Referencia de lo que corrimos en el taller, herramienta por herramienta. Cada paso es un mensaje dentro de Claude Code. Los comandos que empiezan con `/` escriben sus propios archivos. La deliberación de stack es un mensaje normal, sin `/`, y termina en "no escribas archivos" para que sea una discusión y no una acción.

Nota de nombres: en spec-kit los comandos van con guion (`/speckit-constitution`). En versiones previas iban con punto (`/speckit.constitution`). Es lo mismo, confirma con el autocompletado dentro de Claude Code.

Prerrequisitos

- **spec-kit** necesita `uv`, Python 3.11+ y Git.
- **cc-sdd** solo necesita Node.js (usa `npx`).
- Claude Code instalado y logueado.

```
mkdir cafe-speckit cafe-ccsdd
```

---

spec-kit

Terminal

```
uv tool install specify-cli --from git+https://github.com/github/spec-kit.git@v0.12.2
specify self check
cd cafe-speckit
specify init . --integration claude
claude
```

Si `--integration claude` da error de nombre, corre `specify integration list` y usa el nombre exacto que salga.

Flujo (un mensaje por paso, dentro de Claude Code)

1. Constitución

```
/speckit-constitution Sistema de control de ventas para un café pequeño operado por personal no técnico. Presupuesto bajo, sin equipo de TI ni mantenimiento continuo. Crea principios enfocados en: simplicidad sobre completitud (cada feature debe justificarse); integridad de los datos de venta (una venta registrada es inmutable; las correcciones son contra-ventas, nunca ediciones); operación sin conexión (la caja no puede detenerse si se cae el internet); durabilidad de los datos local-first (los datos viven en el navegador, así que el almacenamiento no se asume permanente: debe existir una estrategia explícita de respaldo/exportación ante borrado del navegador); validación estricta de toda entrada del cajero (nunca un total negativo ni un producto vacío); testing proporcional que priorice la lógica de cálculo de totales sobre la UI. Incluye gobernanza sobre cómo estos principios guían las decisiones técnicas posteriores.
```

2. Especificación (sin stack)

```
/speckit-specify El cajero registra cada venta con sus productos, cantidades y categorías. El sistema calcula totales de ventas por día y por mes, identifica el producto más vendido en un período dado, y muestra el desglose de ventas por categoría. El personal no es técnico, así que la interacción debe ser mínima y a prueba de errores. No describas tecnología, base de datos ni frameworks todavía.
```

3. Clarificación

```
/speckit-clarify
```

Cuando pregunte, responde: devoluciones, corte diario y zona horaria, si hay varias cajas, y cómo se carga el catálogo de productos.

4. Deliberación de stack (mensaje normal, sin `/`)

```
Ya decidí el enfoque general: webapp local-first como PWA, sin backend, datos en el navegador, funciona offline. Dentro de ese enfoque dame 2-3 opciones con trade-offs: (a) framework de UI (vanilla JS vs. algo ligero tipo Preact/Svelte vs. React); (b) capa de datos (IndexedDB directo vs. wrapper tipo Dexie vs. una solución local-first tipo RxDB). Evalúa cada una frente al principio de simplicidad y al de durabilidad de datos. No recomiendes una sola ni escribas archivos.
```

5. Plan

```
/speckit-plan Webapp local-first como PWA instalable. Funciona completamente offline mediante service worker. Los datos se almacenan localmente en el navegador (IndexedDB). Sin servidor ni backend. Los reportes se exportan a CSV para la contadora externa. Incluye una estrategia explícita de respaldo/exportación de datos, coherente con el principio de durabilidad local-first. Ajusta framework y capa de datos a la decisión tomada en la deliberación.
```

6. Consistencia, tareas e implementación

```
/speckit-analyze
```

```
/speckit-tasks
```

```
/speckit-implement
```

Los artefactos quedan en `specs/001-.../` (spec.md, plan.md, tasks.md) y la constitución en `.specify/memory/constitution.md`.

---

cc-sdd

Terminal

```
cd cafe-ccsdd
git init
npx cc-sdd@latest --claude-skills --lang es
claude
```

Para ver qué escribiría antes de aplicar, agrega `--dry-run`.

Flujo (un mensaje por paso, dentro de Claude Code)

1. Discovery

```
/kiro-discovery Sistema de control de ventas para un café pequeño: registrar ventas con productos, cantidades y categorías; reportes de totales por día y por mes; producto más vendido en un período; desglose de ventas por categoría. Personal no técnico, debe funcionar sin conexión, presupuesto bajo, sin equipo de mantenimiento. Enfoque previsto: webapp local-first (PWA), sin backend, datos en el navegador.
```

2. Init de la spec

```
/kiro-spec-init ventas-cafe
```

3. Requisitos (formato EARS)

```
/kiro-spec-requirements ventas-cafe
```

Revisa `requirements.md`: los "El sistema deberá...", "Cuando... deberá...", "Si... entonces deberá...".

4. Clarificación manual (cc-sdd no tiene comando, se usa la compuerta)

```
Antes de continuar al diseño: revisa requirements.md, lista ambigüedades, supuestos y casos borde sin cubrir, y pregúntame lo que necesites resolver. No apruebes aún.
```

5. Deliberación de stack (mensaje normal, sin `/`)

```
Enfoque decidido: webapp local-first (PWA), sin backend, datos en el navegador, offline. Dame 2-3 opciones dentro de ese enfoque (framework de UI y capa de datos/IndexedDB) con trade-offs frente a simplicidad y durabilidad de datos. No escribas archivos.
```

6. Diseño, tareas e implementación

```
/kiro-spec-design ventas-cafe
```

En `design.md`, mira los diagramas y el File Structure Plan.

```
/kiro-spec-tasks ventas-cafe
```

```
/kiro-impl ventas-cafe
```

`/kiro-impl` corre una tarea por vuelta: implementa con la prueba primero y un revisor aparte valida. Es seguro re-correrlo si lo interrumpes.

Dónde escribe cada herramienta

- spec-kit: `.specify/` y `.claude/commands/`.
- cc-sdd: `.claude/skills/` y `.kiro/`.

En carpetas separadas no chocan.