はじめに

Gitバージョン確認

```
git --version
```

ヘルプを見る

困ったらヘルプを見ましょう.

```
git help <command>
```

初期設定

ユーザの登録

名前とメールアドレスを登録する.
ただし，これによってサーバやネットワークに接続するわけではなく， 自身のGit上の設定である.

```
git config --global user.name "Your Name"
git config --global user.email "Your E-mail@example.com"
```

Gitの設定を確認する.

```
git config --list
```

リポジトリを作成する

作業ディレクトリの作成

まずは作業するディレクトリを作成する.

```
cd ~
mkdir git-tour
cd git-tour
```

リポジトリの作成

gitコマンドを使って，リポジトリを作成する.
リポジトリに必要なファイルは.gitディレクトリに格納される.

```
git init
ls -all
```

ファイルを追跡する

ファイルの作成

```
echo hoge > sample.txt
cat sample.txt
```

ファイルのコミット

`git add`でファイルをリポジトリに追加する.
`git commit`でファイルをコミットし，ファイルをリポジトリに保存する.

```
git add sample.txt
git commit -m "first commit."
```

コミット履歴の確認

```
git log
```

ブランチを作成する

ブランチを作る.

```
git branch
git branch dev
git branch
git checkout dev
```

今は，devブランチにいる． ファイルを作成し，コミットする.

```
echo fuga >> sample.txt
git add sample.txt
git commit -m "add fuga"
git log
```

mainブランチに戻り，コミットログを確認する.

```
git checkout main
git log
```

マージする

mainブランチに新しいファイルをコミットする.

```
echo foobar > sample1.txt
git add sample1.txt
git commit -m "add sample1.txt"
```

全体のコミットログを確認する.

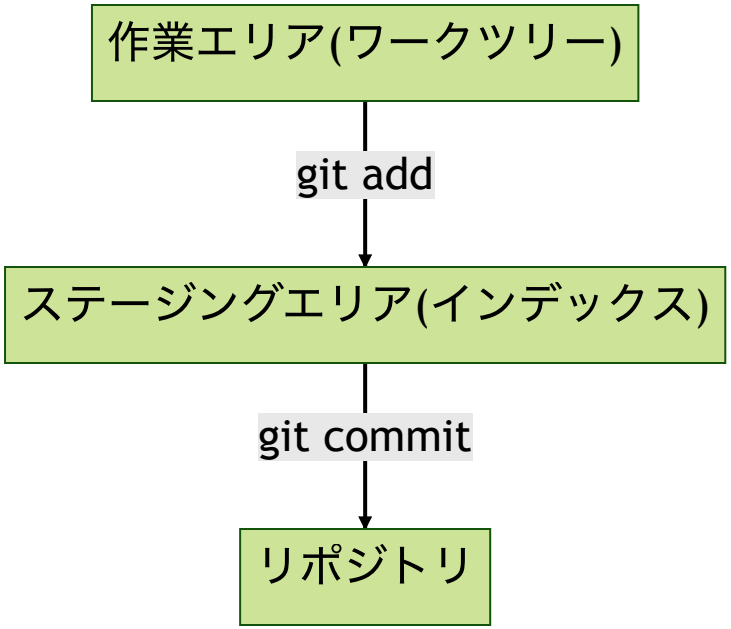
```
git log --all --oneline --graph
```

mainブランチにdevブランチをマージする.

```
git diff main...dev
git merge dev
git log --all --oneline --graph
```

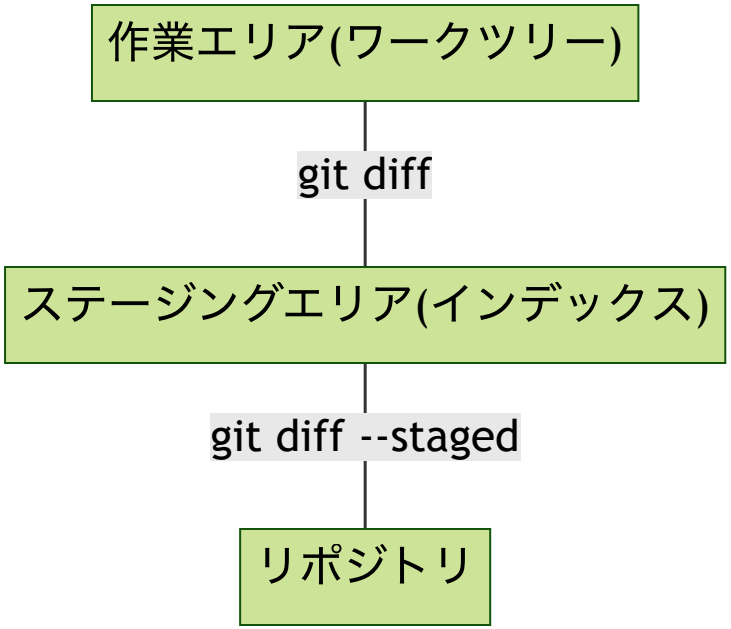
変更の追跡とコミット

ステージングエリアとリポジトリ



- 作業エリア
  - 現在作業中のディレクトリ
- ステージングエリア
  - コミットするファイルを一時的に保存される場所
  - 変更箇所の確認やコミット箇所の選別のために使われる
- リポジトリ
  - ファイルの変更履歴を保存する場所

変更箇所を確認する



- `git diff`
  - 作業エリアとステージングエリアの変更点を確認する
- `git diff --staged` (`--cashed`も同じ意味)
  - ステージングエリアと最新のリポジトリとの変更点を確認する

ファイルの削除とファイル名の変更

ファイルの削除

- `git rm`
  - 作業エリアのファイルを削除し，ステージングエリアへ削除したファイルを反映する
  - `rm`だけでは，ステージングエリアに反映されないため，`git rm`が必要になる

ファイル名の変更

- `git mv`
  - `mv`と`git rm`と`git add`をまとめたコマンド
  - ファイル名を変更，削除した旧ファイルと作成した新ファイルをステージングエリアに反映

部分的コミットとステージングエリアのリセット

部分的コミット

- `git add -p`
  - ステージングエリアに追加する箇所を選べる
  - ハンク(変更した塊)ごとに選ぶ or エディタで選択する

ステージングエリアのリセット

- `git reset`
  - ステージングエリアの状態をリセットする
  - 作業エリアの変更箇所はリセットされない

ログを見る

ログを見る時によく使うコマンド

- `git log`

```
commit:SHA1 ID
Author: Name <>
Date:

    commit massage.
```

すべてのコミットには一意なSHA1 IDが与えられ，これがコミットの名前になる.

さまざまなスイッチ

- `--oneline`
  - 短縮SHA1 IDとコミットメッセージを表示する
- `--graph`
  - コミット履歴をグラフ構造で表示する
- `--all`
  - すべてのブランチを表示する
- `--since="<date>"` or `--after="<date>"`
  - 特定の日付よりも新しいコミットを表示する
- `--until="<date>"` or `--before="<date>"`
  - 特定の日付よりも古いコミットを表示する

ブランチの作成とマージ
クローン

ブランチの作成

- `git branch`
  - 現在のブランチを確認する
- `git branch XXX`
  - XXXという名前のブランチを作成する
- `git checkout XXX`
  - XXXというブランチに変更する
- `git branch -d XXX`
  - XXXというブランチを削除する
- `git checkout -b XXX`
  - XXXというブランチを作成し，変更する

マージ

- 複数のブランチを統合すること
- `git merge YYY`
  - 今のブランチにブランチYYYをマージする
  - 実行時にはマージメッセージが求められる
  - 自動的にマージされるが，競合(コンフリクト)がある場合，競合を解消する必要がある
- `git diff XXX...YYY`
  - ブランチXXXを基準にブランチYYYとの差分を表示する

クローン

- リポジトリのコピーを作ること
- `git clone PATH TO_DIR`
  - パスPATH(URL/ローカルディレクトリ)にあるリポジトリをディレクトリTO_DIRにコピーする

次のステップ

触りながらGitを覚えてましょう

まだ説明できていないこと

- GUIの使い方
- ベアリポジトリ
- プッシュ(push)やプル(pull)
- `git rebase`によるコミットの書き換え
- GitHubの使い方

参考文献

- Git公式サイト
  - ドキュメント
  - 公式Book
- 独習Git (Rick Umali 著)
- 東工大OCW-システム開発プロジェクト応用第一