

Claude Code × Codex

接続・役割分担 完全ガイド

2つのAIを、同じプロジェクトで働かせる。
このガイドを見ながら、実際に繋ぐところまで。

Claude Code = 監督

全体を見る：分解・指示・確認・まとめ

Codex = 専門家

難しい実装・別の目でのレビュー

あなた = 最後の判断

ログイン・お金・公開

つなぐ場所：同じプロジェクトのフォルダ + Git + 指示書 (AGENTS.md)

このガイドは、解説ページ「Claude Code × Codex の繋ぎ方」の続きです。ページでは「なぜ分けるのか・どう分けるのか」を説明しました。ここでは**パソコンの横に置いて、上から順に手を動かせば繋がるように**、手順・コマンド・チェックリスト・テンプレートをまとめています。

読む前に知っておくこと

- 手順は Mac で実際に動かしている方法を元にしてあります（Claude Code 2.1.278・Codex CLI 0.163 で確認）。Windows の人は、各ツールの公式手順に読み替えてください。
- 黒い枠はコマンドです。`#` で始まる行は説明なので、打たなくてかまいません。
- `< >` で囲んだ所は、自分の名前・フォルダ名に書き換えます。
- 仕様は2026年10月30日に公式ドキュメントで確認しました。ツールは頻繁に更新されるので、動かないときは最後の「公式情報の場所」を見てください。

目次

01	全体構成図	08	Claude CodeからCodexへ仕事を渡す方法
02	接続前チェックリスト	09	Codex作業後にClaude Codeへ戻す方法
03	GitHub・リポジトリの準備	10	コピー用指示テンプレート
04	Claude Code側の準備	11	衝突防止ルール
05	Codex側の準備	12	トラブル時チェックリスト
06	同じプロジェクトを扱う方法	付録	公式情報の場所・用語集
07	branch・worktreeの基本		

いちばん短い道のり（30分コース）

02 チェックリスト → 03 の「Git の準備」だけ → 04 → 05 → 08 の「方法B」を1回試す → 09 で結果を確かめる。GitHub と worktree は、慣れてからで大丈夫です。

AI同士を直接会話させる必要はありません。同じフォルダを見せて、ファイルで仕事を受け渡すのが繋ぎ方の正体です。

あなたのパソコン / プロジェクトのフォルダ my-project/ (Git で記録)

AGENTS.md

2つのAIの共通ルール

CLAUDE.md

先頭に @AGENTS.md

.handoff/

指示書と結果の置き場

作るもの

ページ・プログラムなど

Claude Code

① 分ける・指示書を書く

task.md

② 指示書

Codex

③ 範囲の中で作業

result.md

④ 結果と根拠

Claude Code

⑤ 確かめて採用

あなた

ログイン・お金・公開・採用の最終判断。AIは公開の直前で止まる。

GitHub (あとからでOK)

リポジトリを置いて共有する場所。クラウド版の Codex や Web 版の Claude Code を使うときに必要。

役割の決め方

担当	任せること	任せないこと
Claude Code	全体の設計・順番、指示書づくり、Codexを呼ぶかの判断、結果の確認と採用、記録とまとめ	ログイン・課金・公開（あなたの仕事）
Codex	難しい実装・描画、大事な変更の独立レビュー、原因不明のバグの別視点、Claude Codeが同じ原因で2回失敗した所	簡単な作業、念のための二重チェック、判断が要ること、取り消せない操作
あなた	本人のログイン・二段階認証、お金、公開・本番反映、採用の最終判断	コマンドのコピペや結果の転送（AIにやらせる）

呼ぶ回数の目安：1つの仕事につき原則1回。直してからもう一度見てもらうまでで最大2回。迷ったら呼ばない。

始める前に、上から順に確かめます。全部に✓が付いたら、03 へ進みます。

アカウントとプラン

- ☐ Claude の有料プラン（Pro・Max・Team・Enterprise など）がある。
無料プランでは Claude Code は使えません
- ☐ ChatGPT のアカウントがある。
Codex は無料を含む各プランで使えますが、プランごとに使える量の上限が違います
- ☐ どちらも、自分でログインできる（二段階認証のスマホが手元にある）

パソコン

- ☐ ターミナル（Mac は「アプリケーション」→「ユーティリティ」→「ターミナル」）を開ける
- ☐ 空き容量に余裕がある（目安：数GB）
- ☐ 作業用のフォルダを1つ決めた（例：書類の中に `my-project` ）。
iCloud Drive などの同期フォルダは避けると安心です

決めておくこと

- ☐ このプロジェクトの目的を1〜2行で言える
- ☐ AIに触らせない場所（例：本番の設定、秘密情報のファイル）を決めた
- ☐ 公開・送信・お金は、自分が最後にやると決めた
- ☐ 監督は Claude Code、Codex は呼ばれたときだけ、と決めた

入っているかの確認（ターミナルで）

```
# 3つとも番号が出ればOK。出ないものは 03〜05 で入れます
git --version
claude --version
codex --version
```

「command not found」は「まだ入っていない（または場所が登録されていない）」という意味です。壊れているわけではありません。

03

GitHub・リポジトリの準備

Git

フォルダの変更の記録をとる道具。セーブポイントを作って、前の状態に戻せる。

リポジトリ

Git で記録をとっているプロジェクトのフォルダ。

コミット

セーブポイントを1つ作ること。

GitHub

リポジトリをインターネット上に置いて共有する場所。

GitHub は「あとで」、Git は「最初から」

このガイドの方法（パソコンの中で Claude Code が Codex を呼ぶ）は、GitHub がなくても動きます。一方 Git は最初から使います。`codex exec` は、最初の設定では **Git で管理しているフォルダの中でしか動きません**（公式の安全策）。それに、Codex が何を変えたかを確認、だめなら戻すために Git が必要です。

手順1 | Git が使えるか確かめる

```
git --version
```

Mac で番号が出ず「開発者ツールをインストールしますか」と出たら、「インストール」を押して待ちます。

手順2 | 名前を登録する（最初の1回だけ）

```
# セーブポイントに残る名前。公開する場合は表に出るので、本名でなくてもOK
git config --global user.name "<あなたの名前>"
git config --global user.email "<メールアドレス>"
```

手順3 | フォルダをリポジトリにする

```
mkdir -p ~/Documents/my-project
cd ~/Documents/my-project
git init -b main
printf '.handoff/\n.claude/worktrees/\n.env\n' >> .gitignore
# 記録に入るファイルの一覧を見て、入れたくないものがないか確かめる
git status
git add -A
git commit -m "はじめのセーブポイント"
```

`.gitignore` は「記録に入れないもの」の一覧です。指示書の置き場（`.handoff/`）や秘密情報（`.env`）を入れておきます。`>` は「後ろに書き足す」指定なので、すでに `.gitignore` があるフォルダでも中身を消しません。`-b main` は、完成版のブランチの名前を main にする指定です。

手順4（任意） | GitHub に置く

クラウド版の Codex や Web 版の Claude Code を使うとき、別のパソコンと共有するときに行います。GitHub CLI（`gh`）を使うと、ブラウザでのログイン1回で済みます。

```
# GitHub CLI を入れる（Homebrew がある場合）
brew install gh
# ブラウザでログイン（ここはあなたの操作）
gh auth login
# 非公開（private）のリポジトリを作って送る
gh repo create my-project --private --source=. --remote=origin --push
```

送る前に：`--private`（非公開）を必ず付けます。`git log --stat` で、送るセーブポイントに入っているファイルを確認めます。パスワード・APIキー・顧客情報は、一度コミットすると履歴に残るので、非公開でも送りません（見つけたら、送る前に Claude Code に相談）。送るかどうかは AI ではなく、あなたが決めます。

04 Claude Code側の準備

手順1 | Claude Code を入れる（どちらか）

方法	やること	向いている人
デスクトップアプリ	Claude のデスクトップアプリを開き「Code」タブを使う。ターミナル版のインストールは不要	画面で操作したい人
ターミナル版	下のコマンドで入れる（公式の推奨方法）	コマンドに慣れている人

```
# macOS・Linux（公式インストーラー）
curl -fsSL https://claude.ai/install.sh | bash
# または Homebrew（自動では更新されないの、ときどき brew upgrade claude-code）
brew install --cask claude-code
# 確認（新しいターミナルを開いてから）
claude --version
claude doctor
```

手順2 | プロジェクトのフォルダで起動してログイン

```
cd ~/Documents/my-project
claude
```

初回はブラウザが開き、Claude のアカウントでログインします（あなたの操作）。デスクトップアプリの場合は、Code タブでこのフォルダを選びます。

手順3 | CLAUDE.md で AGENTS.md を読み込む

共通ルールは `AGENTS.md` に1か所で書き（テンプレート T1）、Claude Code 用の `CLAUDE.md` の先頭に `@AGENTS.md` と書きます（テンプレート T2）。公式ドキュメントが勧めている共有の方法です。

なぜ @AGENTS.md が要るの？

Claude Code は、`CLAUDE.md` が無いときだけ `AGENTS.md` を直接読みます（v2.1.277 以降）。`CLAUDE.md` を置くなら、`@AGENTS.md` の1行で両方を読ませるのが確実です。読み込まれたかは、Claude Code の中で `/memory` を開くと確かめられます。

手順4 | Codex を呼ぶときの許可（方法Bを使う場合）

Claude Code は、コマンドを実行する前に「実行していい？」と聞きます。最初はこの確認を残したまま使います。許可を押す前に、Codex を呼ぶコマンドに `--sandbox read-only`（または `workspace-write`）が付いているかを見ます。

「codex exec を全部許可」はしない

"Bash(codex exec *)" のような広い許可は、制限をすべて外す指定（`--sandbox danger-full-access` ・ `--dangerously-bypass-approvals-and-sandbox`）が付いたコマンドまで、確認なしで通してしまいます。また、Claude Code の禁止設定（deny）は、Codex 中の操作には効きません。**Codex 側の安全は、呼ぶときに付けた `--sandbox` で決まります。**

確認を減らしたくなったら、安全な指定を中に固定した小さなスクリプト（ランナー）を作り、**そのスクリプトだけ**を許可します。実際の運用もこの形で、ランナーが毎回「フォルダ・読むだけ／書き込み・ネットワークなし」を決まった値で付けてから `codex exec` を呼んでいます。Claude Code 自身に push させないなら、`.claude/settings.json` に次を書きます。

```
{
  "permissions": {
    "deny": ["Bash(git push *)"]
  }
}
```

今の許可・禁止の設定は、Claude Code の中で `/permissions` を開くと確かめられます。

手順1 | Codex CLI を入れる（どれか1つ）

```
# Homebrew
brew install --cask codex
# npm (Node.js がある場合)
npm install -g @openai/codex
# macOS・Linux の公式スクリプト
curl -fsSL https://chatgpt.com/codex/install.sh | sh
# 確認
codex --version
```

ChatGPT のデスクトップアプリからも Codex を使えます（アプリで Codex を選び、フォルダを開く）。ただし Claude Code から呼ぶには、ターミナルで `codex` が動く状態が必要です。

手順2 | ChatGPT のアカウントでサインイン

```
codex login
# サインインできたか
codex login status
# うまく動かないときの診断
codex doctor
```

ブラウザが開くので、ChatGPT のアカウントでサインインします（あなたの操作）。API キーでもサインインできますが、その場合はプランの枠ではなく使った分の料金になります。クラウド版の Codex は ChatGPT のサインインが必要です。

手順3 | AGENTS.md が読まれるか確かめる

Codex は、プロジェクトのフォルダの `AGENTS.md` を自動で読みます（パソコン全体の共通ルール `~/.codex/AGENTS.md` も読みます）。

手順4 | 読むだけで1回動かしてみる

```
cd ~/Documents/my-project
mkdir -p .handoff
codex exec --sandbox read-only -c approval_policy="never" \
  -o .handoff/result.md \
  "AGENTS.md を読んで、このプロジェクトのルールを3行でまとめて。ファイルは変更しないで。"
cat .handoff/result.md
```

- `codex exec` は、対話をせずに1回で作業を終える「非対話モード」です。Claude Code から呼ぶときはこれを使います。
- `--sandbox read-only` は「読むだけ」の指定です。何も指定しなくても既定は読むだけですが、設定ファイルで変えている場合に備えて、試すときは明示します。
- `-o` は「最後の返事をこのファイルに書く」指定です。Claude Code はこのファイルを読みます。
- 「Git リポジトリの中ではない」と出たら、03 の手順3が終わっているか確かめます。

sandbox（作業の範囲）の3段階

指定	できること	使う場面
<code>read-only</code> （既定）	読む・調べる・コマンドで確かめる。書き込みは不可	レビュー・調査・原因探し
<code>workspace-write</code>	作業フォルダの中だけ書き込める	範囲を決めた実装・修正

指定	できること	使う場面
<code>danger-full-access</code>	制限なし	このガイドでは使いません

06 同じプロジェクトを扱う方法

2つのAIに同じプロジェクトを扱わせるときの決まりは、3つだけです。

1. 同じフォルダを開く

Claude Code も Codex も、プロジェクトのフォルダ（`my-project/`）で動かす。`codex exec` では `-C` でフォルダを指定できる。

2. ルールは1か所

`AGENTS.md` に書き、`CLAUDE.md` から `@AGENTS.md` で読み込む。片方にだけ書かない。

3. 受け渡しはファイルで

指示書は `.handoff/task.md`、結果は `.handoff/result.md`。会話の中身ではなく、ファイルの場所を渡す。

フォルダの形（例）

```
my-project/
├── AGENTS.md           # 共通ルール（Claude Code・Codex 両方が読む）
├── CLAUDE.md           # 1行目に @AGENTS.md + Claude Code だけへの指示
├── .gitignore          # .handoff/ .claude/worktrees/ .env
├── .claude/
│   ├── settings.json  # push の禁止など
│   └── skills/
│       └── codex-handoff/SKILL.md # Codex の呼び方（テンプレート T5）
├── .handoff/           # 指示書と結果（記録には入れない）
│   ├── task.md
│   └── result.md
└── （作るもの）
```

「監督は1人」を守るための3か条

1. あなたが話しかける相手は Claude Code だけにする。Codex には Claude Code から頼ませる。
2. Codex の結果を使うかどうかは、Claude Code が確かめてから決める。
3. 状態の記録（何が終わって、次に何をするか）は Claude Code が持つ。

もう1つの繋ぎ方：OpenAI 公式のプラグイン

OpenAI は Claude Code 用の公式プラグイン（`openai/codex-plugin-cc`）を出しています。Claude Code の中からコマンドで Codex にレビューや作業を頼めます。08 の「方法C」で手順と違いを説明します。

07 branch・worktreeの基本

branch（ブランチ）

同じプロジェクトの中の「別の下書きライン」。完成版（`main`）を汚さずに試せる。

worktree（ワークツリー）

ブランチごとに「別の作業フォルダ（机）」を用意する仕組み。AIごとに机を分けられる。

ブランチの基本コマンド

```
# 今どのブランチにいて、何が変わっているか（迷ったらまずこれ）
git status
# 作業用のブランチを作って移る
git switch -c work/top-page
# 何を変えたか見る
git diff
# セーブポイントを作る
git add -A
git commit -m "トップページの見出しを直した"
# 履歴を短く見る
git log --oneline -5
# 完成版に取り込む（確かめてから）
git switch main
git merge work/top-page
```

worktree が必要になるのは「同時に動かす」とき

Claude Code が Codex を呼ぶ方法（08 の方法B）では、Claude Code は Codex が終わるまで待ちます。**同時に書き込まないので、同じフォルダのまま衝突しません。**worktree を使うのは、2つのAI（や Claude Code の2つのセッション）を同時に動かしたいときです。

```
# Claude Code：専用の机を作って起動（.claude/worktrees/page/ に、ブランチ worktree-page）
claude --worktree page

# Git：Codex 用の机を、新しいブランチで作る（プロジェクトの隣のフォルダ）
git worktree add ../my-project-codex -b codex/fix
# その机で Codex を動かす
codex exec --sandbox workspace-write -C ../my-project-codex -o ../codex-result.md \
  "Read AGENTS.md, then fix the failing test in tests/. Reply in Japanese."

# Codex の机で、変更を確かめてセーブポイントを作る
git -C ../my-project-codex status
git -C ../my-project-codex diff HEAD
git -C ../my-project-codex add -A
git -C ../my-project-codex commit -m "Codex：テストの修正"
# 元のフォルダ（main）に取り込み、動かすか確かめる
git merge codex/fix
# 机の一覧・片付け（取り込んだあと）
git worktree list
git worktree remove ../my-project-codex
```

`git worktree remove` は、机にコミットしていない変更が残っていると止まります。そのときは無理に消さず（`--force` は使わない）、残っている変更を確かめます。

- デスクトップアプリの Claude Code では、セッションを始める前に「Worktree」をオンにすると、そのセッション専用の机で動きます（最初はオフ）。
- ChatGPT のデスクトップアプリの Codex にも、Local/Worktree/Cloud の実行場所の切り替えがあります。
- `codex exec --worktree` で、Codex に新しい worktree を作らせることもできます。
- 同じブランチを2つの机で同時に開くことはできません（Git の決まり）。机ごとに別のブランチにします。

08

Claude CodeからCodexへ仕事を渡す方法

渡し方は3つあります。最初は方法Aで流れを体験し、慣れたら方法B（または方法C）にします。

	方法A 手で渡す	方法B codex exec	方法C 公式プラグイン
仕組み	Claude Code が書いた引き継ぎメモを、あなたが Codex に貼る	Claude Code が <code>codex exec</code> をコマンドとして実行する	Claude Code の中で <code>/codex:review</code> などのコマンドを使う
コピペ	2回（渡す・戻す）	なし	なし
準備	ほぼなし	指示書の形・許可設定	プラグインの導入（Node.js 18.18 以上）
範囲の決め方	メモに書く	<code>--sandbox</code> で読むだけ／書き込みを選ぶ	レビューは読み取り専用、作業は <code>/codex:rescue</code>
向いている人	まず試したい	自分の型を決めて回したい（このガイドの中心）	手軽にレビューから始めたい

方法A | 手で渡す

1. Claude Code に「この部分を Codex に頼みたいので、引き継ぎメモを書いて」と頼む（テンプレート T8 の形）。
2. ChatGPT のデスクトップアプリの Codex、またはターミナルの `codex` で同じフォルダを開き、メモを貼る。
3. Codex の返事を Claude Code に貼り、「確かめて、採用するものだけ反映して」と頼む（テンプレート T7）。

方法B | Claude Code が codex exec で呼ぶ（実際に使っている方法）

1. `AGENTS.md` ・ `CLAUDE.md` ・ `.claude/settings.json` を用意する（04・06）。
2. Skill（テンプレート T5）を `.claude/skills/codex-handoff/SKILL.md` に保存する。Claude Code が「いつ・どう呼ぶか」を毎回同じ手順で行うようになる。
3. Claude Code に頼む（テンプレート T6）。Claude Code が指示書（T3）を書き、次のコマンドを実行する。

```
# 読むだけ（レビュー・調査）
codex exec --sandbox read-only -C . \
  -c approval_policy="never" \
  -o .handoff/result.md \
  "Read .handoff/task.md and follow it. Reply in Japanese."

# 書き込みあり（このフォルダの中だけ・ネットワークなし）
codex exec --sandbox workspace-write -C . \
  -c approval_policy="never" \
  -c sandbox_workspace_write.network_access=false \
  -o .handoff/result.md \
  "Read .handoff/task.md and follow it. Reply in Japanese."
```

<code>--sandbox</code>	作業の範囲（05 の表）
<code>-C .</code>	このフォルダを作業の場所にする
<code>-c approval_policy="never"</code>	途中で「実行していい？」と止まらない。範囲の外の操作は失敗して終わる（非対話では公式の推奨）
<code>-c sandbox_workspace_write.network_access=false</code>	書き込みありのとき、Codex が実行するコマンドをネットワークにつなげない（Codex 本体の通信は別）
<code>-o</code>	最後の返事をファイルに書く
最後の文	指示書の中身は貼らず「指示書を読んで従って」とだけ書く

Codex の作業は10分以上かかることがあります（実例では1回目が約16分）。Claude Code にはテンプレート T5 で「長くなりそうならバックグラウンドで実行して、終わるまで待つ」と伝えてあります。

方法C | OpenAI 公式のプラグイン (codex-plugin-cc)

```
# Claude Code の中で、1行ずつ
/plugin marketplace add openai/codex-plugin-cc
/plugin install codex@openai-codex
/reload-plugins
/codex:setup
```

/codex:review	今の変更を Codex が読み取り専用でレビューする
/codex:adversarial-review	設計の前提を疑ってかかるレビュー
/codex:rescue	作業そのものを Codex に任せる
/codex:status · /codex:result · /codex:cancel	任せた作業の状況・結果・取り消し

使った分は Codex の利用上限に数えられます。プラグインには、Claude Code の作業が終わるたびに Codex がレビューする「レビューゲート」もありますが、公式の説明でも利用上限を一気に使うおそれがあると注意されています。最初はオフのまま、必要なときだけ /codex:review を使うのがおすすめです。

09 Codex作業後にClaude Codeへ戻す方法

Codex の結果は「案」です。Claude Code が確かめてから、使うかどうかを決めます。この工程を省くと、もっともらしい間違いがそのまま完成品に入ります。

確認の5ステップ (Claude Code にやらせる)

- 1

結果を読む

`.handoff/result.md` の結論と根拠を読む。
- 2

根拠を開く

結論を支えるファイルと行を、Claude Code が自分で開いて本当か確かめる。
- 3

変更を全部見る

`git status` (新しいファイルも出る) と `git diff HEAD` (前のセーブポイントからの変更すべて) で読む。新しいファイルは中身を開く。範囲の外の変更は採用しない。
- 4

動かして確かめる

テストや確認のコマンドがあれば実行する。ページなら画面で見る。
- 5

採否を決める

指摘ごとに「採用／不採用 (理由) ／要確認」。要確認は、あなたが判断する。

結果に応じた次の一手

結果	次にやること
そのまま使える	Claude Code が作業用ブランチにコミット。完成版 (main) に入れるかは、あなたが見て決める

結果	次にやること
一部だけ使える	使う所だけ残し、残りは Claude Code が戻す（コミット前なら、そのファイルを元に戻せる）
足りない	具体的な指摘を付けて、もう1回だけ渡す（合計2回まで）
2回やっても届かない	方法を変える（範囲を狭める・別の作り方にする）。実例では、前髪の描画を画像生成に切り替えた
Codex が動かなかった	12 のトラブル表を見る。利用上限なら待つか、Claude Code だけで続ける

```
# よく使う確認コマンド（Claude Code が実行します）
git status           # 変わったファイル・新しいファイルの一覧
git diff HEAD        # 前のセーブポイントからの変更（ステージ済みも含む）
git diff --stat HEAD  # どのファイルが何行変わったか
git diff <渡す前のコミット>  # Codex がコミットまでしていた場合
```

戻すとき： Claude Code に「このファイルを、Codex に渡す前の状態に戻して」と頼みます。記録しているファイルなら `git restore --source=HEAD --staged --worktree <ファイル>` で前のセーブポイントに戻ります（ステージ済みの変更も含む）。新しく作られたファイルはこのコマンドでは消えないので、確かめてから削除します。どちらも取り消せないなので、実行の前にどのファイルかを確認します。

報告の形（Claude Code に書かせる）

```
Codex に頼んだこと：〇〇のレビュー（読むだけ・1回）
採用：指摘3件（△△の条件分岐・□□の表示崩れ・テスト不足）
不採用：指摘1件（理由：仕様どおりの動き）
要確認：1件（公開の日付をどうするか → あなたの判断）
```

同じ内容のテキスト版（`Claude Code × Codex コピペ用テンプレート.md`）も配布ページからダウンロードできます。PDF からのコピーは改行が崩れることがあるので、テキスト版から使うのがおすすめです。

T1 AGENTS.md（2つのAIの共通ルール。プロジェクトのフォルダの一番上に置く）

```
# AGENTS.md - <プロジェクト名>

## このプロジェクトの目的
<1〜2行で。例：自社サイトのお知らせページを作る>

## 役割
- Claude Code：監督。仕事の分解・指示書づくり・結果の確認・まとめを担当する
- Codex：Claude Code から呼ばれたときだけ、指示書に書かれた範囲の作業をする
- 人間：ログイン・お金・公開・最終判断

## 守ること
- 触ってよい範囲：このフォルダの中だけ
- 触ってはいけない：<例：.env、本番の設定ファイル、design/ フォルダ>
- 公開・本番への反映・push・削除は、人間の承認なしに行わない
- パスワード・APIキー・トークンを読まない・書かない・出力しない
- 作業は main ではなく、作業用のブランチで行う
- 1つのフォルダを、2つのAIが同時に書き換ええない

## 受け渡しの場所
- 指示書と結果は .handoff/ に置く（Git の記録には入れない）
```

T2 CLAUDE.md（Claude Code 用。AGENTS.md と同じフォルダに置く）

```
@AGENTS.md

# Claude Code へ
- あなたは監督です。Codex を呼ぶかどうかは、下の「呼ぶ条件」で判断してください
- 呼ぶ条件：難しい実装・描画／大事な変更の独立レビュー／同じ原因で2回続けて失敗した
- 呼ばない：簡単な作業・念のための二重チェック・人間の判断が要ること
- Codex の結果は「案」として扱い、根拠と変更を自分で確かめてから採用してください
```

T3 指示書 .handoff/task.md (Claude Code が書き、Codex が読む)

```
# 目的
<1〜2行。何を判断・作成してほしいか>

# 対象
- 見てほしいファイル：<相対パス>（中身は貼らない。自分で開いて読む）
- 前提として読むもの：AGENTS.md

# 制約
- 触ってはいけない範囲：<パス>
- 変更してよい範囲：<パス。読むだけなら「なし」>
- 公開・削除・push・外部への送信はしない

# 完了条件
<例：npm test がすべて通る／指摘を重大度の高い順に5つまで>

# 返してほしい形
## 結論
## 根拠（ファイルパス:行、実行したコマンドとその結果）
## 変更したファイル（ある場合）
## 確認できなかったこと
```

T4 Codex の呼び方 (Claude Code が実行するコマンド)

Claude Code が許可を求めてきたら、--sandbox の指定を見てから許可します。"codex exec を全部許可" はしません。

```
# 読むだけ（レビュー・調査）。何も指定しなくても read-only で動きます
codex exec --sandbox read-only -C . \
  -c approval_policy="never" \
  -o .handoff/result.md \
  "Read .handoff/task.md and follow it. Reply in Japanese."

# 書き込みあり（このフォルダの中だけ・Codex が実行するコマンドはネットワークなし）
codex exec --sandbox workspace-write -C . \
  -c approval_policy="never" \
  -c sandbox_workspace_write.network_access=false \
  -o .handoff/result.md \
  "Read .handoff/task.md and follow it. Reply in Japanese."
```

T5 Claude Code に「Codex の呼び方」を覚えさせる Skill

保存場所: .claude/skills/codex-handoff/SKILL.md (このプロジェクトだけで使う場合)

```
---
name: codex-handoff
description: 難しい実装、大事な変更の独立レビュー、同じ原因で2回続けて失敗したときに、Codex (codex exec) へ範囲を
決めて作業を渡し、結果を確かめてから続ける。簡単な作業には使わない。
---

# Codex へ渡すときの手順

1. 呼ぶ理由を1行で書けるか確かめる。書けなければ呼ばない
2. .handoff/task.md に指示書を書く (目的・対象・制約・完了条件・返してほしい形)
3. 次のどちらかを実行する。10分を超えそうなら、バックグラウンドで実行して終わるまで待つ
   - 読むだけ: codex exec --sandbox read-only -C . -c approval_policy="never" -o .handoff/result.md
     "Read .handoff/task.md and follow it. Reply in Japanese."
   - 書き込みあり: codex exec --sandbox workspace-write -C . -c approval_policy="never" -c sandbox_workspace_write.network_access=false -o .handoff/result.md "Read .handoff/task.md and follow it. Reply in Japanese."
4. .handoff/result.md を読む
5. 確かめる (省略しない)
   - 結論を支える根拠のファイル・行を自分で開く
   - git status (新しいファイルも出る) と git diff HEAD で、変更されたファイルを全部読む。新しいファイルは中身を開く。範囲外の変更は採用しない
   - テストがあれば実行する
   - 指摘ごとに「採用／不採用 (理由) / 要確認」を決める
6. 足りなければ、具体的な指摘を付けてもう1回だけ渡す (合計2回まで)
7. 報告に「Codex に何を頼み、何を採用し、何を採用しなかったか」を1〜3行で書く

# やらないこと
- 自分 (Claude Code) が権限の確認で止められた操作を、Codex に代わりにやらせない
- 公開・削除・push・課金・外部への送信を Codex にさせない
- パスワードやAPIキーを指示書に書かない
```

T6 Claude Code への依頼文 (Codex に渡してほしいとき)

この作業のうち、<難しい所>だけを Codex に渡してください。

- ・最初は読むだけ (read-only) で、問題点の洗い出しを頼む
- ・指示書は .handoff/task.md に書き、中身を見せてから実行する
- ・戻ってきたら、根拠と変更を確かめてから、採用するものだけ反映する
- ・公開・push はしない。終わったら、何を採用して何を採用しなかったかを教えて

T7 Codex から戻った結果の確認を頼む文 (Claude Code へ)

.handoff/result.md と、git status・git diff HEAD の変更 (新しいファイルも) を確かめてください。

- ・指摘ごとに「採用／不採用 (理由) / 要確認」の表にする
- ・指示書の範囲の外で変わったファイルがあれば、最初に教える
- ・テストがあれば実行して、結果を書く
- ・採用するものだけを反映して、作業用のブランチにコミットする (push はしない)

T8 手で渡すときの引き継ぎメモ（コピペで Codex に貼る）

【Claude Code からの引き継ぎ】

目的：

プロジェクトのフォルダ：

今の状態（終わっていること）：

あなたに頼みたいこと（範囲）：

読んでほしいファイル：

触ってはいけない範囲：

完了条件：

返してほしい形：結論／根拠（ファイル：行）／変更したファイル／確認できなかったこと

T9 .gitignore に足す行

```
.handoff/  
.claude/worktrees/  
.env
```

11 衝突防止ルール

AGENTS.md に書いておくと、Claude Code と Codex の両方が守ります。

1. 1つのフォルダを、2つのAIが同時に書き換えない。

順番に動かす。同時に動かすなら worktree で机を分ける。

2. 同じブランチを2か所で同時に使わない。

机（worktree）ごとに別のブランチ。

3. 作業は main 以外のブランチで。

main には、確かめたものだけを入れる。

4. Codex に渡す前にコミットする。

Codex の変更だけを `git diff HEAD` で見分けられ、いつでも戻せる。

5. Codex は最初「読むだけ」。

書き込ませるときは、フォルダの中だけ・ネットワークなし。

6. 指示書に「触ってはいけない範囲」を必ず書く。

範囲の外の変更は採用しない。

7. 監督は Claude Code だけ。

あなたが話しかけるのも、状態を記録するのも Claude Code。

8. 止められた操作を、もう一方のAIに回さない。

Claude Code が許可の確認で止めたことは、あなたが判断する。

9. 取り消せない操作は人がやる。

公開・送信・削除・push・課金。

10. 秘密情報は書かない・読ませない。

パスワード・APIキー・トークンは `.env` などに置いて `.gitignore` に入れ（記録に入れない）、`AGENTS.md` にも「読まない」と書く（`.gitignore` は AI が読むのを止めない）。

Claude Code の複数セッションも同じ

Claude Code を2つ同時に開いて同じプロジェクトを触るときも、ルール1・2は同じです。作業を始める前に `git status` と `git worktree list` で、他の作業の途中の変更がないかを確認めます。見覚えのない変更があれば、勝手に直したり消したりせず、止まって確認します。

12 トラブル時チェックリスト

こうなった	確かめること	直し方
<code>command not found: codex</code> (または <code>claude</code>)	インストールしたか／新しいターミナルを開いたか	05 (04) の手順1をやり直し、ターミナルを開き直す
Codex が「Git リポジトリの中ではない」と止まる	<code>git status</code> がエラーにならないか	03 の手順3でリポジトリにする。どうしても外で動かすなら <code>--skip-git-repo-check</code>
Codex がログインを求める・認証エラー	<code>codex login status</code>	<code>codex login</code> でサインインし直す (あなたの操作)
利用上限・混雑で動かない	エラーの文に limit などがあるか	何度も再実行しない。時間をおくか、Claude Code だけで続ける
Codex がファイルを書き換えられない	<code>--sandbox</code> が <code>read-only</code> のままか	本当に書き込みが要るか確かめてから <code>workspace-write</code>
Codex が範囲の外のファイルを変えた	<code>git status</code> ・ <code>git diff</code>	その変更は採用しない。Claude Code に、そのファイルを渡す前の状態に戻してもらう (09)。指示書の「触ってはいけない範囲」を足す
Claude Code が毎回、実行の許可を求める	コマンドに <code>--sandbox</code> が付いているか	最初はそれで正常。減らすなら、安全な指定を固定したランナーだけを許可する (04 の手順4)
Claude Code が AGENTS.md のルールを守らない	<code>/memory</code> に AGENTS.md が出るか	<code>CLAUDE.md</code> の1行目に <code>@AGENTS.md</code>
Codex の作業が終わらない・途中で切れる	頼んだ範囲が広すぎないか	範囲を狭めて1回だけやり直す。長い作業はバックグラウンドで実行させる
<code>git worktree add</code> で「既に使われている」と出る	<code>git worktree list</code>	別のブランチ名にする (<code>-b</code> で新しく作る)
<code>git merge</code> で「conflict (衝突)」	どのファイルが衝突したか (<code>git status</code>)	Claude Code に「衝突を見せて、どちらを残すか案を出して」と頼み、あなたが決める
同じ失敗を何度もくり返す	同じ原因で2回失敗したか	そこで Codex に別の目で見てもらおう (読むだけ)。3回目を同じやり方で試さない
何が起きているか分からない	<code>claude doctor</code> ・ <code>codex doctor</code>	出た文をそのまま Claude Code に見せて、原因と次の一手を聞く

このガイドの仕様は、2026年10月30日に下の公式ページで確認しました。動かないとき・新しい機能を知りたいときは、ここを見てください。

内容	公式ページ
Claude Code のインストール	code.claude.com/docs/en/setup
CLAUDE.md・AGENTS.md の読み込み	code.claude.com/docs/en/memory
Claude Code の worktree	code.claude.com/docs/en/worktrees
Claude Code の権限設定	code.claude.com/docs/en/permissions
Claude Code の Skill	code.claude.com/docs/en/skills
デスクトップアプリの Claude Code	code.claude.com/docs/en/desktop
Codex CLI（インストール）	github.com/openai/codex
Codex のサインイン	learn.chatgpt.com/docs/auth
codex exec（非対話モード）	learn.chatgpt.com/docs/non-interactive-mode
Codex のコマンド一覧	learn.chatgpt.com/docs/developer-commands
Codex の AGENTS.md	learn.chatgpt.com/docs/agent-configuration/agents-md
Codex の料金・プラン	learn.chatgpt.com/docs/pricing
Codex の公式 Claude Code プラグイン	github.com/openai/codex-plugin-cc

用語集

ターミナル

文字でパソコンに指示を出すアプリ。コマンドを貼って Enter で実行する。

コミット

セーブポイントを1つ作ること。

worktree

ブランチごとに別の作業フォルダを用意する仕組み。

非対話モード

途中で質問せず、1回で作業を終えて結果を返す動かし方（`codex exec`・`claude -p`）。

コマンド

ターミナルに打つ1行の指示。

ブランチ

同じプロジェクトの中の別の下書きライン。完成版は main。

push/pull

GitHub へ送る／GitHub から取ってくる。

AGENTS.md

AIへの共通ルールを書くファイル。Codex が自動で読み、Claude Code は CLAUDE.md から読み込める。

リポジトリ

Git で変更の記録をとっているプロジェクトのフォルダ。

マージ

ブランチの変更を、別のブランチ（例：main）に取り込むこと。

sandbox

AIが作業できる範囲の制限（読むだけ・フォルダの中だけ など）。

Skill

Claude Code に覚えさせる手順書のファイル（`SKILL.md`）。

このガイドは2026年10月時点の情報です。Claude Code・Codex の機能・料金・設定方法は変わることがあります。Claude・Claude Code は Anthropic の、ChatGPT・Codex は OpenAI の商標です。このガイドは各社とは関係のない個人の解説です。

Mogi | 小さな会社の仕組みづくり mogi-shinichi.com