SnapTrack設計書

**SnapTrack**は、軽量で効率的なバージョン管理システムを目指して設計されています。このシステムは、開発者がソースコードやファイルの変更を記録し、過去の状態を再現可能にします。また、リモートリポジトリを利用してチームでの共同作業も可能です。

# SnapTrackの基本概念

1. Repository／Repo（リポジトリ／リポ）

**リポジトリ**とは、ファイルとそのバージョン履歴を管理するための保存場所です。プロジェクトの変更履歴を追跡し、その進化を管理するために使用されます。SnapTrackでは、リポジトリは.snaptrackフォルダです。

1. Working Directory（作業ディレクトリ）

**作業ディレクトリ**とは、.snaptrackフォルダが格納されているディレクトリを指します。このディレクトリは、SnapTrackがバージョン管理を行うプロジェクトの**ルートディレクトリ**として機能します。ここには、ユーザーが操作する実際のプロジェクトファイルが含まれており、SnapTrackはこれらのファイルの変更を監視してバージョン管理を行います。

1. Staging（ステージング）

**ファイルをステージングする**とは、そのファイルの変更を次のコミットに含める準備ができた状態としてマークすることを意味します。ユーザーが作業ディレクトリでファイルを編集、作成、または削除した場合、それらの変更は最初はリポジトリによって未追跡、または変更済みとして扱われます。ステージングすることで、ユーザーは変更されたファイルのバージョンを明示的にステージングエリアに追加します。ステージングエリアは、一時的なスペースであり、変更をリポジトリにコミットして永続的に保存する前に準備を整える場所です。SnapTrackでは、このステージングエリアはindexファイルによって表現されており、ステージングされた変更を管理します。indexファイルの説明は[こちら](#indexfile)。

1. Commit（コミット）

コミットとは、ステージングエリアの変更を取得し、それをリポジトリに保存して、プロジェクトの履歴に永続的な記録を作成するプロセスを指します。各コミットは、特定の瞬間における追跡されたファイルの状態を記録し、作成者、タイムスタンプ、変更内容を説明するメッセージなどのメタデータを含みます。コミットはリポジトリのバージョン履歴の構成要素として機能し、プロジェクトの進化を追跡する手段を提供します。また、特定の変更を確認したり、元に戻したり、そこからブランチを作成したりするための参照ポイントとしても利用できます。

1. Branch（ブランチ）

リポジトリにおける**ブランチ**は、並行した開発のラインを指します。ブランチを使用すると、ユーザーはメインプロジェクトや他のブランチに影響を与えることなく、独立した変更作業を行うことができます。ブランチは特定のコミットを指すポインタとして開始し、新しいコミットが追加されるとそのブランチは前進し、独自のタイムラインを維持します。これにより、ブランチは新機能の開発、バグ修正、アイデアの試行など、異なる作業フローを整理し管理するための重要なツールとなります。また、ブランチは複数のユーザーが同じプロジェクトの異なる部分で同時に作業できるようにすることで、コラボレーションや独立した開発を可能にします。

SnapTrack における**作業ブランチ**とは、ユーザーが現在アクティブに変更を加えているブランチを指します。これは、バージョン管理システム内でのユーザーの現在のコンテキストを表しており、別のブランチに切り替えるまでは、すべてのコミットや更新がこのブランチに適用されます。

1. Merging（マージ）

**マージ**とは、あるブランチの変更を別のブランチに統合するプロセスです。これにより、別のブランチで開発された更新、機能、または修正を対象のブランチに取り込むことができます。マージが実行されると、SnapTrackはブランチ間の変更を比較し、それらの差分を対象のブランチに適用します。マージは共同作業のワークフローにおいて重要であり、並行するブランチで行われた作業を一つの統一されたプロジェクトにまとめることができます。また、バージョン履歴の整合性を維持しながら、複数の開発ラインをスムーズに統合することを可能にします。「マージコンフリクト」と呼ばれる概念があり、それについては後ほど説明します。

1. Hash（ハッシュ）

SnapTrackでは、**ハッシュ**とは任意のサイズの文字列（またはバッファ）を一意に表す20バイトの数値です。SnapTrackは、[SHA-1ライブラリ](https://github.com/josufh/SHA1)を使用して実装されたSHA-1アルゴリズムを使用してこれらのハッシュを生成します。この文脈で、バッファは通常、ファイルの内容を指します。ハッシュは、2つのファイルの内容を効率的に比較し、それらが同一の内容を持つかどうかを判断するために使用されます。

1. Object（オブジェクト）

オブジェクトは、SnapTrackがバージョン管理のために情報を保存する際に使用するファイルの一種です。オブジェクトファイルにはさまざまな種類がありますが、すべてに共通する特徴があります。それは、ファイル名がその内容のハッシュ値になっていることです。この仕組みにより、他のファイルから簡単に参照できるようになっています。オブジェクトファイルについては、それぞれの具体的な用途が説明される際に[詳しく解説](#_各オブジェクトファイル形式説明)します。

1. Remote（リモート）

リモートリポジトリとは、オフィスネットワーク内のリモートサーバー上にホストされているバージョン管理されたストレージ場所です。プロジェクトのコードベースを集中管理する役割を果たし、同じネットワークに接続された任意のコンピューターからアクセスして共同作業を行うことができます。

1. パス

フォーマット標準化のために、SnapTrack のパスではパス区切り文字としてバックスラッシュ（\）が使用されます。ディレクトリ名の最後にバックスラッシュ（￥）を入力する必要があります。

# ローカルリポジトリのフォルダ・ファイルの説明

|  |  |  |
| --- | --- | --- |
| **№** | **パス** | **記述名** |
| 1 | .snaptrack\ | snaptrack |
| 2 | .snaptrack\index | index |
| 3 | .snaptrack\HEAD | HEAD |
| 4 | .snaptrack\refs\branches\ | branches |
| 5 | .snaptrack\refs\remotes\ | remotes |
| 6 | .snaptrack\objects\ | objects |
| 7 | .snaptrackignore | ignore |
| 8 | (外)AppData\Roaming\SnapTrack\config | config |

表１　ローカルリポジトリフォルダ・ファイル一覧

1. snaptrack

SnapTrackの[リポジトリ](#repository)です。

1. index

インデックスファイルは、作業ディレクトリとリポジトリをつなぐ重要な役割を果たします。ステージングされたすべてのファイルの記録として機能し、それらの現在の状態を保持して、次のコミットに備えます。ユーザーがファイルをステージングするたびに、そのファイルに関する詳細情報がインデックスファイルに記録され、SnapTrackがコミット準備が整った変更を正確に追跡できるようにします。

インデックスの形式は、ステージングされた各ファイルの詳細情報を保存するように構成されています。インデックス内の各エントリには、次のフィルドが含まれます：

<file path> <file hash> <file status>

|  |  |
| --- | --- |
| **値** | **名** |
| 0 | Unchanged（変更なし） |
| 1 | New（新しいファイル） |
| 2 | Modified（変更あり） |
| 3 | Deleted（ファイルが削除された） |
| 表２　FileStatus列挙型 | |

**パス：**作業ディレクトリ内のファイルへの相対パスです。

**ハッシュ：**[ハッシュ](#hash)について。

**ステータス：**0から3の列挙型で、前回のコミットからの変更の種類を表す。

1. HEAD

このファイルは**[作業ブランチ](#working_branch)**の名前を指す。

1. branches

このフォルダには、リポジトリ内で作成されたブランチ名を持つファイルが含まれています。各ファイルの内容には、そのブランチ内の最新のコミットを表すオブジェクトファイルを指すハッシュが記録されています。リポジトリの初期化時には、mainブランチが作成されますが、まだコミットが行われていないため、その内容は空です。

1. remotes

このフォルダには、リモートリポジトリにアクセスできるファイル名が含まれている。

1. objects

このフォルダには、すべての[オブジェクトファイル](#object)が含まれています。

1. ignore

このファイルは、作業ディレクトリにあります。このファイルには、SnapTrackに無視させたいファイルとディレクトリのパスが含まれ ます。ファイルの形式は次のとおりです：

|  |  |
| --- | --- |
| **パターン** | **説明** |
| *\*.extension*  例）\*.exe | 指定された拡張子のファイルは無視される。 |
| *path\to\file*  例）test\test.c | ファイルは無視され、パスはリポジトリディレクトリからの相対パスでなければなりません。 |
| *path\to\dir****\***  例）bin**\** | ディレクトリ全体は無視されます。このフォーマットでは、ディレクトリ名の最後にバックスラッシュ（￥）を入力する必要があります。そうしないとファイルとして扱われ、エラーが表示されます。 |
| 表３　ignoreファイル形式 | |

デフォルトで.snaptrack\ディレクトリと.mergeconflictファイルを無視します。

1. config

このファイルにはユーザー情報等が含まれている。フォーマットは以下の通り：

|  |  |
| --- | --- |
| **フィルド名** | **初期** |
| ユーザーネーム | not\_set |
| ユーザーメール | not@set.com |
| ユーザーID | 00000000 |
| 出力言語 | 0（日本語） |
| リモートハブ | [\\10.253.1.150\it人材\共有フォルダ\開発作業\SnapTrackHub](file:///\\10.253.1.150\it人材\共有フォルダ\開発作業\SnapTrackHub) |
| 表４　設定ファイル項目一覧 | |

# リモートリポジトリのフォルダ・ファイルの説明

|  |  |  |
| --- | --- | --- |
| **№** | **パス** | **記述名** |
| 1 | HEAD | HEAD |
| 2 | refs\branches\ | branches |
| 3 | refs\remotes\ | remotes |
| 4 | objects\ | objects |

表５　リモートリポジトリフォルダ・ファイル一覧

リモートは直接開発するためのものではないため、作業ディレクトリが存在しません。代わりに、リモート自体がリポジトリとして機能します。また、一時ファイル（インデックスファイルや無視ファイルなど）は存在しません。さらに、設定ファイルはユーザーごとに固有であるため、リモートに保存する必要はありません。残りのファイルは、ローカルの対応ファイルと同じ機能を持ちます。

# 各コマンド説明・仕様

このセクションでは、使用可能なコマンドを示し、それらの使用方法を説明します。

|  |  |  |
| --- | --- | --- |
| **№** | **コマンド** | **説明** |
| 1 | snaptrack *help* | SnapTrackの使用方法を表示する |
| 2 | snaptrack <command> -h | <command>の使用方法を説明する |
| 3 | snaptrack init | ローカルリポジトリを初期化する |
| 4 | snaptrack config *<field> <value>* | SnapTrackの設定に関連する値を表示または設定する |
| 5 | snaptrack status | 最新コミットの違いを表示する |
| 6 | snaptrack stage <paths> | ファイルをステージングする |
| 7 | snaptrack unstage <paths> | ファイルをアンステージングする |
| 8 | snaptrack commit “message” | コミットを実行する |
| 9 | snaptrack commit -c “message” | 特別なコミットを実行する |
| 10 | snaptrack commit -l | コミットをリストで表示する |
| 11 | snaptrack clean | 追跡外のオブジェクトを削除する |
| 12 | snaptrack revert <hash> | 指定したコミットの変更を元に戻す |
| 13 | snaptrack restore <paths> | 指定したパスのファイルを直前のコミットから復旧する |
| 14 | snaptrack branch *<branch name>* | 新しいブランチを作成する |
| 15 | snaptrack branch -l | ブランチをリストで表示する |
| 16 | snaptrack branch -d <branch name> | ブランチを削除する |
| 17 | snaptrack checkout <branch name> | 作業ブランチを入れ替える |
| 18 | snaptrack merge <branch name> | マージを実行する |

表６　コマンド一覧

* snaptrack *help*

SnapTrackの使用方法を表示します。「help」が入力されない場合、このコマンドは同じ動作をします。

* snaptrack <command> -h

指定されたコマンドの使用方法を表示します。

* snaptrack init

ローカルリポジトリを初期化し、表1で指定された必要なフォルダとファイルをそれぞれのパスに作成します。HEADファイルは初期化時に文字列mainが設定されます。config ファイルは表3の仕様に従って作成および設定されます。ただし、configファイルがすでに存在する場合は、変更や上書きは行われません。

* snaptrack config *<field> <value>*

コマンドがフィルドと値を指定せずに入力された場合、すべての設定値が表示されます。フィルドのみを指定した場合は、そのフィルドに関連する値が表示されます。フィルドフフィルドした場合は、指定された値に従って該当するフィルドが更新されます

* snaptrack status

作業ディレクトリの現在の状態と作業ブランチ内の最後のコミットとの違いを表示します。変更はステージング済みファイルと未ステージングファイルで分類されます。コマンドの出力形式は次のとおりです：

|  |
| --- |
| ステージング済み  　　<file status> <file path>  未ステージング  　　<file status> <file path> |

**変更**されたファイル、**新規**ファイル、または**削除**されたファイル**のみを表示**します。変更されていないファイルは出力に含まれません。

* snaptrack stage <paths>

コマンド引数で指定されたディレクトリ内の特定のファイルをステージングします。ユーザーは、「.」を引数として指定することで、作業ディレクトリ内のすべての変更をステージングすることができます。また、引数として1つ以上のファイルやディレクトリをリスト形式で指定することも可能です。以下は使用例を示します。

snaptrack stage .

snaptrack stage file

snaptrack stage file1 file2 dir1\ dir2\dir3\file3

<paths>引数が指定されていない場合、snaptrack stage -hと同じ動作をします。

* snaptrack unstage <paths>

stageコマンドと同様に動作しますが、ファイルをステージングする代わりにアンステージングします。stageコマンドと同じ形式に従います。

* snaptrack commit “message”

作業ブランチでコミットを実行します。最後のコミット以降に変更がない場合は、メッセージを表示して何も実行しません。作業ディレクトリに拡張子が.mergeconflictのファイルが存在する場合、ユーザーに対して競合を解決するよう警告メッセージを表示し、何も実行しません。コミットメッセージが指定されていない場合、このコマンドはsnaptrack commit -hと同じ動作をします。

* snaptrack commit -c “message”

コミットを実行しますが、ベースファイルとの差分のみを保存する代わりに、ファイル全体のコピーを作成します。コミットおよびコードベースファイルの保存方法についての説明は後述します。

* snaptrack commit -l

作業ブランチのコミット一覧を表示します。形式は以下の通りです：

|  |
| --- |
| commit <コミットハッシュ>  Author: <ユーザーネーム> <ユーザーID> <ユーザーメール>  Date: <コミット日時>  <コミットメッセージ> |

* snaptrack clean

リポジトリのどのブランチのどのコミットでも追跡されていないオブジェクトファイルをすべて削除します。

* snaptrack revert <hash>

コミットハッシュを引数として受け取り、指定されたコミットの状態を作業ディレクトリにリバートします。この操作により、直近のコミットに続く新しいコミットが作成され、ブランチのコミット履歴は維持されます。コミットハッシュが最新のコミットと一致する場合は、エラーメッセージを表示して何も実行しません。同様に、コミットハッシュがブランチの履歴内の既存のコミットに対応していない場合も、エラーメッセージを表示して何も実行しません。

* snaptrack restore <file>

このコマンドはstageコマンドと同様に、パスのリストを引数として受け取ります。指定されたファイルやディレクトリの状態を、最後のコミット時の状態に復元します。作業ディレクトリ内で新規に作成されたファイルが指定された場合、その内容を消去しますが、ファイル自体は削除しません。作業ディレクトリ内で削除されたファイルが指定された場合、最後にコミットされた内容でファイルを再作成します。引数としてパスが指定されていない場合は、エラーメッセージを表示して何も実行しません。

* snaptrack branch *<branch name>*

新しいブランチを作成しますが、そのブランチに入れ替えません。ブランチがすでに存在する場合は、何も実行しません。引数としてブランチ名が指定されていない場合、現在の作業ブランチ名を表示します。

* snaptrack branch -l

リポジトリのブランチの一覧を表示する。

* snaptrack branch -d <branch name>

引数で渡された名前のブランチを削除します。ブランチが作業ブランチである場合は削除できませんし、main branchを削除することもできません。

* snaptrack checkout <branch name>

現在の作業ブランチを、引数で渡されたブランチに入れ替えます。現在の作業ブランチにコミットされていない変更がある場合、入れ替えは実行できません。

* snaptrack merge <branch name>

指定したブランチの変更を現在の作業ブランチに統合します。このプロセスは、以下のルールと手順に従って実行されます。

**ルール：**

1. 指定された<branch name>が現在の作業ブランチと同じ場合、マージは許可されません。
2. <branch name>がmainブランチである場合も、マージは許可されません。

**手順：**

1. ブランチポイントを取得
   * <branch name>と作業ブランチが分岐したコミットのハッシュを特定します。
2. 新しいファイルの処理

* マージ対象ブランチに存在し、作業ブランチに存在しないファイルは、そのまま作業ブランチにコピーされます。

1. 共有ファイルの処理

* 両方のブランチに存在するファイルについて：
  + ファイルが同じハッシュを持つ場合、何も行いません。
  + ハッシュが異なる場合：
    - 作業ブランチのファイルがブランチポイントのハッシュと一致する場合、マージ対象ブランチのファイルで置き換えます。
    - マージ対象ブランチのファイルがブランチポイントのハッシュと一致する場合、何も行いません。
    - 両方のファイルがブランチポイントのハッシュと一致しない場合：
      * 作業ブランチのファイルをfile\_name\_BRANCHNAME.file\_extension.mergeconflictの形式でリネームします。
      * マージ対象ブランチのファイルを同じ形式でコピーします。

1. リポジトリをステージング

* すべての変更をステージングしますが、.mergeconflict拡張子のファイルは常に無視されます。

1. 変更をコミット

* ステージングされた変更をコミットします。ただし、リポジトリ内に.mergeconflictファイルが存在する場合、コミットは許可されません。

この仕組みにより、マージの際にデータが意図せず上書きされることを防ぎ、競合が発生した場合には解決が求められるように設計されています。

# 各構造体説明

* File

ファイルとそのプロパティを保存するために使用されます。

|  |  |  |
| --- | --- | --- |
| **プロパティ名** | **プロパティ型** | **説明** |
| path | 文字列 | 作業ディレクトリからの相対的なファイルパスを表す |
| hash | 文字列 | ファイル内容のハッシュ |
| status | [FileStatus列挙](#filestatus) | ファイルのステータス |
| staged | 数字 | ファイルがステージングステータス |

作業ディレクトリ内のファイル、またはインデックスファイル内のファイルエントリを表します。また、複数のファイル構造を格納するリストとして使用でき、ハッシュやファイルの異なる状態を効率的に比較することが可能です。

* Source

|  |  |  |
| --- | --- | --- |
| **プロパティ名** | **プロパティ型** | **説明** |
| type | SourceType列挙 | ファイルの種類を示す |
| prev\_hash | 文字列 | ベースファイルのハッシュ |
| body\_size | 数字 | ボディの長さ |
| body | 文字列／DeltaInstruction構造体 | ボディの値 |

作業ディレクトリ内のソースファイルを表します。ただし、これはリポジトリ内でファイル内容を格納するために使用されます。ソースファイルは2つの異なる方法で格納されており、その詳細は次のセクションで説明されます。

* DeltaInstruction

|  |  |  |
| --- | --- | --- |
| **プロパティ名** | **プロパティ型** | **説明** |
| type | 列挙 | 命令の種類を示します（COPYまたはINSERT） |
| offset | 数字 | COPYの場合、基準ファイル内のコピー開始位置を指定します |
| length | 数字 | コピーまたは挿入するデータのバイト数を指定します。 |
| data | 可変長 | INSERTの場合、挿入する新しいデータを格納します。COPYでは不要です。 |

この構造体について、Sourceファイルの説明に詳しく説明します。

* Commit

|  |  |  |
| --- | --- | --- |
| **プロパティ名** | **プロパティ型** | **説明** |
| hash | 文字列 | このコミットのハッシュ |
| prev | 文字列 | 直前コミットハッシュ |
| index | 文字列 | インデックスファイルハッシュ |
| user\_info | User構造体 | ユーザーの情報 |
| timestamp | 文字列 | このコミットの実行日時 |
| message | 文字列 | コミットのメッセージ |

* User

|  |  |  |
| --- | --- | --- |
| **プロパティ名** | **プロパティ型** | **説明** |
| name | 文字列 | ユーザー名 |
| email | 文字列 | ユーザーメール |
| id | 文字列 | ユーザーID |

* Config

|  |  |  |
| --- | --- | --- |
| **プロパティ名** | **プロパティ型** | **説明** |
| user\_info | User構造体 | ユーザーの情報 |
| out\_lang | OutLang列挙 | 出力言語 |
| st\_hub\_path | 文字列 | リモートハブのパス |

* OutLang列挙

|  |  |
| --- | --- |
| **値** | **名** |
| 0 | Japanese |
| 1 | English |

# 各オブジェクトファイル形式説明

* Sourceオブジェクト

ソースオブジェクトファイルは、作業ディレクトリのソースファイルをリポジトリに格納したものを表します。このオブジェクトは、ユーザーがファイルをステージングした際に作成されます。ステージングされたファイルがリポジトリにとって新しい場合は、タイプ1（コピー）のソースオブジェクトが作成されます。一方、ステージングされたファイルが既存のファイルで変更が加えられている場合は、タイプ2（デルタ）のソースオブジェクトが作成されます。これらの2つのタイプは、次の列挙型で定義されています。

**SourceType列挙**

|  |  |
| --- | --- |
| **値** | **名** |
| 0 | Copy |
| 1 | Delta |

ソースファイルのファイル名は、作業ディレクトリ内の対応するファイルの内容を表すハッシュです。ソースファイルの内容は、Source構造体のバイナリ表現です。

**タイプ1**（コピー）のソースファイルは、次の形式に従います。

コピーのソースオブジェクトファイルは、ステージングまたはコミット時点での作業ディレクトリから取得されたソースファイルの完全なスナップショットを表します。デルタファイルがファイルの変更部分のみを保存するのに対し、フルコピーのソースオブジェクトファイルにはファイル全体の内容が含まれています。このタイプのオブジェクトは、リポジトリに新規追加されたファイルのように、差分を計算する以前のバージョンが存在しない場合に使用されます。

ファイルはzlib圧縮形式で保存されており、将来の参照や再構築に必要なデータを完全に保持しつつ、ストレージ効率を向上させています。

**タイプ2**（デルタ）のソースファイルは、次の形式に従います。

デルタファイルは、コミット間で行われたファイルの変更を保存し、ファイル全体ではなく差分のみを保存することで、ストレージ効率を向上させます。デルタファイルのボディは、DeltaInstruction構造体のリストで構成されています。

DeltaInstruction構造体は、デルタファイル内の個々の変更を表すために使用されます。この構造体には、**コピー (copy)** と **挿入 (insert)** の2つのタイプがあります。

* **コピー (copy)** 命令は、リポジトリ内の他の場所に既に保存されているデータの一部を参照します。この命令には以下のプロパティが含まれます:
  + **offset**: コピー元のデータの開始位置。
  + **length**: コピーするデータのサイズ。
* 一方で、**挿入 (insert)** 命令は、新しいデータをファイルに追加します。この命令には以下のプロパティが含まれます:
  + **data**: 挿入する新しい内容。
  + **length**: 挿入されるデータのサイズ。

これらの命令を組み合わせることで、デルタファイルはストレージ要件を最小限に抑えつつ、更新されたファイルを効率的に再構築します。

* Indexオブジェクト

コミットを実行した時点での作業ディレクトリの状態を表します。このファイルの内容は、File構造体のリストをバイナリ形式で格納したものです。インデックスオブジェクトファイルのファイル名は、File構造体リストから生成されたハッシュ値となります。このハッシュは、その時点の作業ディレクトリの正確な状態をコミットに関連付けるために、コミットオブジェクトで使用されます。

* Commitオブジェクト

コミット構造体をバイナリ形式で表現したもので、特定のコミットに関連付けられたすべてのメタデータや参照をカプセル化しています。これには、作成者、タイムスタンプ、コミットメッセージ、そしてコミット時点の作業ディレクトリの状態を表すインデックスオブジェクトのハッシュなどの詳細が含まれます。この情報を保存することで、変更履歴が保持され、正確に再構築することが可能になります。

# SnapTrackロジック説明

このセクションでは、各コマンドのロジックについて詳細に説明します。繰り返しを避けるため、ファイルの比較、読み取り、書き込みなどの主要なロジックプロセスは個別に説明します。この方法により、説明が明確になると同時に、コマンドの動作を包括的に理解できるようになります。

想定されるユーザー入力ごとに分けられ、それぞれの入力に対応するロジックが説明されます。また、それに対応する出力形式も示され、コマンドの動作や応答を明確に理解できるようにします。ユーザー入力はドル記号（$）で表され、一般的なプロセスのロジックは矢印（⇨）で示されます。この表記法により、ユーザーが入力するコマンドと、それを処理する内部ロジックを区別して理解しやすくなります。

**原則として、特に明記されていない限り、エラーメッセージの後にプログラムは終了します。**

### snaptrack

「[出力１](#_出力１)」を表示する

### snaptrack help

上述と同じ出力を生成する

* リポジトリが**存在する**場合エラー

.snaptrackフォルダが既に存在するか確認する

存在する場合、「[エラー１](#_エラー１)」を表示する

* リポジトリが**存在しない**場合エラー

.snaptrackフォルダが既に存在するか確認する

存在しない場合、「エラー３」を表示する

* 設定ファイルの初期化

設定ファイル（[表1](#table1)の8）が存在するか確認する

存在しない場合は、ファイルを作成し、[表4](#table4_config)に指定された値で初期化する

### snaptrack init

リポジトリが既に存在する確認する

[表１](#table1)のフォルダ・ファイル1-7を生成する

HEADファイルは「main」文字列で初期化する

設定ファイルを初期化する

「[出力２](#_出力２)」を表示する

### snaptrack init -h

「[出力３](#_出力３)」を表示する

### snaptrack config

設定ファイルを初期化する

設定ファイルの全内容を表示する（[出力４](#_出力４)）

* 指定されたフィルドが有効か確認

指定されたフィルドが有効な設定フィルドであるか確認する（可能なフィルドは[表４](#table4_config)）

無効であれば「[エラー２](#_エラー２)」を表示する

### snaptrack config <field>

指定されたフィルドが有効か確認

設定ファイルを初期化する

指定されたフィルドの値を表示する（[出力５](#_出力５)）

### snaptrack config <field> <value>

指定されたフィルドが有効か確認

設定ファイルを初期化する

指定されたフィルドの値を<value>に設定する（[出力６](#_出力６)）

### snaptrack config -h

「[出力７](#_出力７)」を表示する

* 関数１：指定された**パス**のファイルを取得

1. .snaptrackignoreファイルを読み取り、無視するパターンをリストに格納する。
2. **パス**がディレクトリの場合：
   1. ディレクトリ内を再帰的に走査して、すべてのファイルを取得する。
   2. 見つかった各ファイルについて以下を実行する：
      1. ファイルのパスを無視パターンと照合し、無視必要があるかを判断する。
      2. パターンにマッチする場合は、そのファイルをスキップする。
      3. マッチしない場合：
         1. ファイルのパスとその内容のハッシュを含むFile構造体を作成する。
         2. File.status = Newとし、File.staged = 0（未ステージ）に設定する。
         3. 作成したFile構造体をFileリストに追加する。
3. **パス**がファイルの場合：
   1. ファイルが無視パターンにマッチするかを確認する。
      1. マッチする場合は、そのまま処理を終了する。
   2. マッチしない場合：
      1. ファイルのパスを含むFile構造体を作成する。
      2. File.staged = 0に設定する。
      3. ファイルが存在するかを確認する：
         1. ファイルが存在する場合：
            1. File.status = Newに設定し、その内容のハッシュを計算して保存する。
         2. ファイルが存在しない場合：
            1. File.status = Deletedに設定する。
4. ファイルのリストは「path files」である。

### snaptrack status

### snaptrack status -h

# エラーハンドリング

# 出力メッセージ一覧

#### 出力１

英語版

|  |
| --- |
| Usage: <command> [options]  Commands:  init Initialize a new repository in the current directory.  config Configure user settings (e.g., name, email) and output language.  status Show the status of the working directory and staged changes.  stage Add file changes to the staging area.  unstage Remove files from the staging area.  commit Record staged changes to the repository with a message.  clean Remove untracked object files from the repository.  revert Undo a commit by creating a new commit that reverses it.  restore Restore files to their last committed state.  branch List, create, or delete branches.  checkout Switch to a different branch or restore a commit.  merge Combine changes from one branch into another. |

日本語版

|  |
| --- |
| 使用法: <command> [options]  コマンド:  init 現在のディレクトリに新しいリポジトリを初期化します。  config ユーザー設定（例: 名前、メールアドレス）や出力言語を設定します。  status 作業ディレクトリとステージングされた変更の状態を表示します。  stage ファイルの変更をステージングエリアに追加します。  unstage ステージングエリアからファイルを削除します。  commit ステージングされた変更をメッセージ付きでリポジトリに記録します。  clean リポジトリから追跡されていないオブジェクトファイルを削除します。  revert 元に戻すコミットを作成して、指定したコミットを取り消します。  restore ファイルを最後にコミットされた状態に復元します。  branch ブランチを一覧表示、作成、または削除します。  checkout 別のブランチに切り替えるか、コミットを復元します。  merge 一方のブランチの変更を別のブランチに統合します。 |

#### 出力２

英語版

|  |
| --- |
| Initialized local empty SnapTrack repository |

日本語版

|  |
| --- |
| ローカルの空のSnapTrackリポジトリを初期化しました |

#### 出力３

英語版

|  |
| --- |
| Usage: snaptrack init  Initialize a new repository in the current directory |

日本語版

|  |
| --- |
| 使用法： snaptrack init  現在のディレクトリに新しいリポジトリを初期化します。 |

#### 出力４

英語版

|  |
| --- |
| Snaptrack configuration:  User name Example Name  User email example@mail.co.jp  User ID 12345678  Output language English  SnapTrackHub path \\10.253.1.150\it人材\共有フォルダ\開発作業\SnapTrackHub |

日本語版

|  |
| --- |
| SanpTrack設定：  　　ユーザー名 Example Name  　　ユーザーメール example@mail.co.jp  　　ユーザーID 12345678  　　出力言語 日本語  　　SnapTrackHubパス “\\10.253.1.150\it人材\共有フォルダ\開発作業\SnapTrackHub” |

#### 出力５

英語版

|  |
| --- |
| SnapTrack configuration:  <field> <value> |

日本語版

|  |
| --- |
| SnapTrack設定：  <field> <value> |

#### 出力６

英語版

|  |
| --- |
| SnapTrack configuration:  <field> <old value>　→　<new value> |

日本語版

|  |
| --- |
| SnapTrack設定：  <field> <old value>　→　<new value> |

#### 出力７

英語版

|  |
| --- |
| Usage: snaptrack config [<field> [<value>]]  config Print all settings in the config file  config <field> Print the value of the specified field  config <field> <value> Sets the value of the specified field to <value>  Available fields:  name User name  email User email  id User ID  out\_lang The output’s language  hub\_path SnapTrackHub path |

日本語版

|  |
| --- |
| 使用法：　snaptrack config [<field> [<value>]]  config 設定ファイル内のすべての設定を表示します  config <field> 指定されたフィルドの値を表示します  config <field> <value> 指定されたフィルドの値を<value>に設定します  利用可能なフィルド：  name ユーザー名  email　　　 ユーザーメール  id ユーザーID  out\_lang 出力言語  hub\_path SnapTrackパス |

#### 出力

英語版

|  |
| --- |
|  |

日本語版

|  |
| --- |
|  |

# エラーメッセージ

#### エラー１

英語版

|  |
| --- |
| A SnapTrack repository already exists at this location |

日本語版

|  |
| --- |
| このダイレクトリには既にSnapTrackリポジトリが存在します |

#### エラー２

英語版

|  |
| --- |
| <field> is not a valid configuration field  Execute `snaptrack config -h` for more information |

日本語版

|  |
| --- |
| <field>は有効な設定フィルドではありません  詳細はsnaptrack config -hを実行してください |

#### エラー３

英語版

|  |
| --- |
| Cannot find a SnapTrack repository at this location |

日本語版

|  |
| --- |
| SnapTrackリポジトリが見つかりません |

#### エラー

英語版

|  |
| --- |
|  |

日本語版

|  |
| --- |
|  |