Git付録

用語・コマンド集

ctirus88
citrus.mikan88@gmail.com

基本概念

Git（ギット）

分散型バージョン管理システムの名前です。ファイルの変更履歴を記録し、複数人での開発を支援するツールです。Linuxの生みの親であるリーナス・トーバルズによって開発されました。

関連コマンド:

```
 # Gitのバージョン確認
git --version

# Gitのヘルプ表示
git help
git help コマンド名
```

バージョン管理システム（Version Control System）

ファイルやプロジェクト全体の変更履歴を記録し、管理するシステムです。「いつ」「誰が」「何を」変更したかを追跡できます。

分散型バージョン管理

各開発者が完全なリポジトリのコピーを持つ方式です。中央サーバーに依存せずに作業でき、オフラインでも履歴管理が可能です。

初期設定

基本設定コマンド:

```
 # ユーザー名の設定
git config --global user.name "あなたの名前"

# メールアドレスの設定
git config --global user.email "あなたのメールアドレス"

# デフォルトエディタの設定（VS Code）
git config --global core.editor "code --wait"

# 設定の確認
git config --list
git config user.name

# エイリアス設定
git config --global alias.st status
git config --global alias.co checkout
git config --global alias.br branch
git config --global alias.ci commit
git config --global alias.unstage 'reset HEAD --'
```

リポジトリ関連

リポジトリ（Repository）

プロジェクトのファイルとその変更履歴を保存する場所です。「.git」フォルダ内にすべての情報が格納されます。「レポ」と略して呼ばれることもあります。

関連コマンド:

```
 # 現在のディレクトリをGitリポジトリとして初期化
git init

# 指定した名前のディレクトリを作成してGitリポジトリとして初期化
git init プロジェクト名
```

ローカルリポジトリ（Local Repository）

自分のコンピュータ上にあるリポジトリです。他の人に影響を与えずに自由に変更や実験ができます。

リモートリポジトリ（Remote Repository）

ネットワーク上（GitHubやGitLabなど）にある共有のリポジトリです。チームメンバーがコードを共有するために使用します。

関連コマンド:

```
 # リモートリポジトリの一覧を表示
git remote -v

# リモートリポジトリを追加
git remote add origin https://github.com/ユーザー名/リポジトリ名.git

# リモートリポジトリのURLを変更
git remote set-url origin https://github.com/ユーザー名/リポジトリ名.git

# リモートリポジトリを削除
git remote remove origin
```

clone（クローン）

リモートリポジトリを自分のローカル環境にコピーすることです。初回のダウンロードに使用します。

関連コマンド:

作業領域と状態

ワーキングディレクトリ（Working Directory）

実際にファイルの編集作業を行うフォルダです。ここでコードを書いたり、ファイルを作成・削除します。

ステージングエリア（Staging Area）

次のコミットに含めたい変更を一時的に準備する場所です。「インデックス」とも呼ばれます。

インデックス（Index）

ステージングエリアの別名です。addコマンドで変更をここに登録します。

関連コマンド:

```
 # 現在の状態を確認
git status
git status -s  # 簡潔な表示

# ファイルをステージングエリアに追加
git add ファイル名
git add .  # すべての変更ファイルを追加
git add -A  # すべての変更とファイル削除を追加
git add -p  # 対話的に追加

# ステージング済みの変更を取り消し
git reset HEAD ファイル名
git restore --staged ファイル名  # 新しいコマンド
```

tracked（トラック済み）

Gitが変更を追跡しているファイルの状態です。過去にaddまたはcommitされたファイルを指します。

untracked（未トラック）

Gitがまだ変更を追跡していないファイルの状態です。新しく作成されたファイルがこの状態になります。

staged（ステージ済み）

addコマンドでステージングエリアに追加された変更の状態です。次のコミットに含まれる予定の変更を指します。

変更の記録

コミット（Commit）

特定の時点でのプロジェクトの状態を記録することです。スナップショットのように、その瞬間のファイルの状態を保存します。

関連コマンド:

```
 # 変更をコミット
git commit -m "コミットメッセージ"

# ステージングとコミットを同時実行（新規ファイルは除く）
git commit -am "コミットメッセージ"

# 直前のコミットメッセージを修正
git commit --amend -m "新しいコミットメッセージ"
```

コミットハッシュ（Commit Hash）

各コミットに付けられる一意の識別子です。SHA-1ハッシュという40文字の英数字で表現されます。

コミットメッセージ（Commit Message）

コミット時に記述する変更内容の説明文です。「何を変更したか」を分かりやすく記述します。

HEAD（ヘッド）

現在作業中のブランチの最新コミットを指すポインタです。「今いる場所」を示します。

履歴確認コマンド:

```
 # コミット履歴を表示
git log
git log --oneline  # 1行で簡潔に表示
git log --graph --oneline --all  # グラフ形式で表示
git log -n 5  # 最新のn件のコミットを表示
git log --author="作者名"  # 特定の作者のコミット
git log --since="2024-01-01" --until="2024-12-31"  # 特定期間
git log -p ファイル名  # ファイルの変更履歴を表示
```

ブランチとマージ

ブランチ（Branch）

開発の流れを分岐させる機能です。新機能の開発や実験を、メインの開発ラインに影響を与えずに行えます。

関連コマンド:

```
 # ブランチ一覧を表示
git branch
git branch -a  # リモートブランチも含めて表示

# 新しいブランチを作成
git branch 新ブランチ名

# ブランチを作成して同時に切り替え
git checkout -b 新ブランチ名
git switch -c 新ブランチ名  # 新しいコマンド（推奨）

# 既存ブランチに切り替え
git checkout ブランチ名
git switch ブランチ名  # 新しいコマンド（推奨）

# 直前のブランチに戻る
git checkout -

# ブランチを削除
git branch -d ブランチ名
git branch -D ブランチ名  # 強制削除

# ブランチ名を変更
git branch -m 古いブランチ名 新しいブランチ名
git branch -m 新しいブランチ名  # 現在のブランチ名を変更
```

main/master（メイン/マスター）

プロジェクトのメインとなるブランチの名前です。最近は「main」が使われることが多くなっています。

feature branch（フィーチャーブランチ）

新機能開発のために作成する専用のブランチです。機能完成後にmainブランチにマージします。

競合と解決

競合（Conflict）

同じファイルの同じ箇所を異なる内容で変更した際に発生する問題です。Gitが自動的にマージできない状況を指します。

競合マーカー（Conflict Marker）

競合が発生した際にファイルに挿入される特殊な記号です。`<<<<<<<`、`=======`、`>>>>>>>`で競合箇所を示します。

競合解決（Conflict Resolution）

競合が発生した際に、どちらの変更を採用するかを手動で決定する作業です。

リモート操作

push（プッシュ）

ローカルでコミットした変更をリモートリポジトリに送信することです。自分の変更を共有する際に使用します。

関連コマンド:

```
 # リモートに変更を送信
git push origin ブランチ名

# 初回プッシュ時（上流ブランチの設定）
git push -u origin ブランチ名

# 設定済みの上流ブランチにプッシュ
git push
```

pull（プル）

リモートリポジトリの最新変更をローカルに取得し、現在のブランチに統合することです。

関連コマンド:

```
 # リモートの変更を取得してマージ
git pull origin ブランチ名

# 設定済みの上流ブランチからプル
git pull
```

fetch（フェッチ）

リモートリポジトリの最新情報を取得しますが、現在のブランチには統合しません。内容を確認してからマージできます。

関連コマンド:

```
 # リモートの情報のみ取得（マージしない）
git fetch origin

# すべてのリモートブランチの情報を取得
git fetch --all
```

GitHub/GitLab関連

プルリクエスト（Pull Request）

GitHubで使用される用語で、自分の変更をメインブランチに統合してもらうための依頼です。コードレビューの機能も含みます。

マージリクエスト（Merge Request）

GitLabでプルリクエストと同じ機能を指す用語です。

fork（フォーク）

他人のリポジトリを自分のアカウントにコピーすることです。オープンソースプロジェクトへの貢献でよく使用されます。

変更の取り消し

ワーキングディレクトリの変更を取り消し:

```
 # 特定ファイルの変更を取り消し
git checkout -- ファイル名
git restore ファイル名  # 新しいコマンド（推奨）

# すべてのファイルの変更を取り消し
git checkout -- .
git restore .  # 新しいコマンド（推奨）
```

コミットの取り消し:

```
 # 直前のコミットを取り消し（変更は残す）
git reset --soft HEAD~1

# 直前のコミットを取り消し（ステージングも取り消し）
git reset HEAD~1

# 直前のコミットを完全に取り消し（変更も破棄）
git reset --hard HEAD~1

# 特定のコミットまで戻る
git reset --hard コミットハッシュ
```

高度な概念

stash（スタッシュ）

現在の変更を一時的に保存する機能です。急いで別のブランチに切り替える必要がある際に使用します。

関連コマンド:

```
 # 現在の変更を一時保存
git stash

# メッセージ付きで保存
git stash save "作業内容の説明"

# stashの一覧を表示
git stash list

# 最新のstashを復元
git stash pop

# 特定のstashを復元
git stash apply stash@{n}

# stashを削除
git stash drop stash@{n}

# すべてのstashを削除
git stash clear
```

rebase（リベース）

コミット履歴を整理し、直線的な履歴を作成する機能です。ブランチのベース（基準点）を変更します。

関連コマンド:

```
 # 現在のブランチを指定ブランチの最新に基づいて再構築
git rebase main

# インタラクティブrebase（過去nコミットを編集）
git rebase -i HEAD~n

# rebaseの中断
```

その他の便利なコマンド

```
 # ファイルの各行の変更履歴を表示
git blame ファイル名

# 特定の文字列を含むコミットを検索
git log --grep="検索文字列"

# ファイル内容の検索
git grep "検索文字列"

# ファイル名の変更を追跡
git mv 古いファイル名 新しいファイル名

# 未追跡ファイルを削除
git clean -f
git clean -fd  # 未追跡ディレクトリも含めて削除
git clean -n  # 削除実行前のプレビュー

# リモートで削除されたブランチをローカルからも削除
git remote prune origin
```

トラブルシューティング設定

```
 # 改行コード設定（Windows）
git config --global core.autocrlf true

# 改行コード設定（Mac/Linux）
git config --global core.autocrlf input

# ファイル権限の変更を無視
git config core.filemode false

# 大文字小文字の変更（macOS/Windows）
git mv ファイル名 一時的な名前
git mv 一時的な名前 正しいファイル名
```