Part 4 ／ 120分

自分のPCを、AIの作業場にする

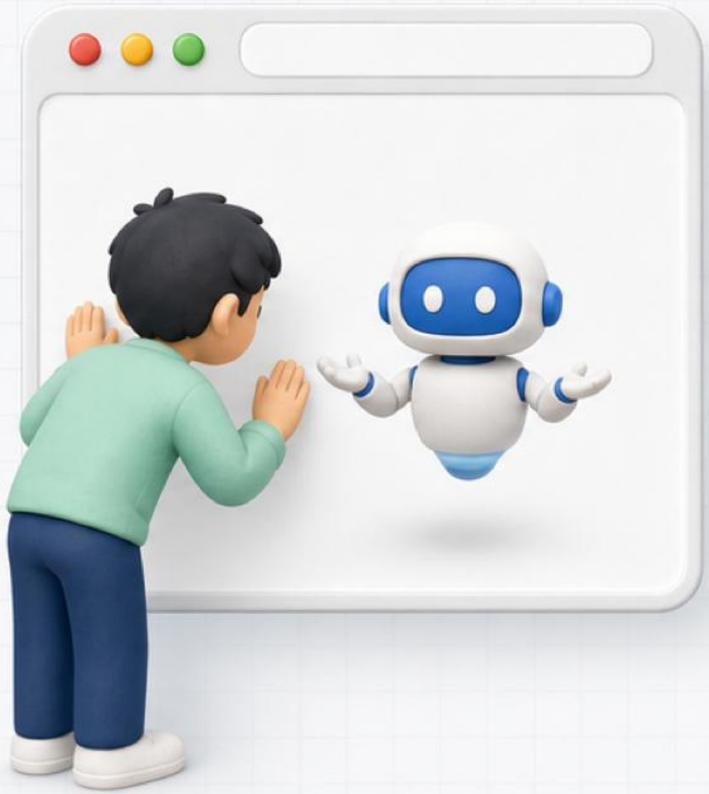
AI駆動開発の環境整備 ― 120分ハンズオン



- uv
- VS Code + Copilot
- Git + GitHub



ブラウザの外に出ると、何が変わるか





= 環境と依存関係を、自分が所有する

開発の『新・三種の神器 2026』



uv ― Python 本体とライブラリを
管理する (実行環境)



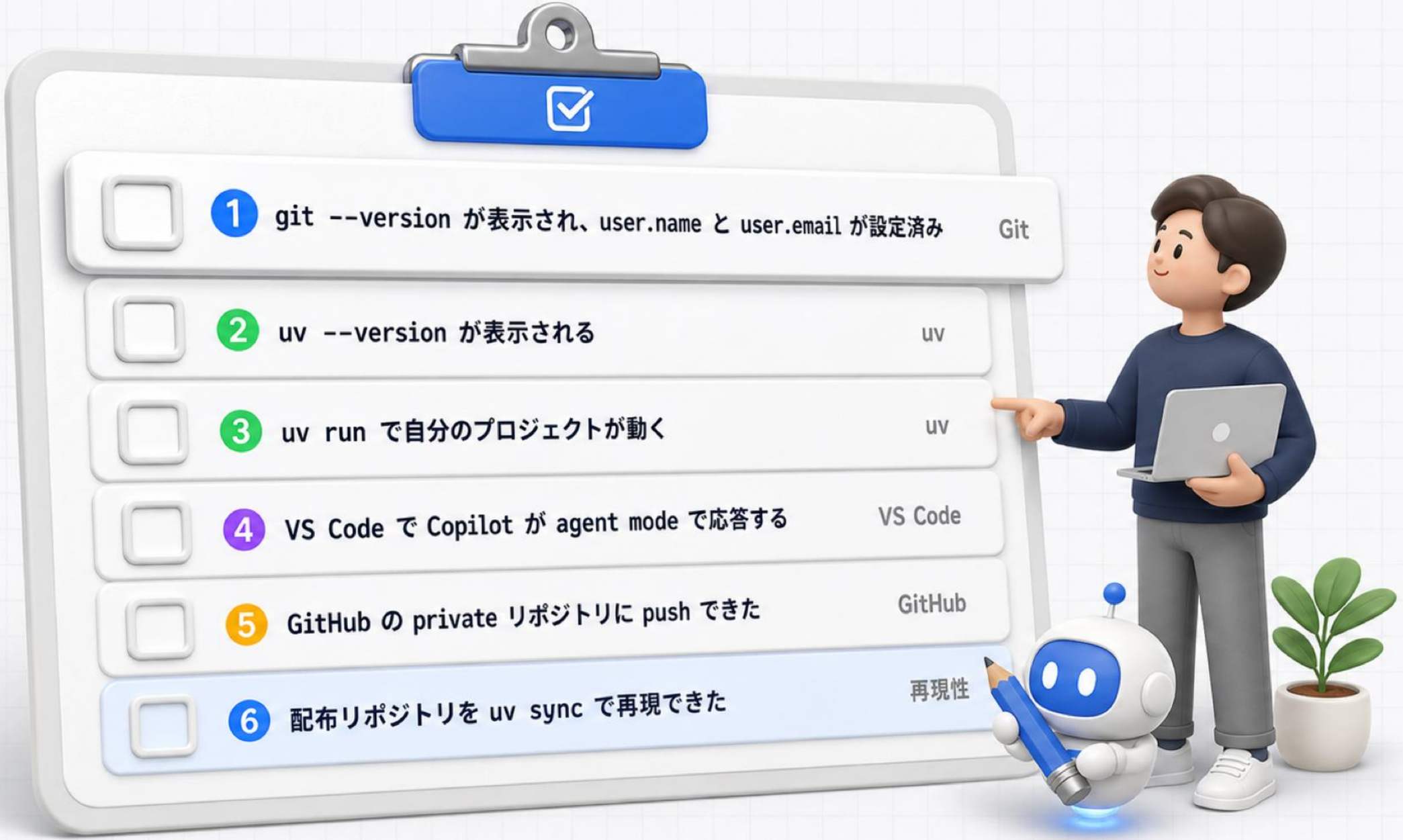
VS Code + GitHub Copilot ―
コードを書く場所と、
書いてくれる相棒 (作業場)



Git + GitHub ―
履歴を残し、外に置く
(セーブと保管庫)

3つで1セット。どれか1つでは AI駆動開発にならない

本パートのゴール: 環境チェックリスト



Git のインストールと初期設定

Windows



https://git-scm.com/download/win からインストーラを実行（既定のままでよい）

Mac



ターミナルで xcode-select --install
（Xcode Command Line Tools に Git が含まれる）



共通：初期設定（この3行は全員実行）

① `git config --global user.name "自分の名前"`

② `git config --global user.email "GitHubに登録したメールアドレス"`

③ `git config --global init.defaultBranch main`

成功したらこう見える：git --version でバージョンが表示され、git config --global --list に自分の名前とメールが出る

uv とは ― Python 環境の管理を1本化する



uv

Python 本体のインストール ／ 仮想環境 ／ ライブラリ管理 ／ 実行 ― 全部これ1つ

- Python そのものを、バージョン指定で入れられる: uv python install 3.12
- Rust 製で高速。数十秒待っていた依存解決が数秒になる
- 2026年の Python 開発の標準的な選択肢



`uv python install` `uv init` `uv run` `uv add` `uv sync`

要点: 今日は5つのコマンドだけ覚える: uv python install ／ uv init ／ uv run ／ uv add ／ uv sync

uv と Python のインストール

① uv を入れる

※ すでに入っている人は確認だけ

Windows (PowerShell)

```
powershell -ExecutionPolicy ByPass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

Mac (ターミナル)

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```



コマンドを打つのが大変なら:「uv install」で検索 → 公式ページ (astral.sh) からコピー

② Python 3.12 を入れる (OS 共通・全員実行)




```
uv python install 3.12
```



成功したらこう見える: `uv --version` でバージョン番号、`uv python list` に 3.12 が表示される



詰まりどころ: uv が見つからない → ターミナルを一度閉じて開き直す (PATH の再読み込み)

uv init でプロジェクトを作る

```
uv init ai-workshop
```

```
cd ai-workshop
```



成功したらこう見える（フォルダの中身）



1. pyproject.toml — このプロジェクトが必要とするものの一覧（人が読む・書く）
2. src/ai_workshop/ — 自分のコードを置く場所（動くサンプル入り）
3. .python-version — 使う Python のバージョン
4. .gitignore ／ README.md — あわせて自動生成される
5. uv.lock — 実際に入れた全部品の正確な版（機械が書く・触らない）

※ uv init は Git リポジトリとしての初期化（.git）も済ませてくれる — 後半で使う

※ uv.lock は最初の uv run の後に現れる

uv run で実行する



```
$ uv run ai-workshop
```



成功したらこう見える

```
Hello from ai-workshop!
```



初回は Python 本体と仮想環境の準備が走る（数秒〜数十秒、ログが流れる）。2回目からは即実行

3 → チェックリスト ③ が埋まりました

uv add でライブラリを追加する

成功したらこう見える（pyproject.toml が書き換わる）

```
> uv add requests
```



pip install は使わない。uv add を使う

```
[]
```

```
]
+
```

dependencies = [] → requests の行が1行追加される

同時に uv.lock も更新される

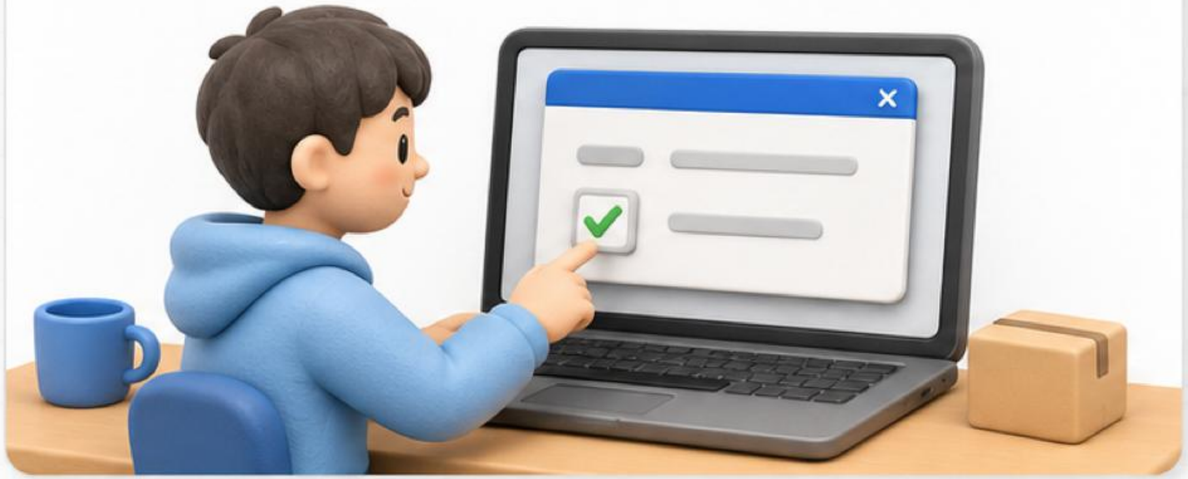
ー レシピと納品書が一緒に更新される

要点： 午後のワークでは全員 uv add streamlit を実行します

VS Code のインストール

配布元: https://code.visualstudio.com/

Windows ― インストーラを実行。
「PATH への追加」にチェックを入れる



Mac ― zip を展開し、Visual Studio Code.app を
「アプリケーション」フォルダへ移動



成功したらこう見える: VS Code が起動し、
ターミナルで code --version が表示される



※ すでに入っている人は起動確認だけ。
次のスライドへ

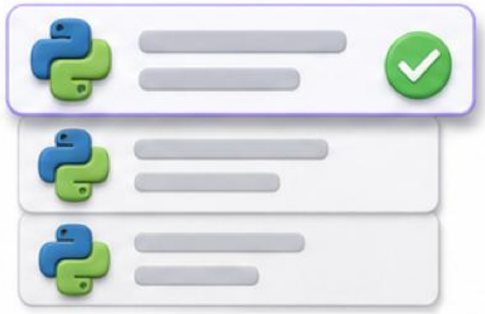
日本語化と Python 拡張



① 

`Japanese Language Pack for Visual Studio Code`

を検索してインストール → 再起動

② 

`Python`
(Microsoft 製)
を検索してインストール



● ※ 英語のままで使いたい人は 1 を飛ばしてよい

成功したらこう見える：メニューが日本語になり、.py ファイルで色分けと補完が効く



GitHub にサインインする



① 左下のアカウントアイコンをクリック

② GitHub でサインインを選ぶ

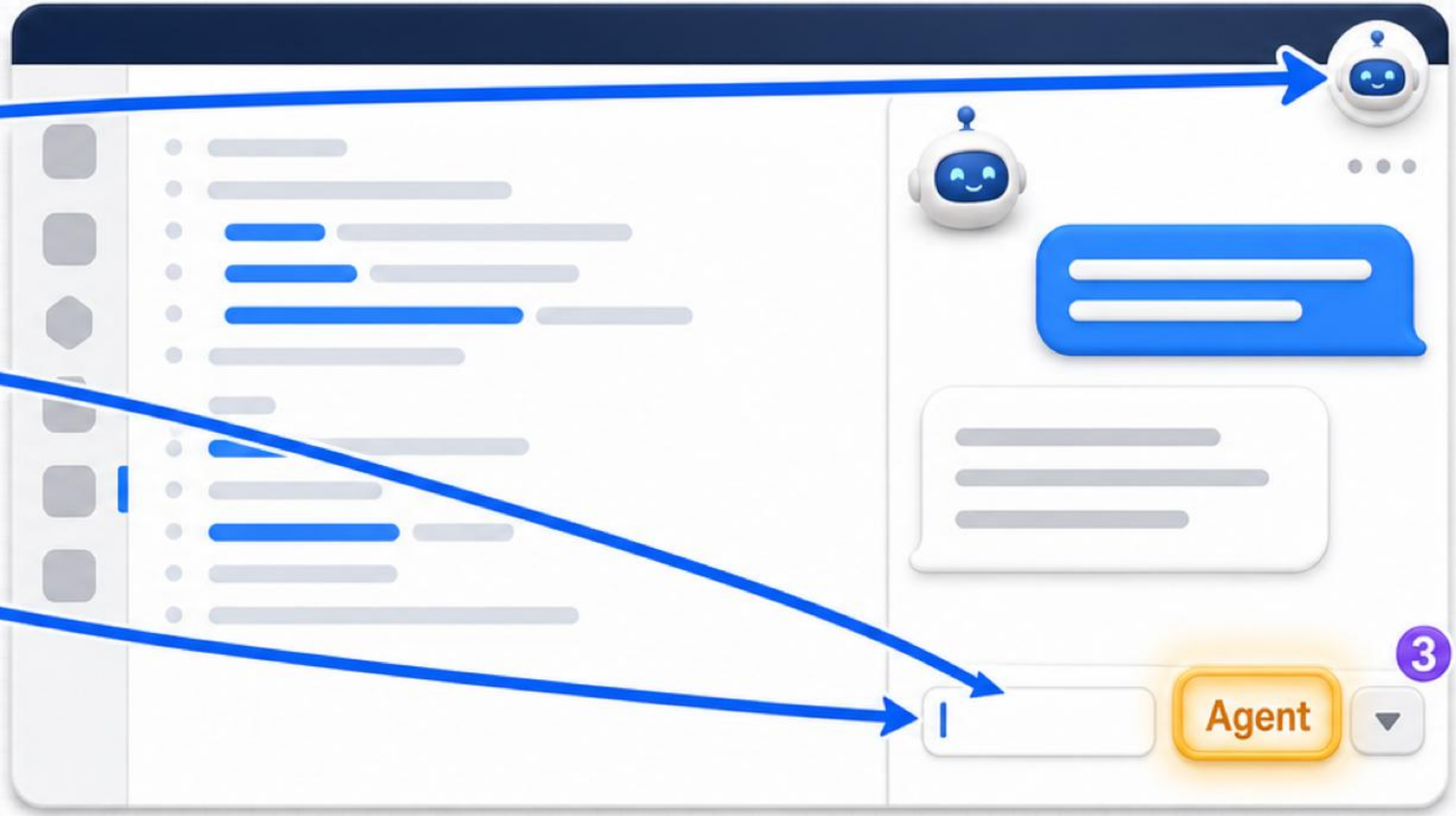
③ ブラウザが開く → 許可 → VS Code に戻る

成功したらこう見える: アカウントアイコンに自分の GitHub ユーザー名が表示される

※ Student Developer Pack で Copilot が有効になっているアカウントで入ること

Copilot の agent mode を確認する

1. ① 画面右上の Copilot アイコンから Chat を開く
2. ② チャット入力欄のモード切替を Agent にする
3. ③ 使うモデルを選ぶ (指定がなければ既定のままでよい)





✅ 成功したらこう見える: Agent と表示された状態で、日本語で送った質問に返事が来る

※ 応答が来ない・Copilot が有効でないと出る人は挙手

hello.py を Copilot に作らせて、自分で実行する



1. VS Code で先ほどの ai-workshop フォルダを開く (code . でもよい)
2. Copilot Chat (Agent) にそのまま送る:

   『hello.py というファイルを作って、Hello, AI駆動開発! と表示するプログラムを書いてください。』
3. 作られたら、自分でターミナルから実行:

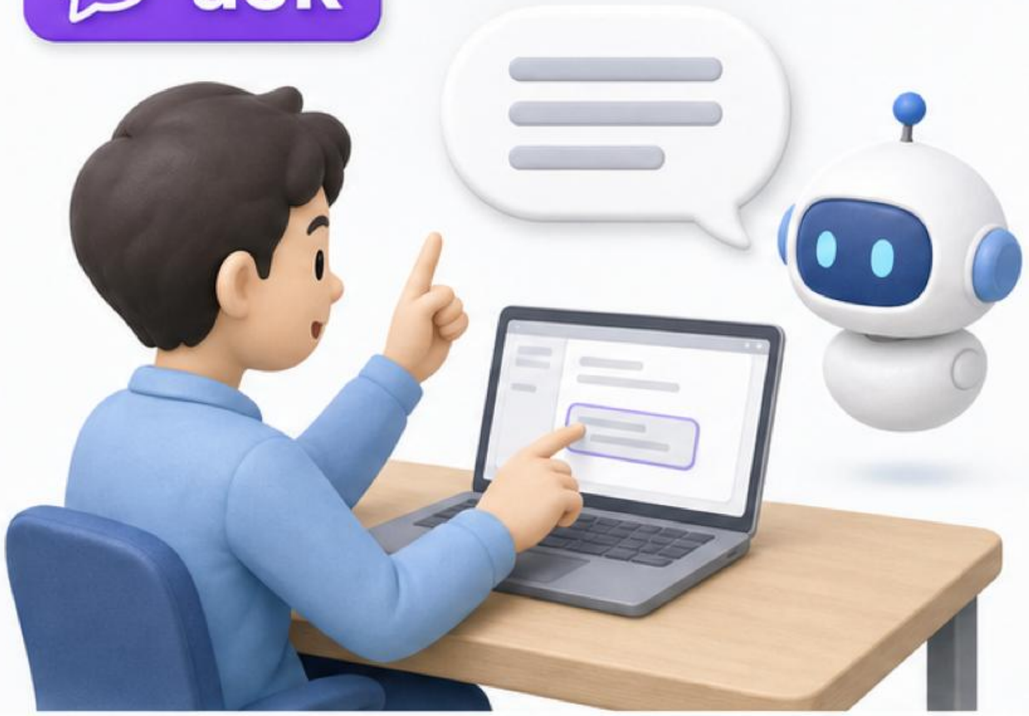
   ```
   uv run hello.py
   ```

成功したらこう見える:
ターミナルに Hello, AI駆動開発! が出る

→ チェックリスト ④ が埋まりました

agent mode と ask mode の違い

ask



ask — 聞く。答えが返るだけ。自分でコピーして貼る

agent



agent — 任せる。ファイルを読み、書き、実行し、結果を報告する

使い分け：『これは何?』『なぜ動かない?』は ask ／『作って』『直して』『動かして』は agent

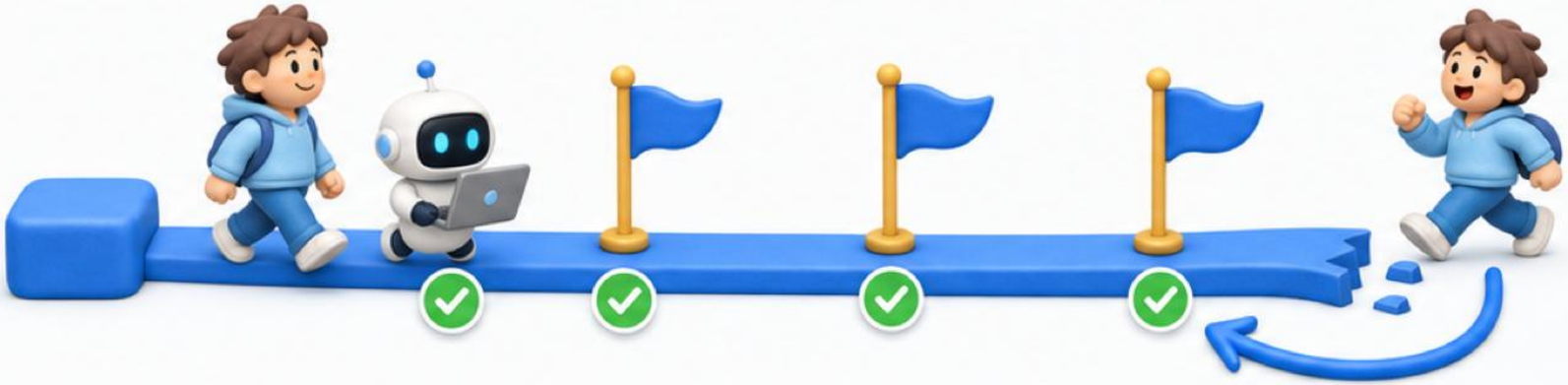
要点：agent は自分の手を動かす。だから、何をしたかを人が読む必要がある

なぜバージョン管理が必要か

AIは1時間で数百行書く。人はその全部を覚えていられない



『昨日は動いていたのに』が起きたとき、戻れるかどうか





Git = セーブポイントを作る道具



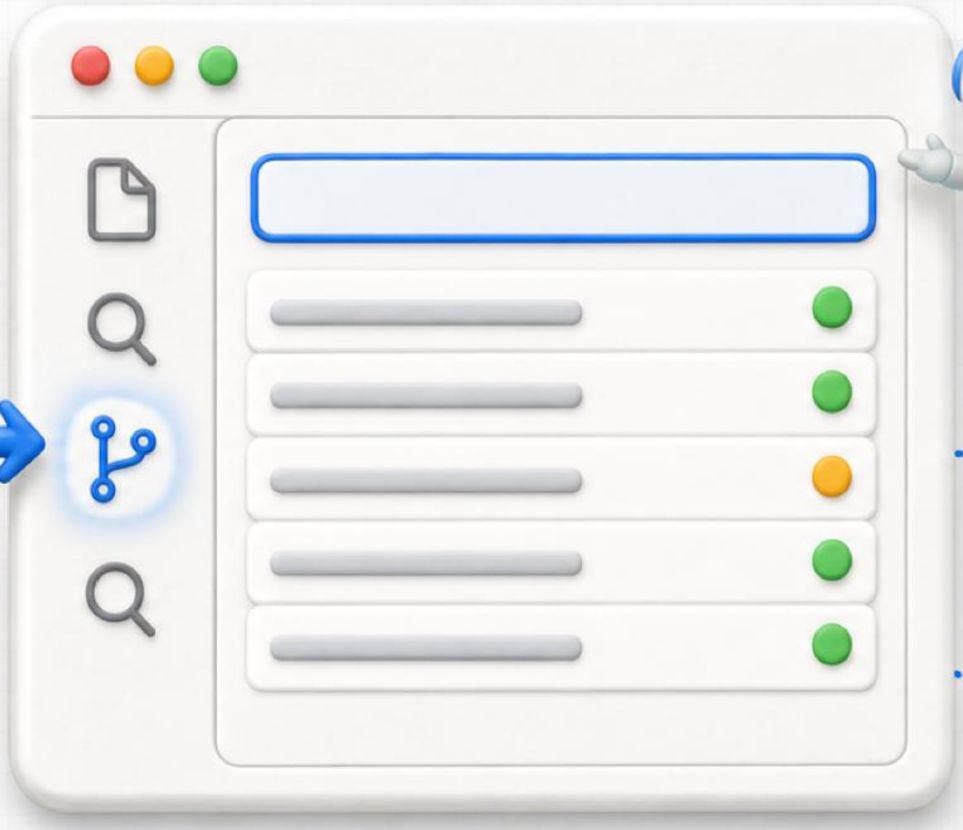
GitHub = セーブデータを外に置く場所（PCが壊れても残る）



要点：AI駆動開発でこそ、履歴が要る

VS Code でリポジトリを確認する







1. 左のサイドバーから「ソース管理」を開く

2. 変更されたファイルの一覧が並んでいる — uv init が Git の初期化を済ませてくれている

3. もし「リポジトリを初期化する」ボタンが出ていたら、それを押せば同じ状態になる



成功したらこう見える：ソース管理に変更ファイルが並んでいる

※ git init をターミナルで打つのと同じこと。今日は画面で進めます

変更を見て、コミットする

① ファイル名をクリック → 変更箇所が左右に並んで表示される（差分）

② 変更の横の ＋ を押して ステージ（コミットに含める印）



③ メッセージ欄に何をしたかを書く：最初のプロジェクトと hello.py を追加

④「コミット」ボタンを押す

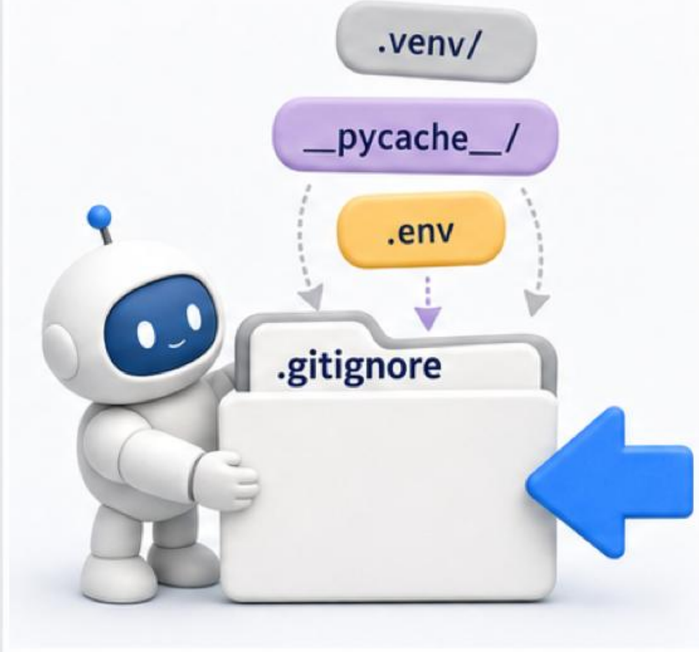
成功したらこう見える：変更ファイルの一覧が空になり、履歴に1件目が残る

要点：メッセージは日本語でよい。未来の自分が読んで分かることだけが条件

push する前の安全確認



① 何が入るかを見る ー コミット前に差分とファイル一覧を必ず確認



② 入れないものを決める ー .gitignore に書く



③ コードに書かない ー API キー・パスワード・個人情報・未公開の研究データ

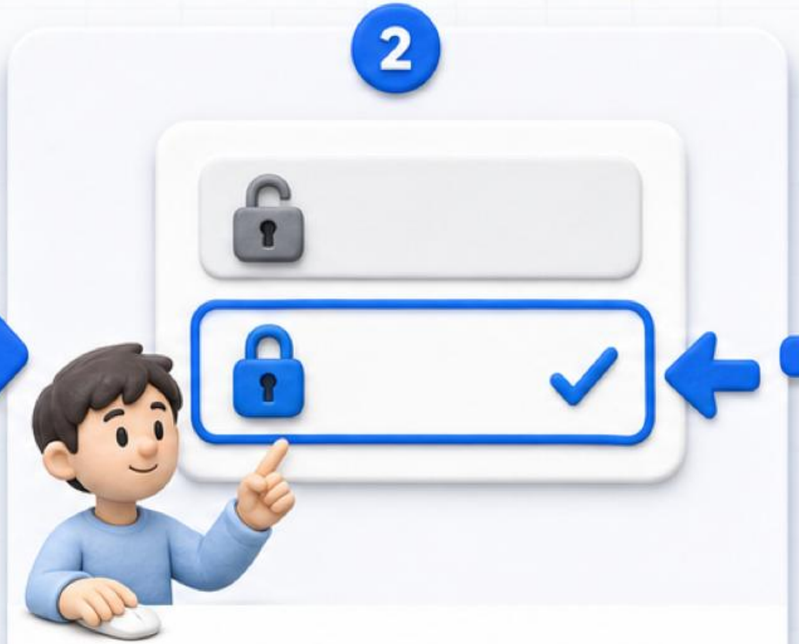
一度 push したものは、消しても履歴に残る

GitHub に private リポジトリを作って push する

前の枚の3点を確認してから進む



① ソース管理の「GitHub に公開 (Publish to GitHub)」を押す



② 公開先の選択で private を選ぶ ← こちらを選ぶ



③ ブラウザが開く → GitHub で許可 → VS Code に戻る



成功したらこう見える:「正常に公開されました」と出て、リポジトリを開くリンクが表示される

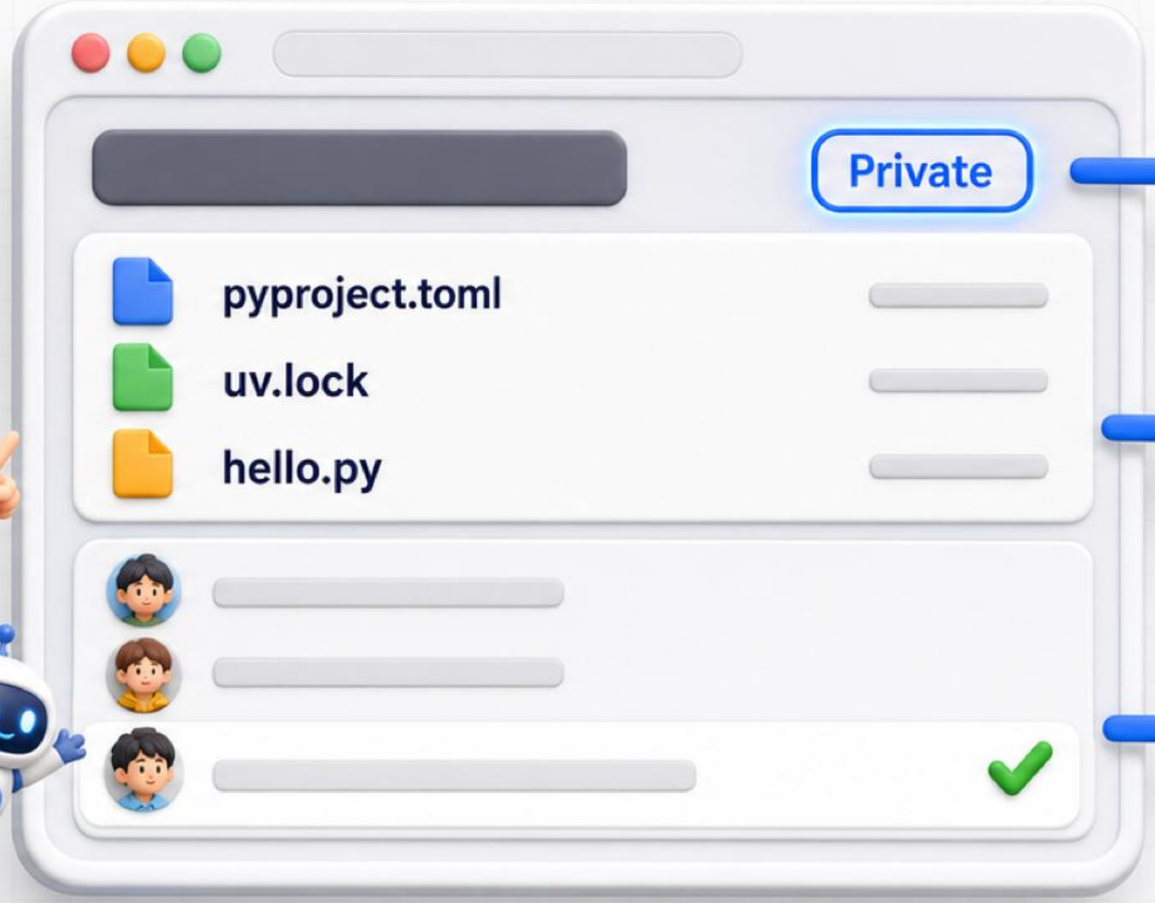
→ チェックリスト ⑤ が埋まりました

要点: 迷ったら必ず private。公開はあとから自分で切り替えられる

GitHub で自分のコードを確認する

—— ブラウザで自分のリポジトリを開く ——





1. 名前の横に
Private と付いているか

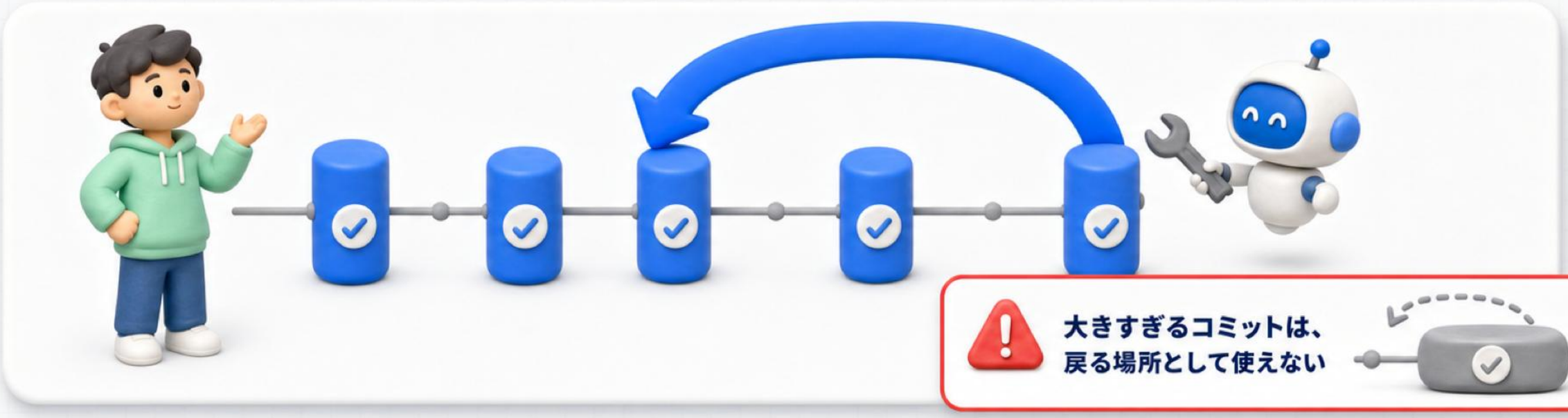
2. pyproject.toml
uv.lock hello.py
が並んでいるか

3. コミット履歴に
自分のメッセージが
残っているか

これが、あなたの最初のポートフォリオです

コミットの粒度: 動いたらコミット

迷ったらこれだけ — 動いたらコミット



大きすぎるコミットは、
戻る場所として使えない





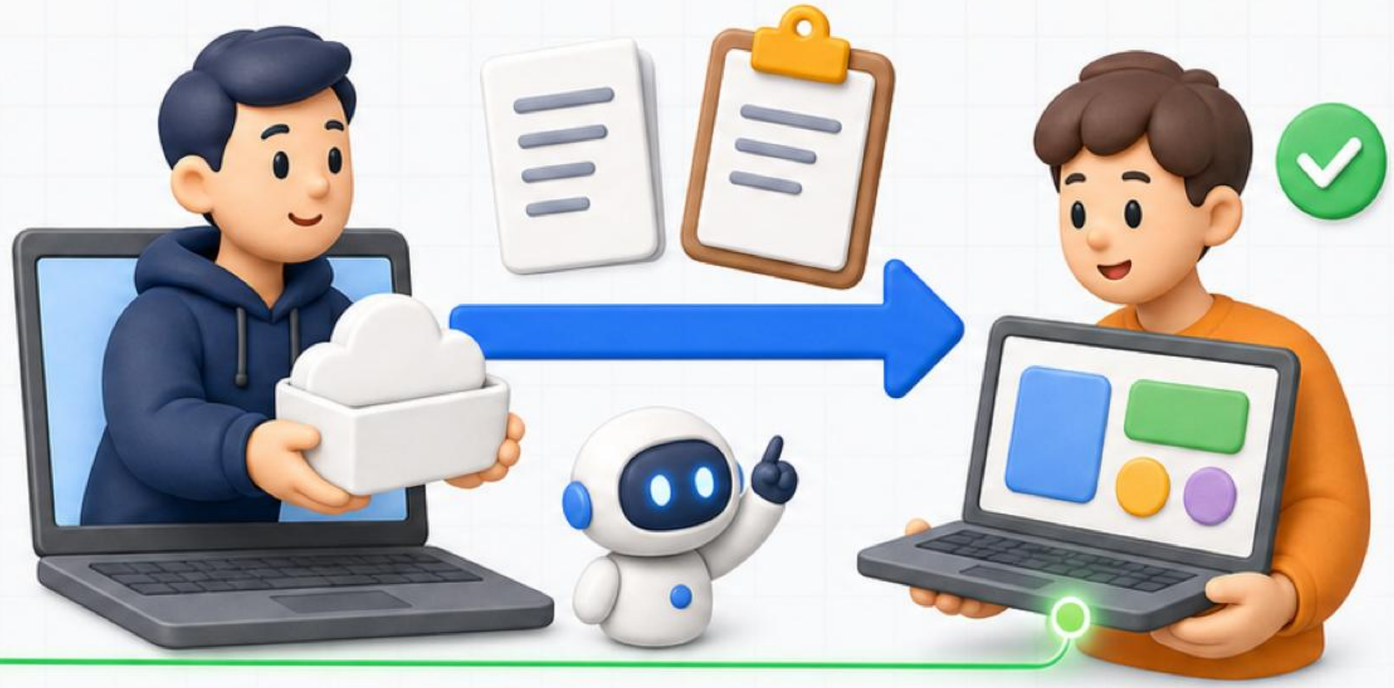


要点: AIに『直して』と頼む前にコミットしておく = いつでも引き返せる

配布リポジトリを clone して動かす

1. `git clone <当日配布URI>`
2. `cd <フォルダ名>`
3. `uv sync`
4. `uv run main.py`

成功したらこう見える：他人が作ったアプリが、自分のPCで動く



なぜ動くのか：**uv.lock** に
『どの部品のどの版か』が全部書いてあるから

→ チェックリスト 6 が埋まりました

環境チェックリスト、確認します

- [x] ① git --version が表示され、user.name と user.email が設定済み
- [x] ② uv --version が表示される
- [x] ③ uv run で自分のプロジェクトが動く
- [x] ④ VS Code で Copilot が agent mode で応答する
- [x] ⑤ GitHub の private リポジトリに push できた
- [x] ⑥ 配布リポジトリを uv sync で再現できた



詰まったときの3手



① エラーを丸ごとコピーして、
Copilot に貼って聞く ―
読めなくてよい、貼ればよい



② 隣の人と画面を見せ合う ―
同じ画面を2人で見ると
原因が見える



③ 3分詰まったら手を挙げる ―
TA が来ます

※ どうしてもインストールできないPCは、隣の人とペアで進めてOK

道具は揃いました。次は、いよいよ本格的な AI駆動開発へ