10 SLIDES

ラズパイ4でのCodexはじめ方

準備 → 64bit OS → 導入 → 最初の実行 → 安全な運用

始める前に用意するもの

Codex以前に、Raspberry Piを安定したLinux開発環境にする。

最低限の構成

- Raspberry Pi 4
- 安定したUSB-C電源
- microSDまたはUSB SSD
- 有線LANまたは安定したWi-Fi
- 初期設定用の画面・入力機器、またはSSH接続

あると運用しやすいもの

- 冷却ケース／ヒートシンク
- GitHubなどのリモートリポジトリ
- 別端末から使うWebターミナルやSSH
- 空き容量と温度を確認する仕組み

電源
不安定さは停止・破損の原因

冷却
長時間処理の速度低下を防ぐ

保存先
増えるデータはSSDが扱いやすい

本サイトの実例：冷却、SSD移行、状態監視、遠隔操作を各テーマで掲載

最初の落とし穴：32bit / 64bit

私の環境では32bit版Raspberry Pi OSで導入が進まず、64bit版へ移行してCodexを起動した。

32bit OS

- 古い環境では選ばれている場合がある
- 新しい開発ツールの前提と合わないことがある
- 導入を始める前に確認する

64bit OS

- 現在の開発環境として選びやすい
- Raspberry Pi 4の64bit CPUを活用
- 移行前にデータと設定を退避する

```
uname -m
getconf LONG_BIT
```

aarch64 / 64 が確認の目安。結果だけで断定せず、OS情報も確認する。

OSを入れたら、土台を整える

先にOS・ネットワーク・時刻・空き容量を安定させると、Codexの問題と本体の問題を切り分けやすい。

1 更新

OSとパッケージを最新の安定状態へ。再起動が必要なら先に行う。

2 確認

ネットワーク、時刻、CPU温度、メモリ、ディスク空き容量を確認。

3 保全

作業データをGitや別ストレージへ退避。秘密情報はリポジトリへ入れない。

4 作業場所

プロジェクトごとにフォルダを分け、Codexをそのルートから起動する。

```
sudo apt update
sudo apt full-upgrade

df -h
free -h
vcgencmd measure_temp
```

一度に全部変えず、更新 → 再起動 → 確認の順で進める

Node.jsを確認し、Codex CLIを導入

Codex CLIはターミナルで動く。インストール前後のバージョンを記録しておく。

1. Node.js / npm

- 64bit環境向けのNode.jsを用意
- node と npm が実行できるか確認
- 古い環境を無理に継ぎ足さない

2. Codex CLI

- 公式パッケージをグローバル導入
- codex --version で確認
- 更新時も公式案内を優先

```
node --version
npm --version

npm install --global @openai/codex
codex --version
```

インストール方法は更新されるため、公開時点の公式ドキュメントも確認

OpenAI公式：Codex CLI Getting Started

起動してChatGPTでサインイン

作業フォルダへ移動して codex を起動。画面の案内に従い、ChatGPTアカウントでサインインする。

1 移動

Codexに見せるプロジェクトのルートへ移動する。

2 起動

codex を実行。初回セットアップ画面を確認する。

3 認証

表示されたサインイン手順を別端末のブラウザも使いながら進める。

4 権限

最初は既定の安全な権限で開始し、要求内容を読んで判断する。

```
cd ~/projects/my-first-project
codex
```

APIキーの貼り付けを前提にせず、まずはCodexが案内するChatGPTサインインを使う。

利用可能なプラン・上限は変わるため、現在のアカウント画面で確認

OpenAI公式：Using Codex with your ChatGPT plan

最初の依頼は「調査 → 小さな変更 → 検証」

曖昧な大仕事より、対象・目的・完了条件を短く伝える。

伝えるとよい情報

- 何を作っているか
- 変更してよい範囲
- 使うコマンド
- 完了とみなす条件

確認するもの

- 変更されたファイル
- 実行されたコマンド
- テスト結果
- 意図しない削除や秘密情報

最初のプロンプト例

このフォルダの構成を確認し、READMEの内容を要約してください。変更はまだ行わず、次に試す小さな改善を3つ提案してください。

読む → 提案を見る → 一つ選ぶ → 変更差分を見る → テストする

権限・Git・秘密情報の3点を守る

Codexは作業を進められるからこそ、境界と復旧手段を先に作る。

権限

- 最初は既定のサンドボックス
- ネットワークや外部書き込みは理由を確認
- 分からないコマンドは承認前に質問

Git

- 作業前に状態を確認
- 一つの目的ごとに差分を小さくする
- 動いた時点でコミットを残す

秘密情報

- .envや認証ファイルを公開しない
- APIキーをプロンプトやログへ貼らない
- 公開前にステージ対象を確認

```
git status
git diff

# 内容を確認してから
git add <必要なファイルだけ>
```

Pi 4では、重い作業を分けて運用する

Codex本体だけでなく、ビルド・画像・ブラウザ・常駐サーバーが同じCPUとメモリを使う。

安定させる習慣

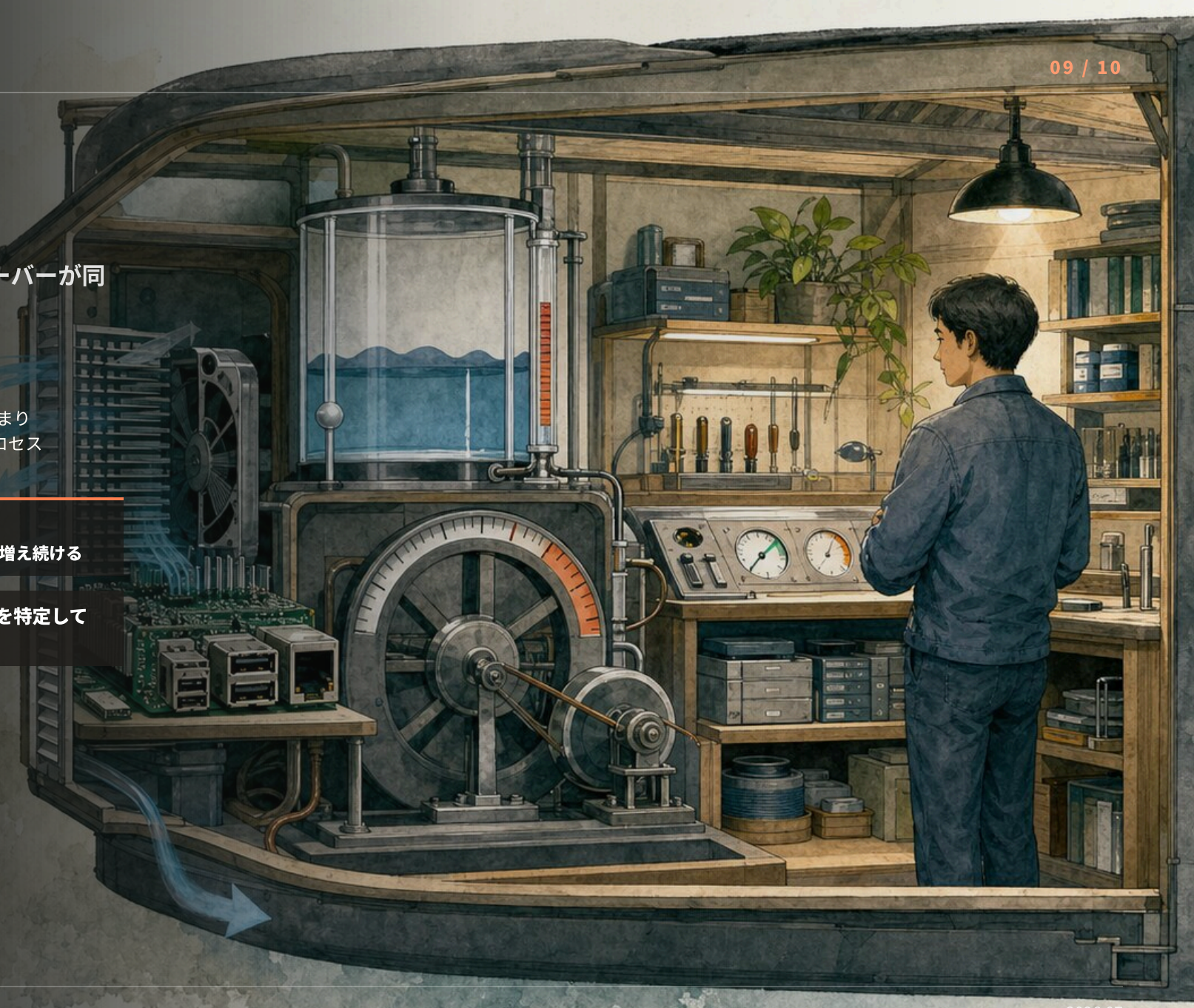
- ビルドと画像処理を同時に走らせない
- プレビューサーバーは検証後に止める
- free -h と uptime で余裕を確認
- 大きな処理は1ファイルずつ

止まりやすい兆候

- swapがほぼ埋まる
- CPU温度やロードが高止まり
- Next.jsやNodeの残留プロセス
- microSDの空き容量不足

CPU	RAM	STORAGE
同時実行を減らす	空きとswapを両方見る	履歴・画像は増え続ける

止まったら、キャッシュ全消去ではなく、自分が起動したプロセスだけを特定して終了する。



次に作るものを、一つ選ぶ

Raspberry Piの価値は、GPIO・カメラ・センサー・ローカルデータへCodexの開発力をつなげられること。

初級

- READMEを整える
- 温度を表示する小さなページ
- CSVを読む簡単なグラフ

中級

- BME280の履歴ダッシュボード
- iPhone向けWebターミナル
- カメラのパン・チルトUI

運用

- systemdでサービス化
- GitHubへバックアップ
- 状態監視と障害記録

最初の一歩：作業フォルダを一つ作り、Codexに「変更せず構成を説明して」と頼む。

64bit化から起動までの実話 ↗　Raspberry Pi実践ガイド ↗