IACADEMY

DE FUNDAMENTOS A ARQUITECTURA DE IA

MÓDULO 03

Tu primer workflow

Fundamentos

iacedemy.com — 2026

MÓDULO 03

Tu primer workflow

Nivel: Fundamentos

Autor: Ricardo Gutierrez

Publicación: Mayo 2026

Plataforma: iacedemy.com

Instalar Claude Code

Claude Code es la CLI oficial de Anthropic. No es un chat en el navegador: es un **agente que trabaja en tu terminal**. Lee archivos, ejecuta comandos, crea codigo, hace commits. Es como tener un desarrollador senior a tu lado.

Requisitos

- **Node.js 18+**: descargalo de nodejs.org si no lo tienes.
- **Plan Anthropic** que incluya Claude Code (Max o Team).
- **Terminal**: cualquiera sirve (Terminal de Mac, Windows Terminal, iTerm2, Warp).

Instalacion

```
 # Paso 1: Instalar Claude Code globalmente
npm install -g @anthropic-ai/claude-code

# Paso 2: Verificar la instalacion
claude --version

# Paso 3: Iniciar tu primera sesion
claude
```

La primera vez te pedira autenticarte. Se abre un enlace en tu navegador, inicias sesion con tu cuenta de Anthropic, y queda configurado.

[IMG: Captura de terminal mostrando la instalacion de Claude Code y la pantalla de bienvenida]

Troubleshooting

Si algo falla, verifica estos 3 puntos:

1. `node -v` debe mostrar v18 o superior. Si es inferior, actualiza Node.
2. `npm -v` debe funcionar sin errores. Si no, reinstala npm.
3. Tu plan de Anthropic debe incluir Claude Code. Verificalo en console.anthropic.com.

El 90% de los errores de instalacion son por versiones antiguas de Node.js.

Montar el directorio .claude/

El directorio `.claude/` es donde vive la configuracion de Claude Code para tu proyecto. Sin el, Claude Code no sabe nada de ti ni de tu proyecto. Con el, se convierte en un asistente personalizado.

Estructura basica

```
mi-proyecto/
  .claude/
    CLAUDE.md          # Instrucciones del proyecto (obligatorio)
    settings.json      # Configuracion tecnica: hooks, permisos, MCP
    commands/           # Comandos reutilizables (/)
      review.md        # /review - code review automatico
      deploy.md        # /deploy - checklist de deploy
    docs/               # Documentacion accesible para Claude Code
      api-spec.md      # Especificacion de API
      style-guide.md   # Guia de estilo
    memory/             # Memoria persistente entre sesiones
      decisions.md     # Decisiones de arquitectura
      patterns.md      # Patrones frecuentes del equipo
```

Crear la estructura

```
 # Crear todos los directorios de golpe
mkdir -p .claude/commands .claude/docs .claude/memory

# Crear los archivos base
touch .claude/CLAUDE.md
touch .claude/settings.json
```

Cada directorio tiene un proposito:

- **CLAUDE.md**: Lo mas importante. Es tu README para la IA.
- **settings.json**: Hooks, permisos, servidores MCP.
- **commands/**: Prompts guardados que invocas con `/`.
- **docs/**: Documentacion que Claude Code puede leer bajo demanda.
- **memory/**: Notas que persisten entre sesiones.

CLAUDE.md: el cerebro de tu proyecto

CLAUDE.md es el archivo mas importante de todo el directorio `.claude/`. Es donde le dices a Claude Code quien eres, que proyecto es este, que stack usas y que reglas debe seguir.

Estructura recomendada

```
 # Nombre del Proyecto

## Stack
- Backend: FastAPI (Python 3.11)
- Frontend: Next.js 14 (React, TypeScript)
- DB: Supabase PostgreSQL con RLS
- Deploy: Cloudflare Pages (frontend) + Hetzner (backend)

## Reglas de codigo
- Espanol para comments, ingles para variables y funciones
- Tests obligatorios para toda funcion publica
- Nunca hardcodear secretos (usar variables de entorno)
- Pydantic para validacion de inputs en todos los endpoints
- Typing estricto en Python (mypy compatible)

## Convenciones de naming
- Endpoints: /api/v1/{recurso} (kebab-case)
- Funciones Python: snake_case
- Componentes React: PascalCase
- Variables de entorno: UPPER_SNAKE_CASE

## Errores comunes a evitar
- NO usar print() para logging (usar structlog)
- NO hacer queries SQL sin parametrizar
- NO commitear archivos .env
- NO usar any en TypeScript

## Contexto de negocio
- SaaS B2B para pymes del sector [X]
- Multi-tenant: cada cliente tiene sus datos aislados via RLS
- 3 roles: admin, editor, viewer
```

> **CLAUDE.md es acumulativo**
>
> Claude Code lee el CLAUDE.md del proyecto actual, pero tambien puede leer un CLAUDE.md global (en `~/.claude/CLAUDE.md`). El global tiene tus reglas personales que aplican a todos los proyectos. El del proyecto tiene reglas especificas.

Tip avanzado: cuanto mas especifico sea tu CLAUDE.md, mejores resultados. "Tests obligatorios" es vago. "Pytest con fixtures en conftest.py, coverage minimo 80%, sin mocks de base de datos" es especifico y produce resultados consistentes.

Tu primer hook

Un hook es una accion automatica que se dispara cuando ocurre un evento en Claude Code. Es como un guardia que vigila sin que tengas que estar pendiente.

Tipos de hooks

- **PreCommit**: Se ejecuta antes de cada commit. Perfecto para tests y lint.
- **PostCommit**: Despues del commit. Para notificaciones, deploys.
- **PostFileWrite**: Al crear o modificar un archivo. Para formateo automatico.
- **PreToolUse**: Antes de que Claude Code use cualquier herramienta. Para bloquear operaciones.

Configurar tu primer hook

Edita `.claude/settings.json`:

```
{
 "hooks": {
   "PreCommit": [
     {
       "command": "python -m pytest tests/ -x --tb=short",
       "description": "Ejecutar tests antes de cada commit"
     }
   ],
   "PostFileWrite": [
```

```
        {
          "command": "ruff format $FILE",
          "description": "Formatear archivos Python automaticamente"
        }
      ]
    }
}
```

Ahora, cada vez que Claude Code intente hacer un commit:

1. Se ejecutan los tests automaticamente.
2. Si pasan, el commit se hace.
3. Si fallan, el commit se cancela y ves el error.

Y cada vez que crea o modifica un archivo Python, lo formatea con `ruff` automaticamente.

[IMG: Captura de terminal mostrando un hook PreCommit ejecutando tests antes de permitir el commit]

> **Mi experiencia:** El hook de PreCommit con tests me ha salvado de subir bugs a produccion docenas de veces. Claude Code a veces genera codigo que parece correcto pero tiene edge cases. El hook los atrapa antes de que lleguen al repo.

Tu primer agente

Un agente no es un prompt. Un prompt te da una respuesta. Un agente **itera**: analiza, ejecuta, verifica, corrige y vuelve a intentar hasta completar la tarea.

Prompt vs Agente

- **Prompt:** "Escribe un test para la funcion calcular_precio()". Resultado: 1 archivo con tests.
- **Agente:** "Analiza src/pricing.py, identifica las funciones sin tests, crea los tests, ejecutalos y corrige los que fallen." Resultado: N archivos, N iteraciones, tests pasando.

Ejecutar tu primer agente

```
 # Agente simple: crear tests para funciones sin cobertura
claude "Analiza todos los archivos Python en src/.
Para cada funcion publica que no tenga test en tests/:
1. Crea un test con al menos 2 casos (happy path y edge case)
2. Ejecuta pytest para verificar
3. Si falla, corrige el test o reporta el bug
Formato de salida: tabla [Funcion | Test creado | Resultado]"
```

Lo que hace Claude Code con esta instruccion:

1. Lee la estructura de `src/` para entender el proyecto.
2. Lee cada archivo Python e identifica funciones publicas.
3. Cruza con `tests/` para saber cuales ya tienen tests.
4. Para las que no tienen: crea un archivo de test.
5. Ejecuta `pytest` para verificar.
6. Si un test falla, analiza el error y corrige.
7. Repite hasta que todos pasen.

Esto puede ser 5 pasos o 50, dependiendo del proyecto. El agente decide cuantas iteraciones necesita.

Tips para agentes efectivos

1. **Se especifico en la tarea.** "Mejora el codigo" es vago. "Anade tipado estricto a todas las funciones de src/api/" es concreto.
2. **Define el criterio de exito.** "Hasta que todos los tests pasen" o "Hasta que ruff check devuelva 0 errores".
3. **Limita el scope.** Un agente que toca 3 archivos es mas fiable que uno que toca 30.
4. **Pide que verifique.** "Despues de cada cambio, ejecuta los tests para confirmar que no rompiste nada."

Tu primer comando reutilizable

Los comandos son prompts guardados que invocas con `/`. Son el equivalente a funciones en programacion: escribes una vez, reutilizas siempre.

Crear un comando

Crea un archivo en `.claude/commands/`. El nombre del archivo es el nombre del comando.

Archivo: `.claude/commands/review.md`

```
 Revisa el codigo modificado en el ultimo commit.

Para cada archivo modificado:
1. Identifica bugs potenciales o edge cases no manejados
2. Sugiere mejoras de rendimiento si las hay
3. Verifica que sigue las convenciones definidas en CLAUDE.md
4. Comprueba que tiene tests (o sugiere crearlos)

Formato de salida:
| Archivo | Issue | Severidad | Sugerencia |
|---------|-------|-----------|------------|

Reglas:
- Solo issues reales, no nitpicking estetico
- Severidad: critica (bug), alta (seguridad), media (mejora), baja (estilo)
- Si no hay issues en un archivo, no lo incluyas en la tabla
```

Ahora en Claude Code solo escribes `/review` y se ejecuta. Sin reescribir el prompt cada vez.

Mas comandos utiles

Archivo: `.claude/commands/explain.md`

```
 Explica el archivo que estoy viendo como si fuera un nuevo
miembro del equipo. Incluye:
1. Que hace este archivo (1 parrafo)
2. Dependencias principales
3. Funciones/clases clave y su proposito
4. Gotchas o cosas no obvias
No mas de 300 palabras.
```

Archivo: `.claude/commands/test.md`

```
Genera tests para el archivo que estoy viendo.
- Usa pytest con fixtures
- Incluye: happy path, edge cases, errores esperados
- Naming: test_{funcion}_{escenario}
- Si la funcion tiene dependencias externas, usa mocks
- Ejecuta los tests al final para verificar
```

Archivo: `.claude/commands/refactor.md`

```
Refactoriza el archivo actual siguiendo estas prioridades:
1. Extraer funciones duplicadas
2. Anadir tipado estricto donde falte
3. Simplificar condicionales complejos
4. Mejorar nombres de variables/funciones

Reglas:
- No cambiar la interfaz publica (mismas funciones, mismos parametros)
- Ejecutar tests despues de cada cambio para no romper nada
- Si un cambio es arriesgado, pregunta antes de hacerlo
```

Comandos son compartibles

Si tu `.claude/` esta en el repositorio del proyecto, todos los miembros del equipo tienen acceso a los mismos comandos. Estandarizas como se usa la IA en tu organizacion.

Ejercicio practico

Ejercicio M03: Monta tu primer workflow con Claude Code

1. Instala Claude Code (`npm install -g @anthropic-ai/claude-code`).
2. Crea un proyecto nuevo o usa uno existente.
3. Monta la estructura `.claude/` completa: CLAUDE.md, settings.json, commands/.

4. Escribe un CLAUDE.md con al menos: stack, reglas de codigo, convenciones de naming.
5. Configura 1 hook (sugerido: PreCommit con tests o lint).
6. Ejecuta tu primer agente: "Crea un endpoint GET /health que devuelva status y timestamp".
7. Crea al menos 1 comando reutilizable en `.claude/commands/`.
8. Usa el comando y verifica que funciona.

Bonus: Crea un comando `/onboard` que explique todo el proyecto a un nuevo miembro del equipo leyendo CLAUDE.md, la estructura de directorios y los principales archivos.

Conclusiones clave

Key takeaways del M03

1. Claude Code se instala con `npm install -g @anthropic-ai/claude-code`. Requiere Node.js 18+ y plan Anthropic.
2. El directorio `.claude/` transforma a Claude Code de chat generico a asistente personalizado para tu proyecto.
3. CLAUDE.md es el archivo mas importante. Cuanto mas especifico, mejores resultados.
4. Hooks automatizan calidad: tests antes de commit, formateo al crear archivos, validacion antes de operaciones.
5. Los agentes iteran (analizar, ejecutar, verificar, corregir). Un prompt da 1 respuesta; un agente completa la tarea.
6. Los comandos (`/`) son prompts reutilizables y compartibles. Escribes una vez, usas siempre.

Siguiente: M04 - Automatizacion con hooks y agents

Ahora que tienes Claude Code configurado, el siguiente paso es escalar: hooks avanzados, agentes con 6 partes y el patron Coordinator + Workers.

Ir al Modulo 04

IACADEMY

iacedemy.com

De fundamentos a arquitectura de IA.
12 módulos prácticos. 24 recursos descargables.
Quizzes con certificado. Vídeos profesionales.

Empieza gratis en iacedemy.com/free