



Git 勉強会 3 日目

～基本コマンド & ハンズオン ②～
2022/03/28

2日目アジェンダ

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

3日目アジェンダ(今日はこっち)

- 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

fetch

機能

- リモートリポジトリの内容を取得する
 - 取得した内容はローカルのリモート追跡ブランチに反映するが、ワーキングツリーには反映しない

ユースケース

- 他者が作成したブランチに切り替えたいので、リモートリポジトリの最新状態を取得したい
- pull だとワーキングツリーまで更新してしまうので、とりあえずリモートの状態を確認したい

主なオプション

- all : すべてのリモートリポジトリの内容を取得
 - リモートリポジトリが origin しかない場合は使うことはない

コマンド例

```
$ git fetch # リモートリポジトリの内容をローカルのリモート追跡ブランチに反映
```



イメージ

参考

- [git-fetch – Git コマンドリファレンス\(日本語版\)](#)
- [git pull と git pull –rebase の違いって?図を交えて説明します!](#)

機能

- 他のブランチやコミットの内容を現在のブランチに取り込む

ユースケース

- fetch した内容をワーキングツリーに反映したい(ブランチをマージした記録をログに残したい)
- マージ時にコンフリクトしたので、競合を解決したい

主なオプション

- no-ff : fast-forward であっても必ずマージコミットを作成
- continue : コンフリクト解決後、マージを続行
- abort : コンフリクト解決を中止し、マージ前の状態に再構築

コマンド例

```
$ git switch <マージ先のブランチ名>
$ git merge <マージ元のブランチ名>
```

イメージ

参考

- [git-merge – Git コマンドリファレンス\(日本語版\)](#)
- [git pull と git pull –rebase の違いって?図を交えて説明します!](#)

機能

- コミットを再適用する(ブランチの分岐点を変更したり、コミットの順番を入れ替えたりできる)

ユースケース

- fetchした内容をワーキングツリーに反映したい(コミットログを一直線に保ちたい)
- リベース時にコンフリクトしたので、競合を解決したい
- 複数のコミットをひとつにまとめたい(2つ以上前のコミットを修正したい)

主なオプション

- continue : コンフリクト解決後、リベースを続行
- abort : コンフリクト解決を中止し、リベース前の状態に再構築
- i | --interactive : 複数のコミットを統合

コマンド例

```
$ git switch <マージ先のブランチ名>
$ git rebase <マージ元のブランチ名>
$ git rebase -i HEAD~4 # 最新から 4つ分のコミットを修正・統合
```



イメージ

参考

- [git-rebase – Git コマンドリファレンス\(日本語版\)](#)
- [git pull と git pull –rebase の違いって?図を交えて説明します!](#)
- [git rebase を初めて使った際のまとめ - Qiita](#)
- [rebase -i でコミットをまとめる - Qiita](#)
- [【やっとわかった!】git の HEAD^ と HEAD~ の違い - Qiita](#)
- [git rebase 失敗した時の対処法 - Qiita](#)

pull

機能

- リモートリポジトリの内容を取得し、現在のブランチに取り込む(`git fetch` + `git merge`)

ユースケース

- リモートリポジトリの最新情報をローカルリポジトリに取り込みたい

主なオプション

- `-r` | `--rebase` : フェッチ後に現在のブランチを上流ブランチの上にリベース
 - `git fetch` + `git rebase`

コマンド例

```
$ git switch <ブランチ名> # まず pull したいブランチへ切り替える
$ git pull origin <ブランチ名> # リモートリポジトリの内容を取得してマージ。上流ブランチを設定している場合は git pull で OK
$ git pull -r origin <ブランチ名> # リモートリポジトリの内容を取得してリベース。
```

イメージ

参考

- [git-pull – Git コマンドリファレンス\(日本語版\)](#)
- [git pull コマンドの使い方と、主要オプションまとめ](#)
- [【初心者向け】git fetch、git merge、git pull の違いについて - Qiita](#)
- [git pull と git pull –rebase の違いって?図を交えて説明します!](#)
- [Git のコミットメッセージを後から変更する方法をわかりやすく書いてみた](#)

stash

機能

- 未コミットの変更を退避する

ユースケース

- 他ブランチに切り替えたいが、作業が中途半端なのでコミットはしたくない

主なオプション

- `-u | --include-untracked` : 追跡対象に含まれていないファイル(新規作成ファイルなど)も含めて退避

コマンド例

```
$ git stash -u # 新規作成ファイルも含めて変更を退避
$ git stash -u push "<メッセージ>" # メッセージを付けて変更を退避
$ git stash list # 退避した作業の一覧を見る
$ git stash apply stash@{0} # 退避した作業を戻す。stash@{N} を省略した場合は直前に stash した情報を戻す
$ git stash show stash@{0} -p # 退避した変更の詳細を見る
```

イメージ

参考

- [git-stash Documentation](#)
- [【git stash】コミットはせずに変更を退避したいとき - Qiita](#)

機能

- ファイルを指定した状態に復元する

ユースケース

- add をしたけど取り消したい
- 特定のコミットの時点に戻したい

主なオプション

- `-S | --staged` : インデックスを復元
- `-s | --source` : 指定したコミットの状態にワークツリーファイルを復元

コマンド例

// TODO: checkout の例を追記

```
git restore <ファイル名> # 特定のファイルを保存前に戻す // TODO: 動作確認
git restore --staged . # 対象ファイル全体を add される前に戻す。git reset . 同じ挙動 // TODO: 動作確認
git restore --source <コミット識別子> <ファイル名> # 特定のファイルを、特定のコミット時点に戻す
```

備考



イメージ

参考

- [git-restore – Git コマンドリファレンス\(日本語版\)](#)
- [git switch と restore の役割と機能について - Qiita](#)
- [これからは git restore を使ってみようかな](#)

機能

- ファイルをインデックスから削除し、特定のコミットの状態まで戻す

ユースケース

- 直前のコミットを取り消したい (push する前)
- 目的のブランチに切り替える前に誤って pull してしまったのを取り消したい

主なオプション

- `--soft` : インデックスとワークツリーはそのままで、指定したコミットヘリセット (commit のみ取り消し)
- `--mixed` : ワークツリーはそのまま、インデックスを指定したコミットヘリセット (commit と add を取り消し。デフォルト)
- `--hard` : インデックスとワークツリーを指定したコミットヘリセット (現在のファイル変更も含めて全部取り消し)

コマンド例

```
$ git reset --soft HEAD^ # コミットのみ取り消す。現在のファイル変更はそのまま。HEAD^ は直前のコミットを意味する
$ git reset --hard HEAD^ # commit, add, 現在のファイル変更も全部取り消す
$ git reset --hard 昔のコミットのハッシュ値 # 指定したコミットの状態に戻す
$ git reset --hard <ブランチ> # 指定したブランチの状態に戻す（現在いるブランチに戻す場合は下と同じ）
$ git reset --hard ORIG_HEAD # 直前の reset を取り消す（最新の状態に戻る）
```



イメージ

参考

- [git-reset – Git コマンドリファレンス\(日本語版\)](#)
- [\[git reset \(--hard/--soft\)\]ワーキングツリー、インデックス、HEAD を使いこなす方法](#)
- [第 6 話 git reset 3 種類をどこよりもわかりやすい図解で解説!【連載】マンガでわかる Git ~コマンド編~](#)

機能

- ・ 指定したコミットの内容を打ち消すコミットを生成し、現在いるブランチに適用する

ユースケース

- ・ 直前のコミットを取り消したい (push した後)

主なオプション

- ・ `-e` | `--edit` : コミットメッセージを編集する (デフォルト)
- ・ `--no-edit` : コミットメッセージを編集しない
- ・ `-n` | `--no-commit` : 打ち消しコミットを生成するが、適用しない

コマンド例

```
$ git revert HEAD^ # 直前のコミットを打ち消すコミットを適用
$ git revert -n <コミットID> # 特定のコミットを打ち消すコミットを生成（適用するときは git commit）
$ git revert HEAD~{N} # 直前から N 個前の範囲で複数コミットを打ち消すコミットを適用
$ git revert <コミットID A>..<コミットID B> # A から B の範囲で複数コミットを打ち消すコミットを適用
```

イメージ

参考

- [git-revert Documentation](#)
- [【git コマンド】いまさらの revert - Qiita](#)
- [Git revert と reset について - Qiita](#)

cherry-pick

機能

- ・ 指定したコミットの内容を現在いるブランチに取り込む

ユースケース

- ・ 他のブランチの特定のコミットを現在のブランチに取り込みたい(複数 PR を並行してレビューしているときなど)

主なオプション

- ・ `-e | --edit` : コミットメッセージを編集する (**デフォルト**)
- ・ `--no-edit` : コミットメッセージを編集しない
- ・ `-n | --no-commit` : 指定したコミットを取得するが、適用しない

コマンド例

```
$ git cherry-pick <コミットID> # 特定のコミットを現在のブランチに取り込み、適用  
$ git cherry-pick -n <コミットID> # 特定のコミットを現在のブランチに取り込む（適用するときは git commit）  
$ git cherry-pick <コミットID A>..<コミットID B> # A から B の範囲で複数コミットを現在のブランチに取り込み、適用
```

イメージ

参考

- [git-cherry-pick Documentation](#)
- [git cherry-pick を完全マスター!特定コミットのみを取り込む方法](#)
- [git で他ブランチの特定のコミットを取り込む方法 - Qiita](#)

機能

- 指定されたファイルの各行に最終更新者・日時などの情報を表示する

ユースケース

- この行は誰が変更したのか調べたい

主なオプション

- `-L` : 最終コミットを表示する行の範囲を指定

コマンド例

```
$ git blame <ファイル名> # 特定のファイルについて各行ごとに最終コミット情報を表示する
$ git blame -L 40,50 <ファイル名> # 40～50行目の最終コミット情報を表示する
$ git blame -L 40,+10 <ファイル名> # 40行目から10行分の最終コミット情報を表示する
```

備考

- 最近のエディタでは拡張機能を入れると blame の結果が表示される(Visual Studio Code では [GitLens](#)) 22



イメージ

参考

- [git-blame Documentation](#)
- [インデントコミットで真犯人がわからなくなったりした場合の git blame - Qiita](#)
- [GitLens — Git supercharged - Visual Studio Marketplace](#)

機能

- コミットにタグを付ける、削除する、一覧表示する

ユースケース

- リリースした時点のソースコードに名前を付けたい

主なオプション

- `-a | --annotate` : 注釈付きのタグを作成
- `-m | --message` : メッセージを付与
- `-l | --list` : タグを一覧表示

コマンド例

```
$ git tag <タグ名> # コメント（注釈）なしでタグを作成する
$ git tag -a <タグ名> -m '<タグのコメント>' # コメント（注釈）付きでタグを作成
$ git tag -a タグ -m 'コメント' <コミットID> # 過去のコミットにタグを作成
$ git push origin <タグ名> # タグをリモートリポジトリに反映する
$ git tag -l # タグを一覧表示する
```

イメージ

参考

- [git-tag – Git コマンドリファレンス\(日本語版\)](#)
- [git tag の使い方まとめ - Qiita](#)

reflog

機能

- ・ ブランチのヒントやその他の参照がローカルリポジトリで更新された時期を記録する

ユースケース

- ・ コミット履歴だけでなく HEAD やブランチの動きなど細かいところまで確認したい
- ・ 間違えて `git reset --hard` をしてしまったので戻したい

主なオプション

- ・ `--date=default` : 操作日時を絶対時刻で表示

コマンド例

```
$ git reflog # コミット履歴や HEAD・ブランチの参照に関する変化ログを表示
$ git reflog --date=default # 日時付きで表示
$ git reset HEAD@{1} --hard # reflog で元に戻したいコミットを指定して、reset --hard で戻す
```

イメージ

参考

- [git-reflog Documentation](#)
- [いざという時のための git reflog - Qiita](#)
- [初心者から一歩抜け出すための Git の業 ~ git reflog - Qiita](#)
- [git reflog を日時で参照する](#)



ハンズオン

// TODO: コンテンツ作成