Givery, Inc.

HANDS-ON GUIDE DAY 1

ハンズオンガイド Day 1

生成AI実務研修（2 日間）
受講者用 Day 1 進行ガイド

提供	Givery 株式会社
配布先	協栄産業株式会社 御中
研修形式	7 時間 × 2 日間（連続開催）
版数	v2.0（2026 年 6 月 20 日）

本書は Day 1 の演習進行ガイドです。Claude Code の基礎操作・コマンド・Skills・Subagents・仕様駆動開発の入り口を、手を動かしながら順に押さえます。VS Code の統合ターミナルから Claude Code（CLI）を起動して進めます。演習はすべて個人ワークです。

1. 本日のゴール

Day 1 が終わるとき、受講者は次の状態に到達しています。座学（S01）の内容はオリエンテーション座学編を参照してください。本書は手を動かす S02 以降を扱います。

- `/init` `/context` `/compact` `/clear` と、モデル切替・Effort 操作を、何のために使うかを言葉にしながら使い分けられる
- `.claude/` の配置と、ユーザー設定（`~/.claude/`）とプロジェクト設定（`<repo>/.claude/`）の優先関係を説明できる
- Skills / Subagents / Commands / MCP / Hook の 5 部品が「それぞれ何のための道具か」を、テンプレを動かした実体験として区別できる
- 既存コマンドを動かしたうえで、自作スラッシュコマンドを 1 つ `.claude/commands/` に作って動かしている
- 3 段階の Skill（単機能 / Subagents 並列 / 画像アセット読込）を動かし、複雑な Skill の挙動を観察している
- Subagent を単体と「コマンド起点で 3 体並列」の 2 通りで動かしている
- 仕様駆動開発の考え方を理解し、Spec Kit で在庫アラート通知ツールの仕様化を 3 ステップ（`/specify` → `/plan` → `/tasks`）まで進めている

---

2. タイムテーブル [420min]

時間は分数のみで示します（昼休憩 [60min] を除く実働 [420min]）。S01 はオリエンテーション座学編を使った講義です。

枠	時間	内容
S01	[60min]	Claude Code の基礎と AI 最新動向（座学。座学編を投影）
S02	[60min]	Claude Code 基礎演習（ショートカット / モデル・Effort / 設定の階層 / 5 部品の違い）
S03	[60min]	環境セットアップ確認 + コマンド基礎演習（既存コマンド + 自作コマンド）
昼休憩	[60min]	–
S04	[70min]	Skills 基礎演習（単機能 / Subagents 並列 / 画像アセット読込の 3 段階）
S05	[50min]	Subagents 基礎演習（単体 / コマンド起点で 3 体並列）
S06	[90min]	仕様駆動開発の基礎 + Spec Kit ハンズオン（在庫アラート通知ツールを 3 ステップ）
S07	[30min]	質疑応答・振り返り・閉会

各演習の終わりに 30 秒、「ここまでで詰まった点」を研修チャットに 1 行投下してください。講師がリアルタイムで拾います。

---

3. 開始前のセットアップチェック

演習開始前に以下を確認してください。未完了がある場合は S01 の間に申告してください。詳細手順は別冊「事前セットアップガイド」を参照してください。

- 配布 ZIP（`kyos-handson.zip`）を解凍済みで、VS Code でフォルダ `kyos-handson/` を開いている
- VS Code の統合ターミナル（メニュー「ターミナル」→「新しいターミナル」）で `claude` が起動する
- `python --version` で Python 3.12 以降が表示される
- `python -m pytest --version` で pytest のバージョンが表示される
- `uvx --version` で uv が動く（S06 の Spec Kit で使います）

> **配布フォルダの地図**
>
> `kyos-handson/` の下に、本日使う演習フォルダ（`day1/s02_basics/` 〜 `day1/s05_subagents/`）、Spec Kit 用の `speckit-inventory/`、ヒント集 `hints/`、雛形 `templates/` が入っています。Day 2 用の `kiro-inventory/` `capstone-agent/` `_reference/` は本日は開きません。

---

4. S02：Claude Code 基礎演習 [60min]

道具の素性を、操作で体に入れるパートです。座学で聞いた語彙を、実際のキー操作と画面の変化に結びつけます。作業フォルダは `kyos-handson/day1/s02_basics/` です。

TRY IT 01　ショートカットで「会話の状態」を操る

学習目標　`/init` `/context` `/compact` `/clear` の 4 つを、目的を言いながら使い分ける
作業フォルダ　`kyos-handson/day1/s02_basics/`
形式　個人ワーク。VS Code 統合ターミナルで `claude` を起動して進めます

STEP 1 —— 実行 [5min]

`/init` で CLAUDE.md のたたき台を作る

作業フォルダで Claude Code を起動し、`/init` を実行します。Claude Code がフォルダの中身を調べ、`CLAUDE.md` の初版を提案します。生成された CLAUDE.md を開いて、何が書かれているか（プロジェクト概要、構成、コマンド類）を読んでください。

> **Tips**
> `/init` は「このプロジェクトの取扱説明書を Claude 自身に書かせる」コマンドです。中身は必ず人間が直します。空のプロジェクトでは情報が少ないので、提案も短くなります。

STEP 2 —— 観察 [5min]

`/context` でいま何を読んでいるかを見る

`/context` を実行し、現在のコンテキストの内訳（CLAUDE.md、会話履歴、読み込んだファイルなど）と消費トークンの目安を確認します。CLAUDE.md がコンテキストの先頭に固定で乗っていることを目で見てください。

STEP 3 —— 実行 [5min]

`/compact` と `/clear` の違いを体験する

適当な質問を 3 往復してから `/context` で増えたトークンを確認し、`/compact` を実行します。会話が要約に圧縮され、流れは保たれたままトークンが減ることを確認してください。次に `/clear` を実行し、履歴ごと消えて仕切り直しになることを確認します。

> **Tips**
> 同じ話題を続けるなら `/compact`、話題が変わるなら `/clear`。判断に迷ったら hints/s02_basics_hint.md の使い分け表を開いてください。

TRY IT 02　モデルと Effort をタスクに合わせて切り替える

学習目標　同じお題を別モデル・別 Effort で投げ、応答の粒度と速さの差を体感する
作業フォルダ　`kyos-handson/day1/s02_basics/`

STEP 1 —— 実行 [6min]

`/model` でモデルを切り替える

`/model` でモデル一覧を開き、Haiku 系に切り替えてから「このフォルダの構成を 3 行で説明して」と依頼します。次に Sonnet 系に切り替えて同じ依頼を投げ、応答の深さと速さを見比べてください。

STEP 2 —— 実行 [6min]

Effort（思考の深さ）を変える

同じモデルのまま、「軽く考えて答えて」と「じっくり手順を踏んで考えて」で同じ設計判断を 1 つ相談し、出力の丁寧さがどう変わるかを観察します。Effort をどこで上げ、どこで下げるとコストと品質の釣り合いが取れるかを 1 行メモしてください。

> **Tips**
> 単純な置換・要約は速いモデル、設計判断や複数ファイル横断は深いモデル。「迷ったら標準モデル」を基準に、重いときだけ上げるのが現場の運用です。

TRY IT 03　設定の置き場所と 5 部品の地図を作る

学習目標　ユーザー設定とプロジェクト設定の優先関係を確認し、5 部品（Skills / Subagents / Commands / MCP / Hook）の役割を自分の言葉で 1 枚に整理する

作業フォルダ　`kyos-handson/day1/s02_basics/`

作るファイル　`notes/parts_map.md`

STEP 1 —— 観察 [8min]

設定が重なる順を確認する

VS Code で `~/.claude/`（ユーザー設定）と作業フォルダの `.claude/`（プロジェクト設定）を両方開きます。Claude Code に次を聞いてください。

```
~/.claude/ のユーザー設定とこのプロジェクトの .claude/ 設定は、
同じ項目が衝突したときどちらが優先されますか。
このプロジェクトで実際に効いている設定がどこ由来かを一覧にしてください。
```

確認方法：ユーザー設定が全プロジェクト共通の土台で、プロジェクト設定がそれを上書きする関係になっていることを、Claude の回答と実ファイルの両方で確かめます。

STEP 2 —— 書く [14min]

5 部品の違いを 1 枚に整理する

Skills / Subagents / Commands / MCP / Hook の 5 つについて、「何のための道具か」「いつ発火するか」「どこに置くか」を 1 行ずつ、`notes/parts_map.md` に自分の言葉でまとめます。たたき台を Claude に出させてから、自分で削って整えてください。

```
Claude Code の Skills / Subagents / Commands / MCP / Hook の 5 つについて、
「役割」「発火するタイミング」「置き場所」を 1 行ずつの比較表にしてください。
公式ドキュメントの用語に合わせ、私が判断できる粒度で簡潔に。
```

> **この地図が午後の軸になる**
> S04 〜 S06 で、この表の各部品を順に実物で動かします。今この段階で「どれが手順の自動化（Commands）で、どれが並列の別働隊（Subagents）か」を区別できていると、午後が迷子になりません。整理の見本は hints/s02_basics_hint.md にあります。

早く終わったら

5 部品それぞれについて「協栄産業の自分の業務だと何に使えそうか」を 1 例ずつ書き足してください。

5. S03：環境セットアップ確認 + コマンド基礎演習 [60min]

前半は当日の実機セットアップ確認、後半はスラッシュコマンドの仕組みを「動かす → 自作する」の順で押さえます。作業フォルダは `kyos-handson/day1/s03_commands/` です。

5.1 環境セットアップ確認 [15min]

事前セットアップガイドの動作確認を当日環境で再点検します。次がすべて通ることを確認してください。詰まった項目はこの時間内に講師へ申告します。

```
claude --version
python --version
python -m pytest --version
uvx --version
```

続けて `kyos-handson/day1/s03_commands/` で Claude Code を起動し、「`hello.py` を作って実行し、`Hello, KyoS!` と表示して」と依頼します。ファイル作成と実行の許可を求められたら承認し、出力が出ることまで確認してください。ここまでで「読む・書く・実行する」がそろった状態になります。

5.2 コマンド基礎演習 [45min]

TRY IT 04　既存コマンドを動かして挙動を掴む

学習目標　配布済みの自作コマンド 2 本を動かし、コマンドの正体が「Markdown 1 枚」だと体感する

作業フォルダ　`kyos-handson/day1/s03_commands/`（`.claude/commands/` に `summarize-file.md` と `explain-error.md` が同梱）

STEP 1 —— 実行 [8min]

配布コマンドを呼ぶ

Claude Code で `/` を打ち、候補に `/summarize-file` と `/explain-error` が出ることを確認します。`/summarize-file src/sample.py` を実行し、引数で渡したファイルが要約されることを見てください。

STEP 2 —— 観察 [7min]

コマンドの中身を開く

`.claude/commands/summarize-file.md` を VS Code で開きます。中身が普通のプロンプト文で、`$ARGUMENTS` で引数を受け取っていることを確認してください。ファイル名がそのままコマンド名になる規則も押さえます。

> **Tips**
> コマンドは「いつも書いている長文プロンプトを 1 ファイルに保存したもの」です。難しい仕組みはありません。だからこそ自分の定型作業をどれだけ言語化できているかが効いてきます。

TRY IT 05　自分のコマンドを 1 本作る

学習目標　自分の定型作業を 1 つ選び、スラッシュコマンドとして登録して動かす

作るファイル　`.claude/commands/<name>.md`（例：`review-diff.md`）

ヒント　`hints/s03_commands_hint.md`

STEP 1 ── 考える [5min]

ワンキー化する作業を 1 つ決める

「毎回同じ指示を打っている作業」を 1 つ思い浮かべてください。例：変更差分をレビューして commit メッセージ案を 3 つ出す、エラーログを貼ると原因候補を絞る、関数にテスト雛形を生やす。引数で渡す対象（ファイル名など）も決めます。

STEP 2 ── 書く [15min]

コマンドファイルを作る

Claude Code に作らせても、自分で書いても構いません。Claude に頼む場合の指示例：

```
.claude/commands/review-diff.md を作ってください。
中身は「git の変更差分をレビューし、リスクの高い変更を上から指摘し、
commit メッセージ案を 3 つ出す」プロンプトにしてください。
対象パスは $ARGUMENTS で受け取れるようにしてください。
```

確認方法：`/` を打って候補に自作コマンド名が出ること、実行して期待どおりの動きが返ることを確認します。出てこない場合はファイルの置き場所（`.claude/commands/` 直下か）とファイル名を見直してください。

STEP 3 ── 答え合わせ [5min]

使い回せる形に整える

1 回実行してみて、毎回手で補足説明を足しているなら、その説明をコマンド本文に書き足します。「補足なしで一発で意図どおり動く」状態をゴールにしてください。

早く終わったら：引数を 2 つ受け取るコマンドや、複数行プロンプトのコマンドにも挑戦してください。作ったコマンドは Day 2 でもそのまま使えます。

---

6. S04：Skills 基礎演習（3 段階）[70min]

Skill は「呼ばれたら特定の手順とノウハウを再生する」フォルダ単位のパッケージです。配布済みのテンプレ Skill を、単純なものから複雑なものへ 3 段階で動かし、最後の段で挙動を観察します。作業フォルダは `kyos-handson/day1/s04_skills/` です。

TRY IT 06　段階 1：単機能の Skill を動かす

対象 Skill　`.claude/skills/meeting-summary/`（会議メモを「決定事項 / 宿題 / 未解決」の 3 区分に整形）
学習目標　Skill の最小形（SKILL.md 1 枚）の構造と発火を確認する

STEP 1 ── 実行 [8min]

Skill を呼ぶ

Claude Code で「`data/meeting_sample.md` を会議サマリ形式に整形して」と依頼します。`meeting-summary` Skill が読み込まれ、3 区分の Markdown が返ることを確認してください。`/meeting-summary` と打って手動発火もできます。

STEP 2 ── 観察 [7min]

SKILL.md を開く

`.claude/skills/meeting-summary/SKILL.md` を開き、frontmatter の `name` `description` と、本文に書かれた手順を読みます。description が「いつこの Skill を使うか」の判定材料になっていることを押さえてください。

TRY IT 07　段階 2：Subagents を並列で呼ぶ Skill を動かす

対象 Skill　`.claude/skills/multi-review/`（1 ファイルを設計・可読性・バグ観点の 3 体で並列レビューし、結果を 1 枚に集約）
学習目標　Skill が内部で Subagent を並列起動し、結果をまとめる動きを観察する

STEP 1 ── 実行 [10min]

並列レビューを走らせる

「`src/order.py` を multi-review でレビューして」と依頼します。3 つの観点が同時に走り、最後に統合レポートが返ります。実行中、複数の Subagent が並行して動いている表示を見てください。

STEP 2 ── 観察 [10min]

単機能 Skill との違いを掴む

SKILL.md を開き、本文が「3 観点を並列で起動して集約する」という段取りを書いていることを確認します。段階 1 との差は「自分で全部やる」か「別働隊に振って束ねる」かです。観点が独立しているほど並列が効くことをメモしてください。

> **Tips**
> 観点が互いに依存しない仕事（設計・セキュリティ・パフォーマンスのレビューなど）は並列向き。前の結果を次が使う仕事は並列にしても待ちが発生します。

TRY IT 08　段階 3：画像アセットを読み込む実用 Skill を動かす

対象 Skill　`.claude/skills/ui-from-mock/`（UI モック画像を読み、画面要素を構造化して HTML スケルトンを出す）
学習目標　補助ファイル（画像・テンプレ）を抱えた複雑な Skill の挙動を観察し、どこまで実用になるかを判断する
同梱素材　`assets/mock_screen.png`（架空の在庫管理画面のモック）

STEP 1 ── 実行 [12min]

画像から構造を起こす

「`assets/mock_screen.png` を ui-from-mock で構造化して」と依頼します。Skill が画像を読み、画面の要素（ヘッダー、テーブル、フィルタなど）を一覧にし、HTML スケルトンを出力します。

STEP 2 ── 観察 [13min]

フォルダ構成と限界を見る

Skill フォルダの中に SKILL.md だけでなく、参照テンプレートや変換手順が同梱されていることを確認します。出力された HTML を実際にブラウザで開き、「どこまで合っていて、どこから人間が直す必要があるか」を 1 行で書いてください。

> **3 段階で何を持ち帰るか**
>
> 段階 1 は「手順の再生装置」、段階 2 は「別働隊の指揮」、段階 3 は「素材を抱えた業務パッケージ」。Skill は SKILL.md 1 枚から、画像や補助スクリプトを束ねた実用ツールまで連続でスケールします。自分の業務のどの作業を、どの段階で Skill 化できそうかを考えながら触ってください。各 Skill の作り込みの読み解きは hints/s04_skills_hint.md にあります。

---

7. S05：Subagents 基礎演習 [50min]

Subagent は親とは独立したコンテキストで動く専任エージェントです。文脈を切り分けながら並列化できるのが要点です。配布済みのテンプレを単体・並列の 2 通りで動かします。作業フォルダは `kyos-handson/day1/s05_subagents/` です。

TRY IT 09　単体の Subagent を呼ぶ

対象　組み込みの `code-reviewer` と、配布の自作 `.claude/agents/spec-checker.md`

学習目標　Subagent が親と別コンテキストで動くことを、レビューの切れ味で体感する

STEP 1 ── 実行 [10min]

レビュー専任に振る

「`src/order.py` を code-reviewer サブエージェントでレビューして、指摘を重要度順に並べて」と依頼します。実装の文脈に引きずられない、観点の独立した指摘が返ることを確認してください。

STEP 2 ── 観察 [8min]

自作エージェントの定義を読む

`.claude/agents/spec-checker.md` を開き、責務（何を見て、何を見ないか）が書かれていることを確認します。Subagent は「役割を絞った別人格」だと捉えると設計しやすくなります。

TRY IT 10　コマンド起点で 3 体を並列起動する

対象　配布コマンド `/triple-review`（設計・セキュリティ・パフォーマンスの 3 体を並列起動して集約）

学習目標　1 つのコマンドから複数 Subagent が並列で動き、結果が束ねられる流れを掴む

ヒント　`hints/s05_subagents_hint.md`

STEP 1 ── 実行 [12min]

3 体を同時に走らせる

`/triple-review src/order.py` を実行します。3 つの観点が並列で走り、最後に観点別の指摘がまとまった 1 枚のレポートが返ります。3 体が同時進行している表示を見てください。

STEP 2 ── 観察 [12min]

コマンドと Subagent の役割分担を見る

`.claude/commands/triple-review.md` を開き、「どの観点を、どの Subagent に、何ファイルに対して並列で投げるか」が書かれていることを確認します。コマンドが指揮、Subagent が実働という分担です。逐次で 3 回レビューする場合と比べて待ち時間がどう違うかを 1 行メモしてください。

早く終わったら：`triple-review.md` に 4 体目の観点（例：テスト網羅）を足して、自分の `.claude/agents/` に対応するエージェント定義を 1 つ作ってください。Day 2 の本題材でそのまま再利用できます。

8. S06：仕様駆動開発の基礎 + Spec Kit ハンズオン [90min]

8.1 仕様駆動開発とは [15min]

「いきなりコードを書かせる」と、要求の解釈ずれがコードの形になってから発覚し、修正のたびに別の場所が壊れます。仕様駆動開発（Spec-Driven Development）は、この手戻りをコードの前の文書で潰す方法論です。流れは **Spec（何を作るか）→ Plan（どう作るか）→ Tasks（実行単位に割る）→ Implement（実装・検証）** の 4 段。詳しい考え方はオリエンテーション座学編の該当セクションを参照してください。ここでは Spec Kit を実機で 3 段まで動かします。

Spec Kit（GitHub）と Kiro 式（AWS）の設計思想は対照的です。Spec Kit はテンプレートとコマンドを柔軟にカスタムする思想、Kiro 式は `requirements.md` / `design.md` / `tasks.md` の 3 ファイル固定で迷いを減らす思想です。本研修では Day 1 で Spec Kit を、Day 2 で Kiro 式を、同じ要件で動かして比較します。

出典：github.com/github/spec-kit ／ kiro.dev/docs/specs

8.2 Spec Kit ハンズオン [75min]

HANDS-ON 11　在庫アラート通知ツールを Spec Kit で仕様化する

学習目標	議事メモから `/specify` → `/plan` → `/tasks` の 3 段を回し、コードの前に仕様・計画・タスクを固める流れを体験する
作業フォルダ	`kyos-handson/speckit-inventory/`（`uvx specify init` 済み。`docs/meeting-notes.md` 同梱）
題材	架空案件「在庫アラート通知ツール」の議事メモ（A4 2 枚相当、ダミーデータ）
到達点	spec.md / plan.md / tasks.md の 3 ファイルが揃った状態（実装は Day 2）
ヒント	`hints/s06_speckit_hint.md`

STEP 0 ── 準備 [10min]

議事メモを読み、作るものを 1 行にする

`docs/meeting-notes.md` を自分の目で 3 分読み、「結局なにを作る話か」を 1 行で言語化してください。閾値・通知頻度・通知先など、メモに書かれていない箇所（未確定事項）も拾っておきます。Claude Code は `speckit-inventory/` で起動します。

STEP 1 ── 実行 [25min]

`/specify` で仕様を起こす

議事メモを `/specify` に渡し、仕様を生成します。AI への指示例：

```
/specify docs/meeting-notes.md の内容をもとに、在庫アラート通知ツールの仕様を作成してください。
スコープは「在庫データの監視」「閾値判定」「通知メッセージ生成」の 3 機能とし、
実際の通知送信はモック（送信内容を標準出力に出す）で代替します。
議事メモに書かれていない要件は勝手に追加せず、未確定事項として最後にまとめてください。
```

確認方法：生成された spec を必ず通読し、Step 0 で拾った未確定事項が「勝手に埋められて」いないかを確認します。埋まっていたら「この項目は議事メモに根拠がありません。未確定事項に戻してください」と差し戻します。

> **Tips**
> `/specify` が見つからない場合は、`speckit-inventory/` で Claude Code を起動し直してください。`.specify/` がカレントフォルダ配下にないとコマンドが読み込まれません。

STEP 2 —— 実行 [20min]

`/plan` で作り方を決める

`/plan` を実行し、spec から技術選定・モジュール構成・データ構造を導出させます。生成された plan に「なぜその構成か」の理由が書かれているかを見てください。理由が書けていない選定は、たいてい雰囲気で選ばれています。

確認方法：3 機能（監視・閾値判定・通知生成）が plan の中で別々のモジュールに割れているかを確認します。

STEP 3 —— 実行 [20min]

`/tasks` で実行単位に割る

`/tasks` を実行し、plan を検証可能な粒度のタスク列に分解させます。確認方法は次の 2 点です。タスクの先頭グループが「テスト作成」になっているか（実装が先頭なら並べ替えを依頼）、1 タスクが 15 分以内に終わるサイズに割れているか。

到達点の確認：`specs/` 配下に spec.md / plan.md / tasks.md の 3 ファイルが揃っていれば本日分は完了です。実装（`/implement`）は Day 2 の冒頭で続けます。

早く終わったら：tasks のうち「通知メッセージ生成」をさらに 2 つに分割できないか検討し、分割案を tasks に反映してください。

---

9. 完了チェックと成果物

Day 1 終了時点で、自分の作業フォルダが次の状態になっていることを確認してください。

- `day1/s02_basics/notes/parts_map.md` に 5 部品の整理表がある
- `day1/s03_commands/.claude/commands/` に自作コマンドが 1 つ以上ある
- 3 段階の Skill（meeting-summary / multi-review / ui-from-mock）を動かし、挙動を観察した
- Subagent を単体と「コマンド起点 3 体並列（`/triple-review`）」の 2 通りで動かした
- `speckit-inventory/specs/` に spec.md / plan.md / tasks.md の 3 ファイルが揃っている

終了時に研修チャットへ次の 3 行を投下してください。

- 5 部品のうち、自分の業務で一番効きそうなものとその理由 1 行
- 一番手応えがあった演習 1 つ
- 一番苦戦した演習とその原因 1 行

> **Day 2 への宿題**
>
> Day 2 のメインテーマ「技術営業支援 AI エージェント」を各自の業務に寄せて使うため、自分の業務で「AI に任せたい定型作業」を 1 つ持参してください。1 行のメモで十分です。

---

10. よくある質問

Q1：`/init` が短い CLAUDE.md しか作らない

空に近いプロジェクトでは情報が少ないため当然です。中身は人間が育てるものなので、役割・規約・禁止事項を自分で書き足してください。

Q2：自作コマンドが `/` の候補に出てこない

置き場所がずれているか、ファイル名に問題があります。`.claude/commands/` 直下に `<name>.md` があるかを確認してください。サブフォルダに入れた場合は名前空間が変わります。

Q3：Skill が読み込まれない

frontmatter の YAML 構文（コロンの後の半角スペース、インデント）と、ファイル名が `SKILL.md`（大文字）であることを確認してください。description が曖昧だと自動発火しにくいので、手動発火（`/<skill-name>`）でまず動作を確かめます。

Q4：並列レビューが遅い／途中で止まる

並列で走る Subagent が多いほど時間はかかります。まず単体（TRY IT 09）が動くことを確認してから並列（TRY IT 10）に進んでください。途中で止まったら `Esc` で中断し、対象ファイルを 1 つに絞って再実行します。

Q5：Spec Kit の `/specify` が見つからない

`speckit-inventory/` で Claude Code を起動し直してください。`.specify/` がカレント配下にないとコマンドが読み込まれません。

---

11. 詰まったときの即応マニュアル

症状	確認	対処
`claude` が起動しない	Windows は `where claude`、macOS は `which claude`	VS Code とターミナルを閉じて開き直す。それでもダメなら講師に申告
モデル切替が反映されない	`/model` で現在のモデル表示	切替後に短い依頼を 1 回投げて挙動を確認。セッション再起動で確実に反映される
Skill が一覧に出ない	SKILL.md の frontmatter と置き場所	YAML 構文を直し、`/<skill-name>` で手動発火を試す
Subagent が動かない	`.claude/agents/<name>.md` の有無と frontmatter	組み込みの `code-reviewer` でまず単体動作を確認してから自作に進む
`uvx specify init` が失敗する	`uvx --version`、プロキシ環境変数	事前セットアップガイド §6 のプロキシ設定を確認。配布済みの初期化フォルダ `speckit-inventory/` で進める
Claude Code が意図しないファイルを編集し始めた	許可プロンプトの対象パス	`Esc` で中断し、「○○ 配下のみ編集可、それ以外は禁止」と範囲を明記して再依頼

文書末尾。