



# Git 勉強会 2 日目

～基本コマンド & ハンズオン ①～  
2022/03/25

# 2日目アジェンダ(今日はこっち)

- Git 基本コマンド ① ~個人開発編~
  - init
  - clone
  - switch (checkout)
  - remote
  - branch
  - status
  - add
  - commit
  - push
  - log
  - diff
  - config

# 3日目アジェンダ

- Git 基本コマンド ② ~チーム開発編~
  - fetch
  - merge
  - rebase
  - pull
  - stash
  - restore (checkout)
  - reset
  - revert
  - cherry-pick
  - blame
  - tag
  - reflog



# 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

## コマンド例

```
$ git init
```



イメージ

参考

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

# clone

## 機能

- リポジトリのクローンを作成する

## 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

## 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>
$ git clone <リモートリポジトリの URL> -b <ブランチ>
```



イメージ

参考

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

# switch (checkout) # TODO: branch の前後に移動する?



## 機能

- 作業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

## 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> # git checkout <ブランチ> と同じ  
$ git switch -c <ブランチ> # 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、こちらの方が理解しやすい



## イメージ

## 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](#)

# remote

## 機能

- リモートリポジトリを関連付けする

## 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

## 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>
$ git remote -v
$ git remote remove <リモートリポジトリのURL>
```



## イメージ

## 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

// TODO: 主なオプション  
// TODO: -a

## コマンド例

```
$ git branch
$ git branch -m <変更前のブランチ名> <変更後のブランチ名> # 今いるブランチ上で git branch -m <変更後のブランチ名> でも OK
$ git branch -d <ブランチ名> # git branch -D <ブランチ名> で強制削除
```



## イメージ

## 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](#)

# 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
```



## イメージ

## 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

- 変更したファイルの一部をコミットしたい // TODO: もっといい表現がありそう
- 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

## 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 -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)
```

## イメージ

## 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？](#)

## 機能

- インデックスに追加した変更を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

## 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

## イメージ

## 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

## 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

## 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 (**非推奨**)
- `--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pushが発生しない運用における設定と保護する方法について詳しくは「[こちら](#)」を参考

## イメージ

## 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](#)

## 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

- `--online` : 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 --online
# 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'
```

## イメージ

## 参考

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

# diff

## 機能

- ・コミット同士やコミットと作業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

## コマンド例

```
$ git diff # git add 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 (ハッシュは git log で表示される commit の右にある文字列)
$ 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)
```



## イメージ

## 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](#)

## 機能

- 現在の設定を取得、変更する

## ユースケース

- 複数の git アカウントを使い分けたい

## コマンド例

```
$ git config # 現在いるリポジトリの Git 設定を表示
// TODO: Windows の場合の設定ファイルはどこ?
$ 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 ~/workspace/<リポジトリのルートディレクトリ>
$ 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

- `--global` の設定ファイルは `~/.gitconfig` にある
- `--local` の設定ファイルは `<リポジトリのルートディレクトリ>/ .git/config` にある



## イメージ

## 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](#)



# ハンズオン

// TODO: コンテンツ作成