



Git 勉強会 2 日目

基本コマンド & ハンズオン① ~個人開発編~
2022/03/25

本日のゴール

頭の中に「こんなときはこうする」というインデックスをぼんやりと作ること

- Git の各サブコマンドの存在を知ること
- 各サブコマンドのユースケースを知り、Git で躊躇したときに本資料を見返そうと思い付けること

→ この資料は辞書として使っていいってほしいので、完全に理解しようとしないで OK です！



- コマンド説明(★付きコマンドのみ当日説明します)
 - clone ★
 - config ★
 - init
 - remote
 - branch ★
 - switch (checkout) ★
 - status ★
 - add ★
 - commit ★
 - push ★
 - mv
 - rm
 - log ★
 - diff
- ハンズオン
 - 【参考】ターミナルでよく使うコマンド ①
 - 【参考】ターミナルでよく使うコマンド ②
 - ハンズオン ①:既にリモートリポジトリにあるリポジトリをコピー
 - ハンズオン ②:Git の設定
 - ハンズオン ③:ブランチの作成・移動・名前変更・削除
 - ハンズオン ④:変更をステージに追加
 - ハンズオン ⑤:変更を記録してリモートリポジトリへ送信

コマンド説明(★ 付きコマンドのみ当日説明します)



スライドの基本構成

Git サブコマンド名(ex. clone)

機能

- コマンドの役割・機能の説明

ユースケース

- どんなときに使うか(逆引き用)

イメージ

- 内部構造を理解するため図で説明

主なオプション

- よく使うオプション

コマンド例

- よく使うコマンドの例

備考

- 注意点等(あれば)

参考

- 公式リファレンス + 分かりやすい記事（理解を深めたい人へのおすすめ記事）

clone ★

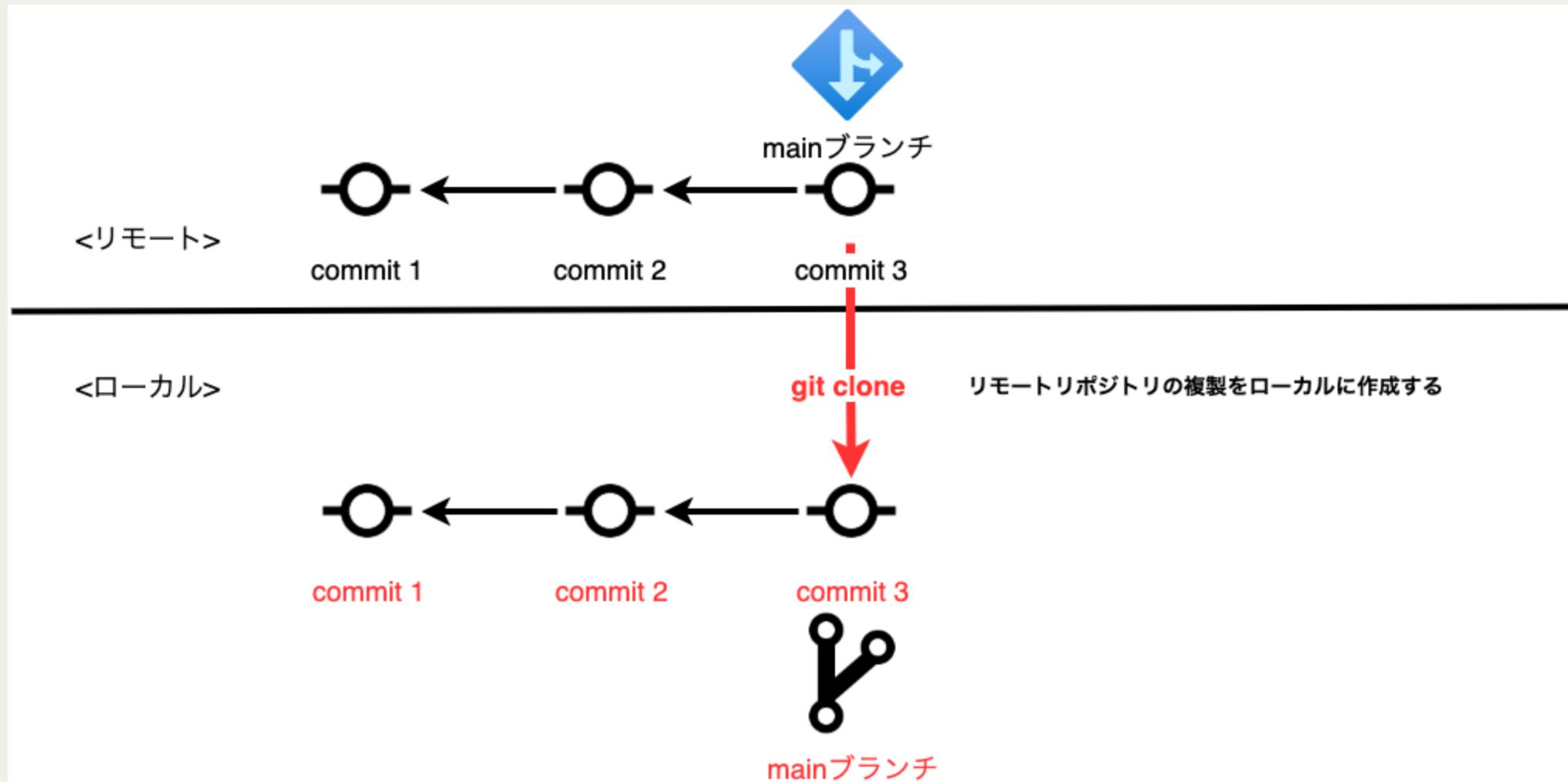
機能

- リモートリポジトリの複製をローカルに作成する

ユースケース

- 既に存在するリモートリポジトリ(GitHub/GitLab)のソースコードをローカルに複製したい

イメージ





主なオプション

- `-b` | `--branch` : 複製したいブランチを指定する

コマンド例

```
$ git clone <リモートリポジトリの URL> # リモートリポジトリをローカルに複製  
$ git clone <リモートリポジトリの URL> -b <ブランチ> # ブランチを指定してリモートリポジトリをローカルに複製  
$ git clone <変更後のディレクトリ名> # リモートリポジトリをローカルに複製し、ディレクトリ名を変更  
$ git clone . # 現在いるディレクトリをルートディレクトリとして複製 (git clone でディレクトリを作成しない)
```

備考

- `git clone` をした場合は `git init` は不要
 - 複製して生成されたローカルリポジトリには既に `.git` ディレクトリが存在する

参考

- [git-clone – Git コマンドリファレンス \(日本語版\)](#)
- [git clone でディレクトリを作らない](#)

config ★

機能

- 現在の Git の設定を取得、変更する

ユースケース

- git を利用し始める際に、ユーザー名やメールアドレス等を設定したい
- プロジェクトごとに複数の git アカウントを使い分けたい
- 現在の Git の設定がどうなっているのかを確認したい

コマンド例

```
$ git config # 現在いるリポジトリの Git 設定を表示  
$ git config --global user.name "<メインアカウントのユーザー名>" # デフォルトのユーザー名を設定  
$ git config --global user.email "<メインアカウントのメールアドレス>" # デフォルトのメールアドレスを設定  
  
$ cd /path/to/repository # 特定のプロジェクトのローカルリポジトリへ移動  
$ git config --local user.name "<サブアカウントのユーザー名>" # 特定リポジトリのユーザー名を設定  
$ git config --local user.email "<サブアカウントのメールアドレス>" # 特定リポジトリのメールアドレスを設定
```

備考

- `/path/to/repository` の箇所はユーザーごとにパスが異なるので、そのままコピペしないように注意
- `--global` の設定ファイルは `~/.gitconfig` にある
- `--local` の設定ファイルは `/path/to/repository/.git/config` にある

参考

- [git-config Documentation](#)
- [Git をインストールしたら真っ先にやっておくべき初期設定 - Qiita](#)
- [複数の git アカウントを使い分ける - Qiita](#)



init

機能

- Git ローカルリポジトリを作成する
 - .git ディレクトリ(フォルダ)が作成される

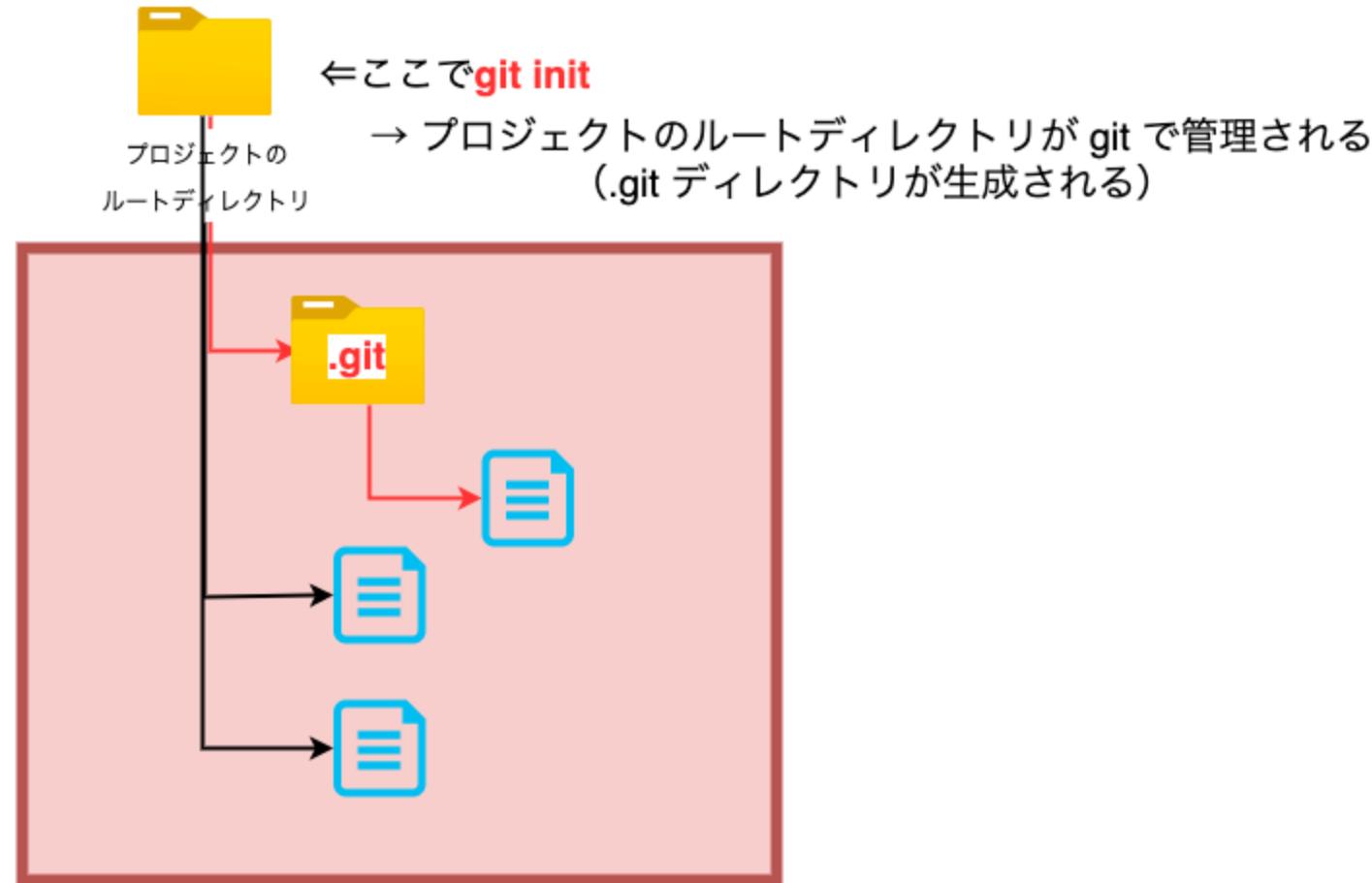
ユースケース

- Git でバージョン管理を始めたい

イメージ

<リモート>

<ローカル>





コマンド例

```
$ cd /path/to/repository # 特定のプロジェクトのルートディレクトリへ移動  
$ git init # git でバージョン管理を開始
```

備考

- git clone をした場合は git init は不要
 - 複製して生成されたローカルリポジトリには既に .git ディレクトリが存在する
- /path/to/repository の箇所はユーザーごとにパスが異なるので、そのままコピペしないように注意
- .git ディレクトリにはリポジトリを Git でバージョン管理するために必要なすべてのファイル (Git リポジトリのスケルトン) が格納されている
 - 中身の詳細については .git ディレクトリの中身を見てみる [.. - Qiita](#) を参照

参考

- [git-init – Git コマンドリファレンス \(日本語版\)](#)
- [Git リポジトリの取得 – Git コマンドリファレンス \(日本語版\)](#)
- [.git ディレクトリの中身を見てみる \[.. - Qiita\]\(#\)](#)

remote

機能

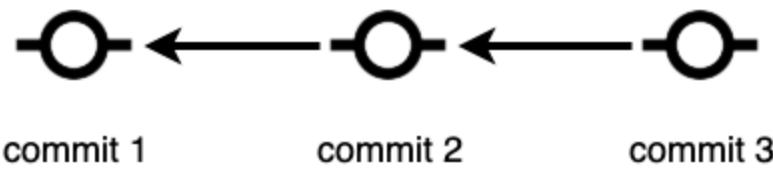
- リモートリポジトリの関連付けの設定を管理する

ユースケース

- リモートリポジトリ(GitHub / GitLab)を登録(削除)したい
- リモートリポジトリの名前と場所(URL)を確認したい
 - Fork したリポジトリなど、リモートリポジトリが複数ある場合に便利

<リモート>

<ローカル>



origin main

origin/main

`git remote add origin <リモートURL>`

ローカルリポジトリをリモートリポジトリを関連づける
(`git push origin main` が実行できるようになる)





主なオプション

- `-v | --verbose` : 詳細を表示

コマンド例

```
$ git remote add origin <リモートリポジトリのURL> # 指定したリモートリポジトリを origin という名前で管理（関連付け）する
$ git remote -v # 関連付け設定されているリモートリポジトリとその詳細を一覧表示する
$ git remote remove <リモートリポジトリのURL> # リモートリポジトリの関連付け設定を削除する
```

参考

- [git-remote Documentation](#)
- [【git remote】コマンド\(基礎編\)——リモートリポジトリを追加、削除する](#)

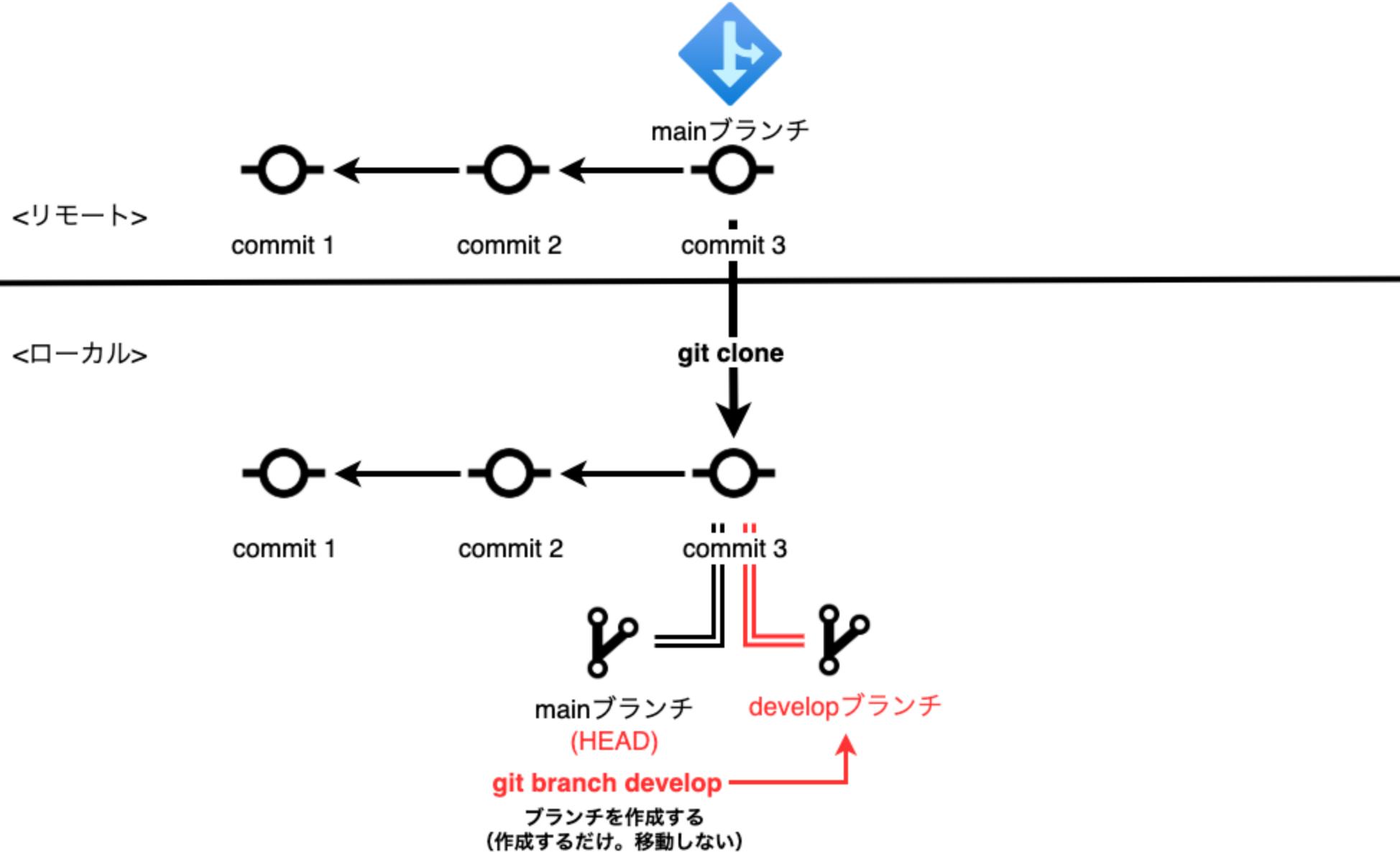
branch ★

機能

- ブランチを作成、削除、一覧表示する

ユースケース

- ローカルリポジトリにブランチを作成したい
- ローカルリポジトリにあるブランチを確認したい
- ローカルリポジトリにあるブランチの名前を変えたい
- ローカルリポジトリにあるブランチを削除したい



主なオプション

- `-a | --all` : リモート追跡ブランチを含んだブランチの一覧を表示
- `-m | --move` : 現在チェックアウトしているブランチ名を指定したブランチ名で変更
- `-d | --delete` : 指定したブランチを削除(`-D` で強制削除)
 - 指定したブランチに push していないコミットが残っている場合はエラーとなり削除できない
 - その場合は push するか、`-D` で強制削除

コマンド例

```
$ git branch -a # ローカルとリモートにあるブランチ一覧を表示  
$ git branch <ブランチ名> # ブランチを作成  
$ git branch -m <変更後のブランチ名> # 今いるブランチ名を変更  
$ git branch -d <ブランチ名> # 指定したブランチを削除。-D で強制削除
```

参考

- [git-branch – Git コマンドリファレンス\(日本語版\)](#)
- [git branch コマンド - Qiita](#)

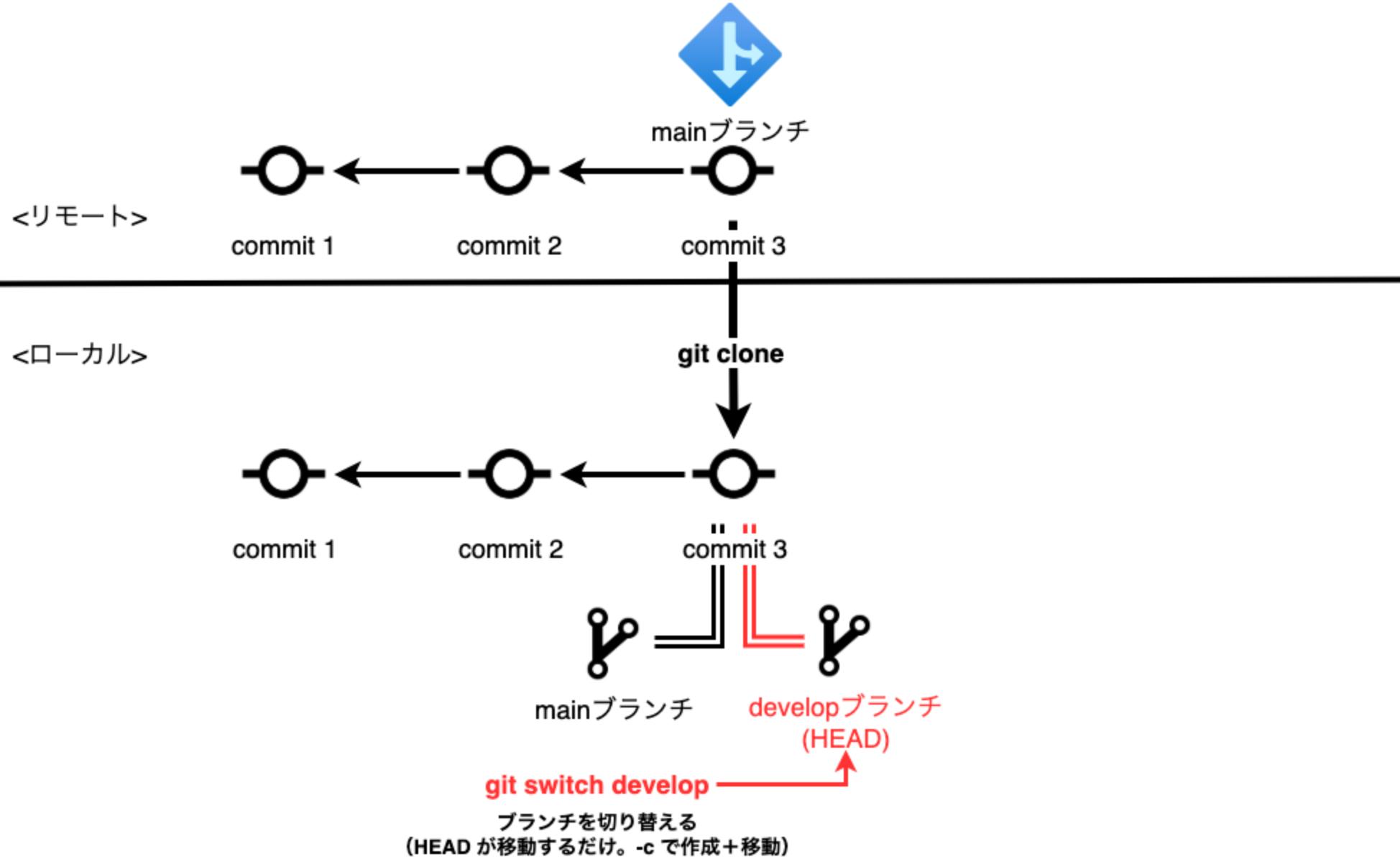
switch (checkout) ★

機能

- ワークツリーを異なるブランチに切り替える

ユースケース

- 既に存在するブランチに移動したい
- 新しいブランチを作成したい(そのまま移動したい)



主なオプション

- `-c | --create` : ブランチを新規作成して切り替え

コマンド例

```
$ git switch <ブランチ> # <ブランチ> で指定したブランチに移動する。git checkout <ブランチ> と同じ  
$ git switch -c <ブランチ> # <ブランチ> で指定したブランチを新規作成して、移動する。git checkout -b <ブランチ> と同じ
```

備考

- `switch` は Git バージョン 2.23.0 でリリース (2019/08/16)
- `checkout` は複数の役割を兼ね備えてしまっているため、こちらの方が直感的に理解しやすい

参考

- [git-switch – Git コマンドリファレンス\(日本語版\)](#)
- [git checkout の代替としてリリースされた git switch と git restore](#)

status ★

機能

- ワークツリーにあるファイルの状態を表示する

ユースケース

- どのファイルを変更したのか、add, commit 濟かどうかを知りたい
- コンフリクトしたのでどうすればいいか知りたい
- よく分からぬけどエラーになったから対処方法を知りたい

主なオプション

- `-s` | `--short` : 短い形式で表示

コマンド例

```
$ git status  
$ git status -s
```

参考

- [git-status – Git コマンドリファレンス\(日本語版\)](#)
- [git status -s でちょっと幸せになれる - Qiita](#)

add ★

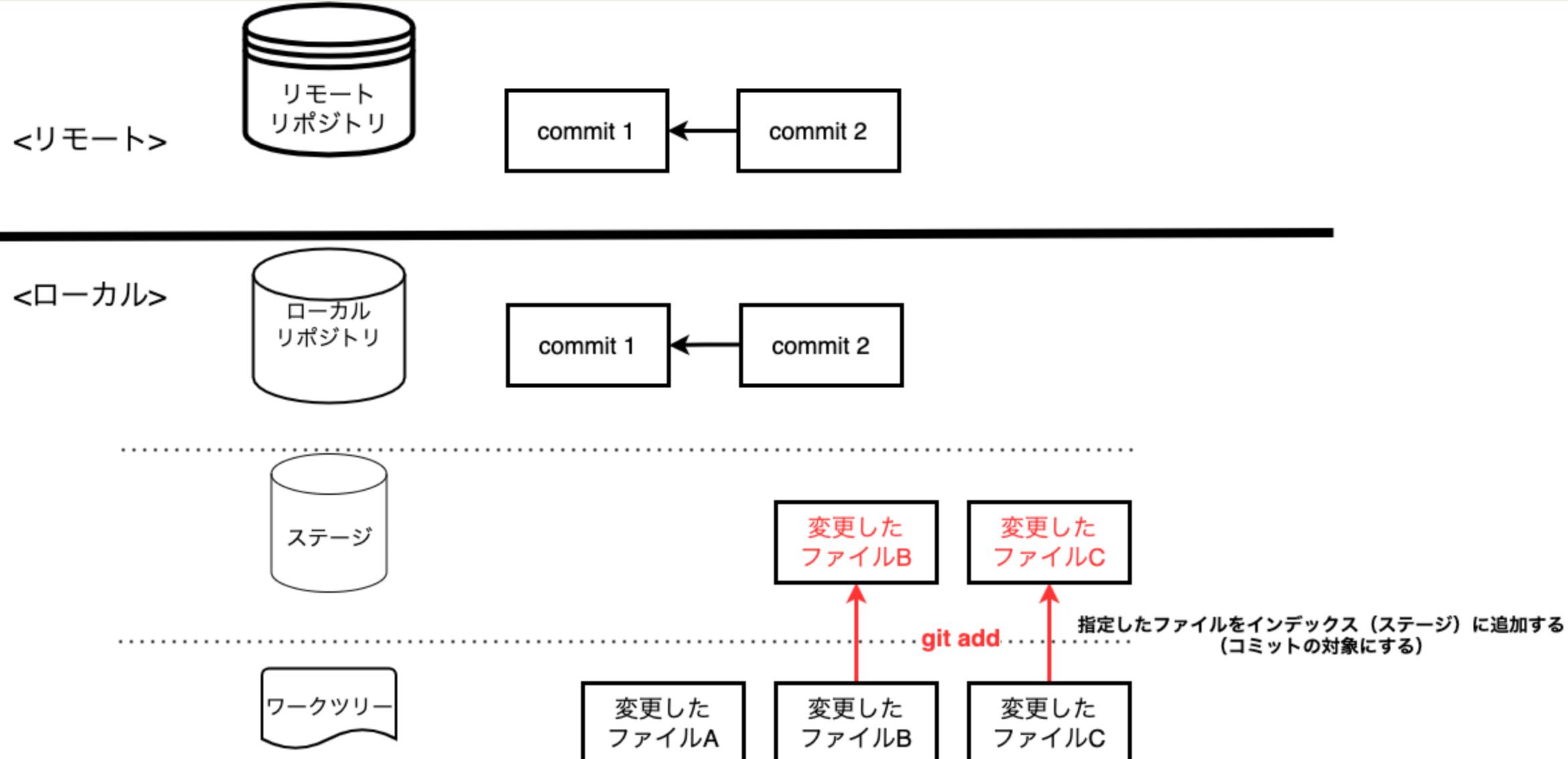
機能

- ファイルをインデックスに追加(ステージング)する(コミットの対象にする)

ユースケース

- 修正を入れた複数のファイルのうち、一部をコミット対象(インデックスに追加)したい
- 修正を入れた全てのファイルをコミット対象(インデックスに追加)したい
- コンフリクトを解決したファイルをコンフリクト解決済の状態に変更したい

イメージ



主なオプション

- `-p | --patch` : ファイル内の任意の変更行のみインデックスに追加

コマンド例

```
$ git add ./src/index.html # ./src/index.html のみインデックスに追加
$ git add ./src/ # ./src ディレクトリ以下の全てのファイルをインデックスに追加
$ git add . # 変更済の全てのファイルをインデックスに追加
$ git add *.java # * (ワイルドカード) で特定の文字列にマッチするファイルをインデックスに追加 (この場合は .java ファイル)
$ git add -p ./src/index.html # ./src/index.html の一部の変更行をインデックスに追加 (インタラクティブモードで選択する)
```

備考

- `add -p` による部分ステージングは Visual Studio Code などのエディタ機能を使ってやるのがおすすめ

参考

- [git-add – Git コマンドリファレンス \(日本語版\)](#)
- [【git add】コマンド——変更内容をインデックスに追加してコミット対象にする](#)
- [git add -p 使ってますか？](#)
- [VSCode で git add -p を快適に行う](#)

commit ★

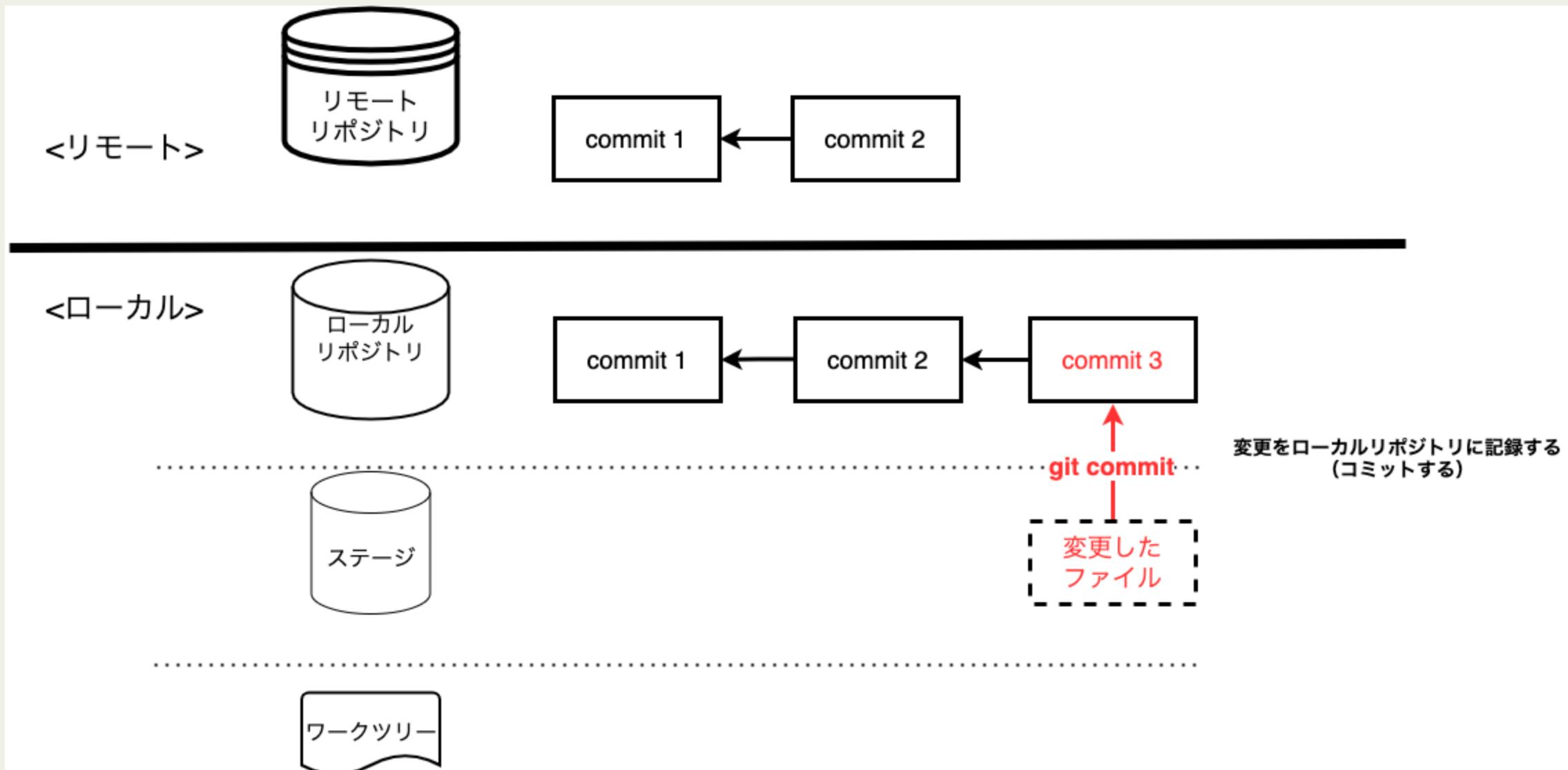
機能

- ・ インデックスに追加した変更をローカルリポジトリに記録する

ユースケース

- ・ 変更した内容を記録(コミット)したい
- ・ 直前のコミットメッセージを修正したい

イメージ



主なオプション

- `-m` | `--message` : コミットと同時にコミットメッセージを記録する

コマンド例

```
$ git commit  
$ git commit -m "<メッセージ>"  
$ git commit --amend -m "<修正後のメッセージ>"
```

備考

- 2つ以上前のコミットを修正したい場合は `git rebase -i` を利用する

参考

- [git-commit – Git コマンドリファレンス\(日本語版\)](#)
- コミットの修正には `git commit --amend` が便利

push ★

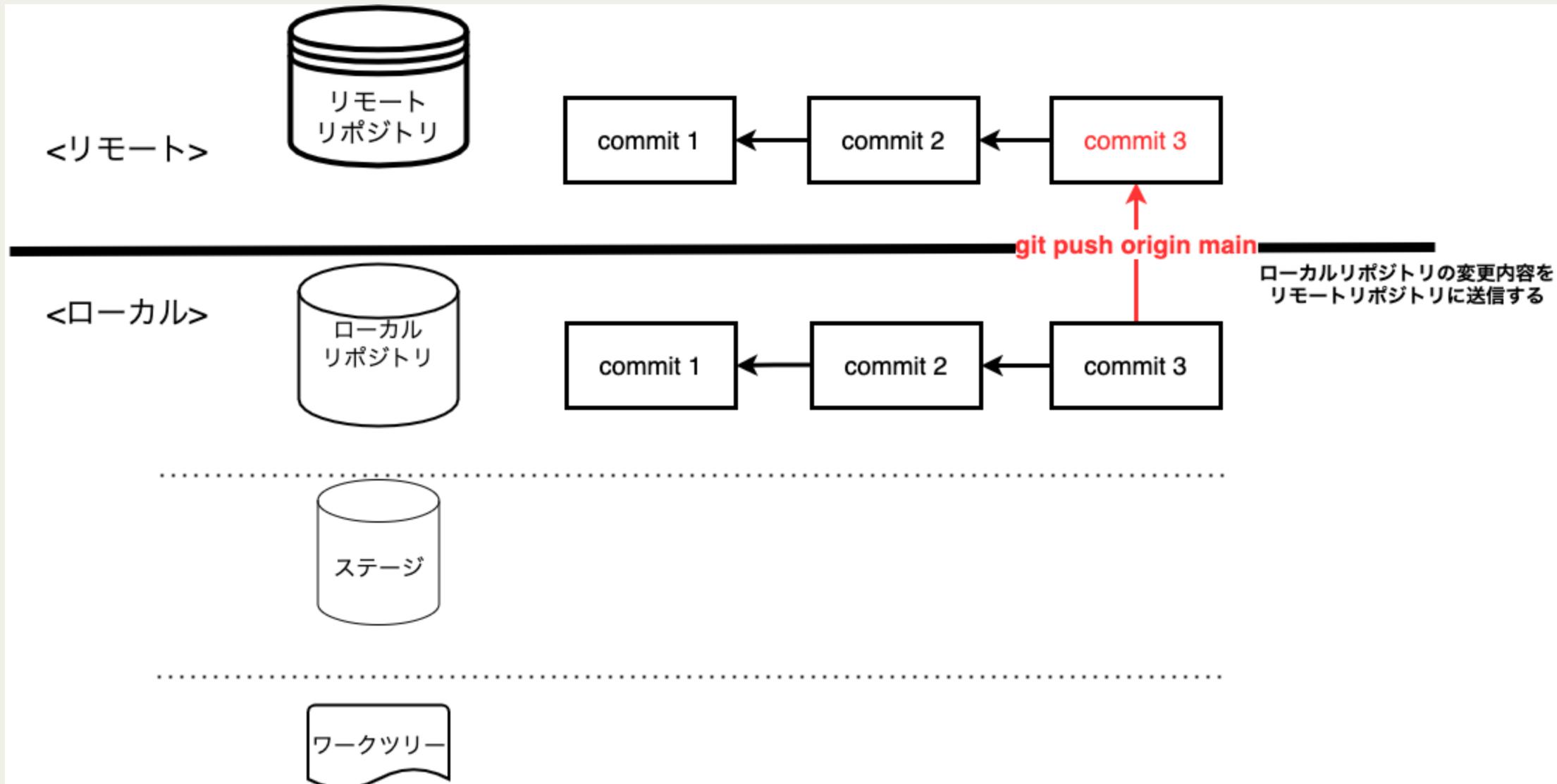
機能

- ローカルリポジトリの変更内容をリモートリポジトリに送信する

ユースケース

- ローカルリポジトリに記録した内容をリモートリポジトリに反映したい

イメージ





主なオプション

- `-u | --set-upstream` : 上流ブランチを設定する
- `-f | --force` : プッシュを強制する (できるだけ使わない!)
- `--force-with-lease` : プッシュを強制する (リモートと比較してローカルが最新のときだけ成功する)

コマンド例

```
$ git push -u origin <ブランチ名> # 上流ブランチを設定  
$ git push origin <ブランチ名> # 上流ブランチが設定されている状態なら git push でも可  
$ git push --force-with-lease origin <ブランチ名> # 強制プッシュ
```

備考

- 強制プッシュは過去のコミットを上書きする高リスクコマンド。極力使わない
- 強制プッシュが極力発生しない運用にする。設定で保護する。どうしてもプッシュしないといけない場合はチームメンバーに確認したうえで `--force-with-lease` で強制プッシュする

参考

- [git-push – Git コマンドリファレンス\(日本語版\)](#)
- [git push コマンドの使い方と、主要オプションまとめ](#)
- [Git 用語:上流ブランチとは?](#)
- [git push -f をやめて --force-with-lease を使おう - Qiita](#)

mv

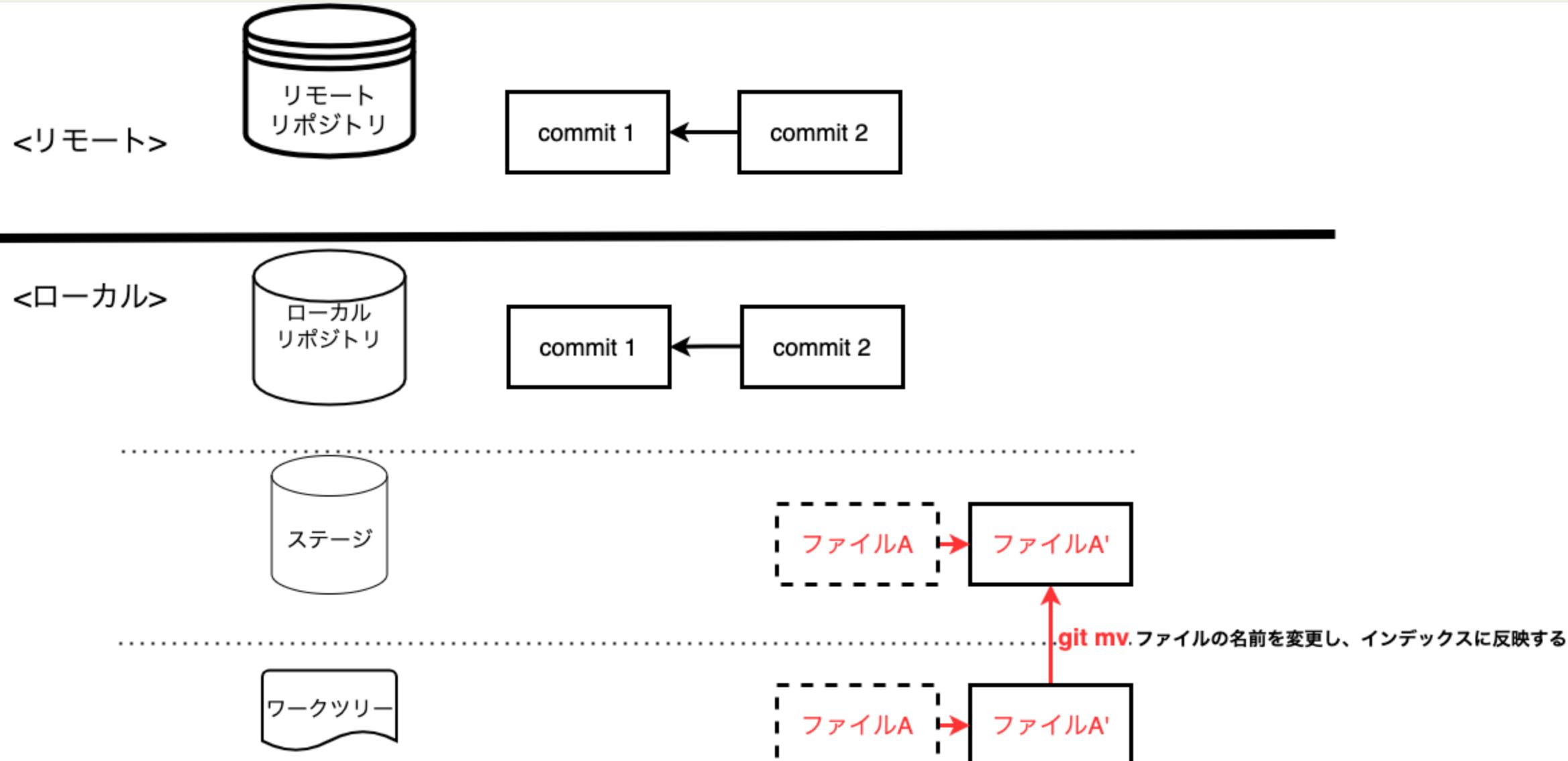
機能

- ファイルやディレクトリの名前を変更し、インデックスに反映する (move)

ユースケース

- Git 管理しているファイルの名前を変更したい

イメージ



コマンド例

```
$ git mv <変更前のファイル名> <変更後のファイル名> # ファイル名を変更し、インデックスに反映
```

備考

- mv コマンドでファイル名を変更しても、その後 git add, git rm をすれば同じ挙動になる

```
# git mv ではなく mv でファイル名変更をしてしまっても  
$ mv <変更前のファイル名> <変更後のファイル名>
```

```
# 以下のコマンドを実行すればファイル移動したことをインデックスにも反映できる（結果的に git mv と同じ操作になる）  
$ git add <変更後のファイル名>  
$ git rm <変更前のファイル名>
```

参考

- [git-mv – Git コマンドリファレンス\(日本語版\)](#)
- [Git で管理しているファイルのリネームを git mv でなく mv してしまったときにはどうなるのか調べてみた - Qiita](#)

rm

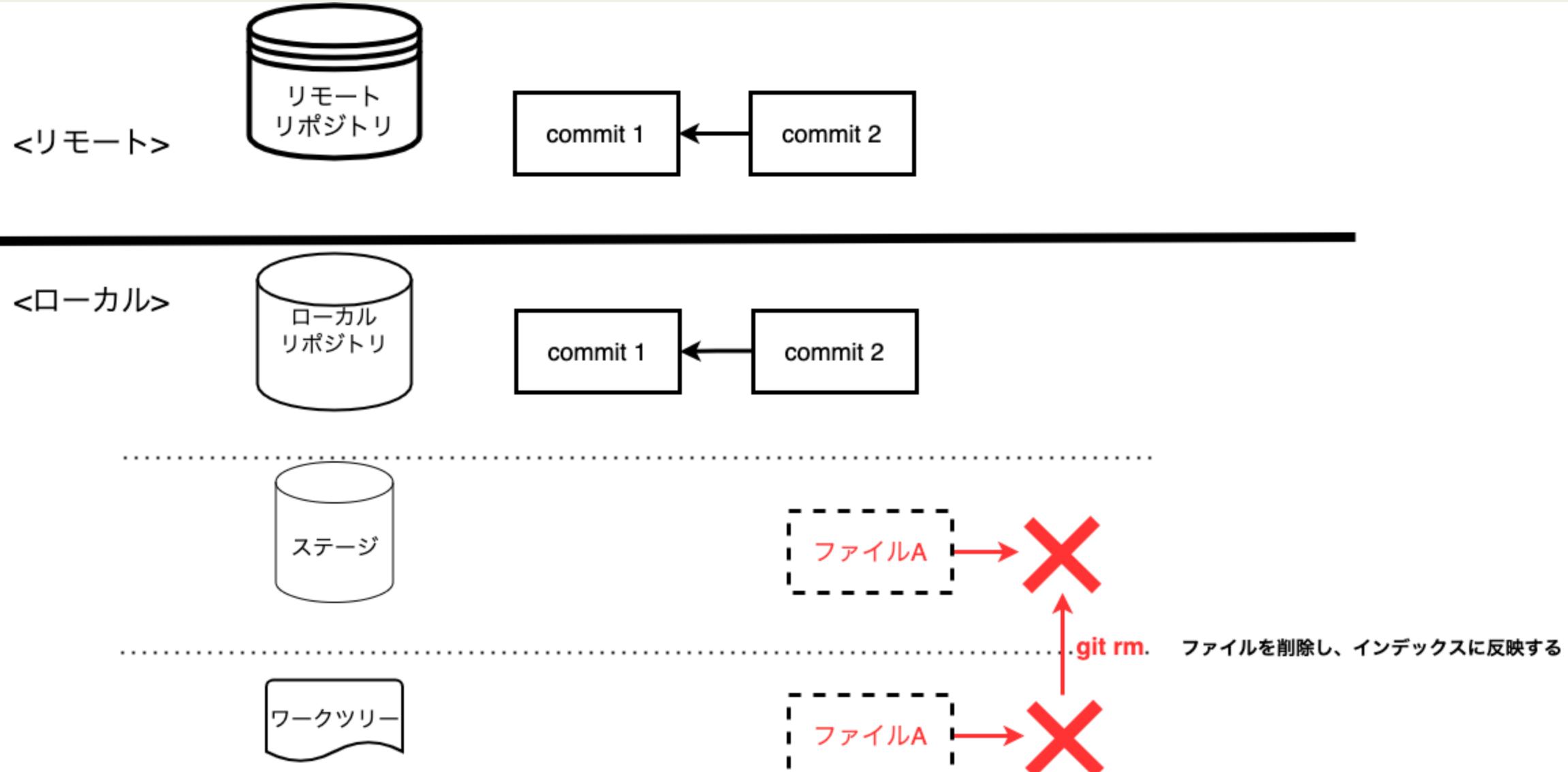
機能

- ファイルを削除し、インデックスに反映する (remove)

ユースケース

- Git 管理しているファイルを削除したい
- `.gitignore` ファイルを編集したけど反映されない

イメージ



主なオプション

- `-r` : 先頭のディレクトリが指定されている場合、ディレクトリ以下にあるファイルも再帰的に削除
- `--cached` : インデックスからのみ削除(ワークツリーにあるファイル自体は削除されず、Git 管理のみ止める)



コマンド例

```
$ git rm <ファイル名> # 指定したファイルを削除 (Git 管理も止める)  
$ git rm -r --cached . # ファイル全体キャッシュ削除
```

備考

- `rm` コマンドでファイル名を変更しても、その後 `git add` をすれば同じ挙動になる

```
# git rm ではなく rm でファイル削除をしてしまっても  
$ rm <ファイル名>
```

```
# 以下のコマンドを実行すればファイル削除したことをインデックスにも反映できる (結果的に git rm と同じ操作になる)  
$ git add <ファイル名>
```

参考

- [git-rm – Git コマンドリファレンス\(日本語版\)](#)
- [【Git 初心者入門】いちいちステージングエリアに add しなくても削除できる git rm](#)
- [.gitignore に記載したのに反映されない件 - Qiita](#)

log ★

機能

- コミット時のログを表示する

ユースケース

- コミット履歴を確認したい
 - 自分のコミットが成功したか確認したい
 - 他の人がこのブランチでどのようなコミットをしてきたのか知りたい

主なオプション

- `--oneline` : 各コミットのログを 1 行で表示
- `--no-merges` : マージコミットを除いて表示

コマンド例

```
$ git log
$ git log --oneline
# 以下は応用例（エイリアスに登録しておくと便利）
$ git log --graph --pretty=format:'%x09%C(auto) %h %Cgreen %ar %Creset%x09by"%C(cyan ul)%an%Creset" %x09%C(auto)%s %d'
```

備考

- GitHub や Visual Studio Code の拡張機能を使う方が log が見やすいのでおすすめ（以下の参考記事を参照）

参考

- [git-log – Git コマンドリファレンス（日本語版）](#)
- [git log よく使うオプションまとめ - Qiita](#)
- [git log のオプションと綺麗にツリー表示するためのエイリアス - Qiita](#)
- [GitHub 上（ブラウザ上）でコミット履歴を確認する方法](#)

diff

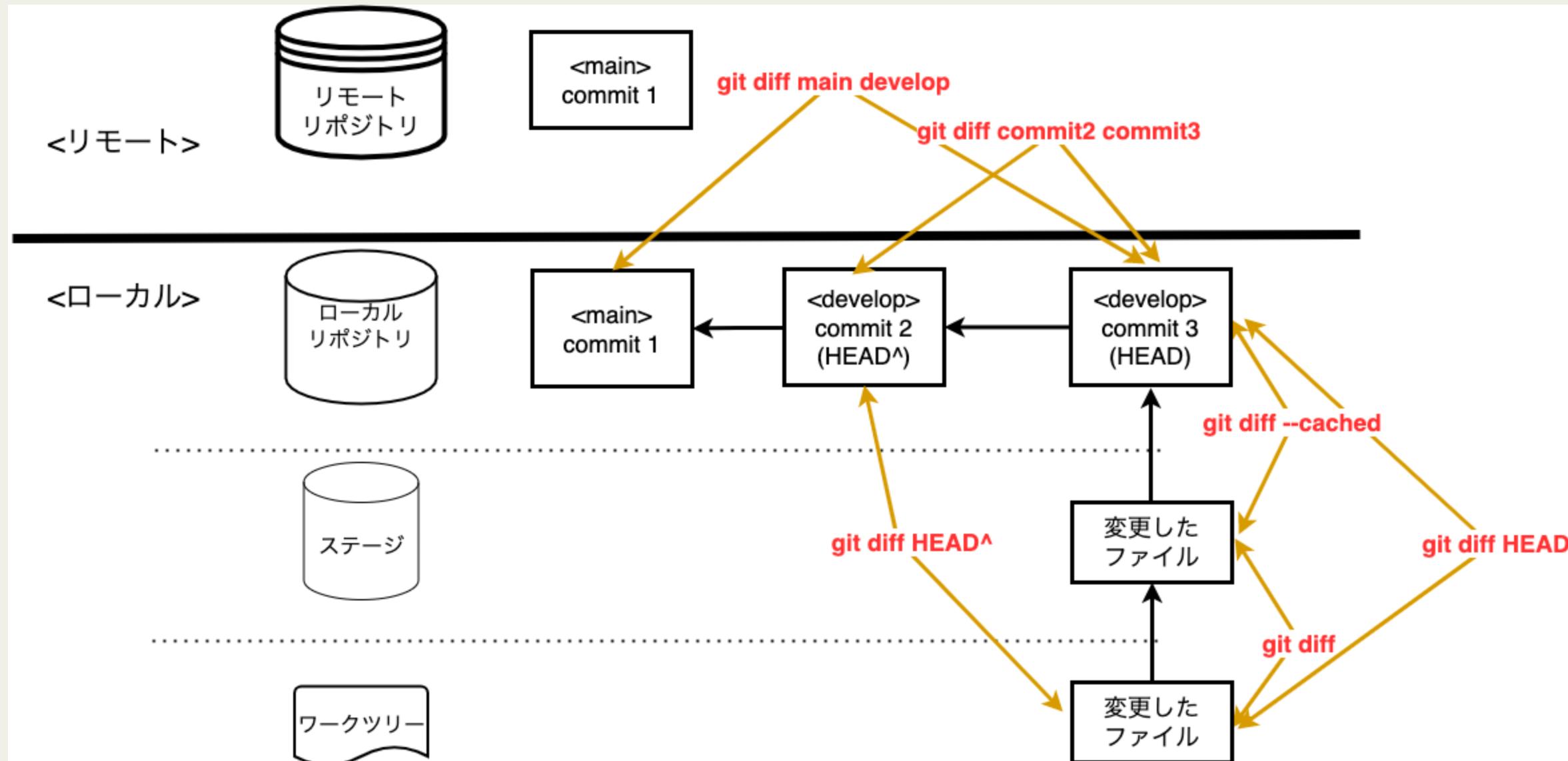
機能

- コミット同士やコミットとワークツリーの内容を比較する

ユースケース

- push する前にリモートリポジトリとの変更点を確認したい
- コミット同士を比較したい
- ブランチ間の差分を比較したい

イメージ



主なオプション

- `--cached | --staged` : インデックスとローカルリポジトリを比較



コマンド例

```
$ git diff # git add する前に変更点の比較  
$ git diff --cached # git commit する前に変更点の比較  
$ git diff HEAD^ # 直前のコミットの差分を表示  
$ git diff リモート名/ブランチ名..HEAD # git push する前にリモートとの変更点の比較  
$ git diff <変更前のコミットID>..<変更後のコミットID> # コミット同士の比較（コミットIDは git log で調べる）  
$ git diff <ブランチA>..<ブランチ名B> # ブランチ同士の比較  
$ git diff -- <ファイルパスA> <ファイルパスB> # 別ファイル同士の比較（-- の後はパスとして認識される）
```

備考

- GitHub や Visual Studio Code の拡張機能を使う方が diff が見やすいのでおすすめ(以下の参考記事を参照)

参考

- [git-diff – Git コマンドリファレンス\(日本語版\)](#)
- [忘れやすい人のための git diff チートシート](#)
- [【やっとわかった!】git の HEAD^と HEAD~の違い - Qiita](#)
- [github でブランチ・commit 間の diff を見る - Qiita](#)
- [vscode で git の異なる branch 間の差分を確認する方法 - Qiita](#)



ハンズオン

【参考】ターミナルでよく使うコマンド ①

cd

- ・ディレクトリを移動します。

ls

- ・ディレクトリの内容を表示します。ls -a コマンドで、隠しファイルを含めたディレクトリ全内容を表示します。

mkdir

- ・ディレクトリを新規作成します。

touch

- ・ファイルを作成します

rm

- ・ファイルを削除します。

【参考】ターミナルでよく使うコマンド ②

cp

- ファイルをコピーします。

mv

- ファイルの移動とファイル名の変更を行います。

cat

- ファイルの中身を表示します。

ハンズオン①:既にリモートリポジトリにあるリポジトリをコピー

リモートリポジトリをローカルに複製してみましょう
(次頁に解答例があります)

1. 自身のプロジェクト置き場(リモートリポジトリを複製する場所)へ移動
2. リモートリポジトリをローカルに複製
 - i. 事前準備で用意した個人リモートリポジトリがない場合は、以下の本勉強会用のリモートリポジトリを複製してください
 - a. `git@github.com:irongineer/git-exercise.git`
3. リモートリポジトリがローカルに複製されていることを確認
4. 複製してきたリポジトリに `.git` ディレクトリがあることを確認



ハンズオン①:既にリモートリポジトリにあるリポジトリをコピー(解答例)

```
$ cd workspace # 自身のプロジェクト置き場（リモートリポジトリを複製する場所）へ移動（※ workspace は例です）
$ git clone git@github.com:irongineer/git-exercise.git # リモートリポジトリをローカルに複製（※ git-exercise は例です）
$ ls -a # リモートリポジトリがローカルに複製されていることを確認
$ ls -a git-exercise # 複製してきたリポジトリに .git ディレクトリがあることを確認（※ git-exercise は例です）
```

ハンズオン②:Git の設定

自身のプロジェクトで Git のローカル設定を行ってみましょう
(次頁に解答例があります)

1. 自身のプロジェクトのルートディレクトリへ移動
2. ローカルリポジトリのユーザー名を設定
3. ローカルリポジトリのメールアドレスを設定
4. メインエディタを Visual Studio Code に設定。他のエディタでも OK(vim など)
5. 上記設定を確認

ハンズオン②:Git の設定(解答例)

```
$ cd git-exercise # ハンズオン用ディレクトリへ移動 (※ git-exercise は例です)
$ git config --local user.name "<メインアカウントのユーザー名>" # ローカルリポジトリのユーザー名を設定
$ git config --local user.email "<メインアカウントのメールアドレス>" # ローカルリポジトリのメールアドレスを設定
$ git config --local core.editor 'code --wait' # メインエディタを Visual Studio Code に設定。他のエディタでも OK (vim など)

# 設定の確認
$ git config user.name
$ git config user.email
$ git config core.editor
$ cat git-exercise/.git/config # local の設定ファイルを確認 (global の設定ファイルは ~/.gitconfig。※ git-exercise は例です)
```

ハンズオン③: ブランチの作成・移動・名前変更・削除

ローカルリポジトリにブランチを作成し、移動や名前変更などを行ってみましょう
(次頁に解答例があります)

なお、ハンズオン①で本勉強会用のリモートリポジトリを複製している場合は、他の人の重複防止のため、作成するすべてのブランチ名に <社員番号> を付けてください (ex: develop-1234)

1. ハンズオン用ディレクトリへ移動
2. develop ブランチを作成
3. develop ブランチに切り替え
4. temp ブランチを作成して切り替え
5. temp ブランチを temp2 ブランチに名前変更
6. ローカルとリモートリポジトリにあるブランチ一覧を確認(エディタで vim が開いた場合は :q で終了)
7. develop ブランチへ切り替え
8. temp2 ブランチを削除

ハンズオン③:ブランチの作成・移動・名前変更・削除(解答例)

```
$ cd git-exercise # ハンズオン用ディレクトリへ移動（※ git-exercise は例です）
$ git branch develop # develop ブランチを作成
$ git switch develop # develop ブランチに切り替え
$ git switch -c temp # temp ブランチを作成して切り替え
$ git branch -m temp2 # temp ブランチを temp2 ブランチに名前変更
$ git branch -a # ローカルとリモートリポジトリにあるブランチ一覧を確認（エディタで vim が開いた場合は `:q` で終了
$ git switch develop # develop ブランチへ切り替え
$ git branch -d temp2 # temp2 ブランチを削除
```

ハンズオン④: 変更をステージに追加

ファイルを変更し、ステージに追加してみましょう
(次頁に解答例があります)

1. ハンズオン用ディレクトリへ移動
2. develop ブランチへ切り替え
3. touch index.html
4. index.html ファイルを作成
5. Visual Studio Code でリポジトリを開く
6. index.html に <h1>develop での変更</h1> と追記
7. 状態を確認
 - i. Changes not staged for commit に modified: <ブランチ名>/index.html と表示される
8. ワークツリーとインデックスの差分を比較
9. ワークツリーの全ての変更ファイルをインデックスに追加
10. 状態を確認
 - i. Changes to be committed に modified: <ブランチ名>/index.html と表示される
11. インデックスとローカルリポジトリの差分を比較

ハンズオン④:変更をステージに追加(解答例)

```
$ cd git-exercise # ハンズオン用ディレクトリへ移動 (※ git-exercise は例です)  
$ git switch develop # develop ブランチへ切り替え  
$ touch index.html # index.html ファイルを作成  
$ code git-exercise # Visual Studio Code でリポジトリを開く (※ git-exercise は例です)
```

(index.html に「<h1>develop での変更</h1>」と追記)

```
$ git status # 状態を確認。Changes not staged for commit に modified: <ブランチ名>/index.html と表示される  
$ git diff # ワークツリーとインデックスの差分を比較 (エディタで vim が開いた場合は `:q` で終了)
```

```
$ git add . # ワークツリーの全ての変更ファイルをインデックスに追加  
$ git status # 状態を確認。Changes to be committed に modified: <ブランチ名>/index.html と表示される  
$ git diff --staged # インデックスとローカルリポジトリの差分を比較 (エディタで vim が開いた場合は `:q` で終了)
```

ハンズオン⑤: 変更を記録してリモートリポジトリへ送信

ハンズオン④で変更したファイルをローカルリポジトリに記録し、リモートリポジトリに送信してみましょう
(次頁に解答例があります)

0. (ハンズオン④の続きから)
1. 変更をローカルリポジトリに記録
2. 状態を確認
 - i. Your branch is ahead of 'origin/<ブランチ名>' by 1 commit. nothing to commit, working tree clean と表示される
3. 変更履歴を確認
4. 記録した変更をリモートリポジトリに送信
5. 状態を確認
 - i. Your branch is up to date with 'origin/<ブランチ名>'. nothing to commit, working tree clean と表示される

ハンズオン⑤: 変更を記録してリモートリポジトリへ送信(解答例)

```
$ git commit -m "develop を追記" # 変更をローカルリポジトリに記録
$ git status # 状態を確認。Your branch is ahead of 'origin/<ブランチ名>' by 1 commit. nothing to commit, working tree clean と表示される
$ git log # 変更履歴を確認 (エディタで vim が開いた場合は `:q` で終了)
$ git push origin develop # 記録した変更をリモートリポジトリに送信
$ git status # 状態を確認。Your branch is up to date with 'origin/<ブランチ名>'. nothing to commit, working tree clean と表示される
```

本日のゴール(再掲)

頭の中に「こんなときはこうする」というインデックスをぼんやりと作ること

- Git の各サブコマンドの存在を知ること
- 各サブコマンドのユースケースを知り、Git で躊躇したときに本資料を見返そうと思い付けること

→ この資料は辞書として使っていいってほしいので、完全に理解しようとしないで OK です！

次回

次回は **基本コマンド & ハンズオン ② ~チーム開発編~** です!
おたのしみに!