

Odoo 20

con Claude Code

Buenas prácticas para programar
módulos de Odoo con inteligencia
artificial: del primer CLAUDE.md a un
flujo de equipo profesional



```
~/odoo20-claude - claude

> Planifica el módulo cc_obras: modelo, vistas,
seguridad y pruebas. Sigue el patrón de Odoo 20.

● Explorando odoo/addons/project ...
● Plan: 5 archivos · 3 pruebas · 1 regla multiempresa
✓ ./odoo-bin --test-tags /cc_obras    3 passed
✓ pre-commit run                      all checks ok

> █
```

Odoo 20 con Claude Code

Buenas prácticas para programar módulos de Odoo con un agente de IA, del primer CLAUDE.md a un flujo de equipo completo

Autor: Marlon Falcón Hernández

ERP, CRM y Software · Valencia, España · marlonfalcon.com

Edición para Odoo 20.0 · Octubre de 2026

Continuación de *Odoo 20 para Nuevos Programadores* y *OWL para Odoo 20*

Índice

Introducción

Qué es Claude Code y cómo encaja en el desarrollo con Odoo 20

Capítulo 0 · Instalar Claude Code y preparar un repositorio Odoo 20

Parte I · El flujo de trabajo

Capítulo 1 · CLAUDE.md: enseñar Odoo 20 al agente

Capítulo 2 · Tu primer módulo con Claude Code

Capítulo 3 · Modo plan, prompts y contexto

Capítulo 4 · Permisos y seguridad

Introducción

Este libro enseña a programar módulos de Odoo 20 trabajando junto a Claude Code, el agente de programación de Anthropic que vive en tu terminal y en tu editor. Es la tercera entrega de la serie: en *Odoo 20 para Nuevos Programadores* aprendiste modelos, vistas e informes; en *OWL para Odoo 20*, el cliente web. Aquí aprenderás a hacer todo eso más rápido sin perder calidad.

La idea central es sencilla: Claude Code escribe código muy deprisa, pero solo escribe *buen* código Odoo si tú le das contexto, límites y una forma de comprobar su trabajo. Cada capítulo construye una de esas tres cosas.

Cada capítulo sigue el mismo formato que los libros anteriores: un ejercicio real que se construye por pasos, una prueba para comprobar que funciona, el resultado que verás en Odoo 20 y un reto para practicar solo.

A quién va dirigido

A quien ya sabe crear un módulo sencillo de Odoo (manifiesto, modelo, vista, seguridad y menú) y quiere usar un agente de IA de forma profesional. No hace falta experiencia previa con Claude Code: el Capítulo 0 empieza desde la instalación.

También sirve a responsables técnicos de partners de Odoo que quieren fijar reglas comunes para que todo el equipo use el agente igual.

Requisitos

Componente	Versión recomendada	Para qué
Odoo	20.0 Community o Enterprise, desde el código fuente	Ejecutar módulos y pruebas
Python	La que pida Odoo 20	Servidor de Odoo y pruebas
PostgreSQL	La que pida Odoo 20	Base de datos de desarrollo
Git	Cualquiera actual	Control de versiones y puntos de retorno
Claude Code	La última disponible	El agente de programación
Cuenta de Claude	Plan Pro, Max, Team o Enterprise, o clave de API	Autenticar Claude Code
Editor	VS Code o JetBrains (opcional)	Ver los cambios del agente junto al código

A diferencia del libro de OWL, aquí **sí conviene tener Odoo instalado desde el código fuente** (o en Docker con el código montado): Claude Code necesita poder leer el núcleo de Odoo para copiar sus patrones y ejecutar `odoo-bin` para lanzar las pruebas.

Preparar el entorno

1. Instala Odoo 20 como en el capítulo *Instalar Odoo 20 en Windows y Mac* del primer libro.
2. Crea un repositorio Git llamado `odoo20-claude` con una carpeta `addons/` para tus módulos.
3. Crea una base de datos solo para desarrollo, por ejemplo `dev20`. Nunca conectes Claude Code a una base de datos de producción.
4. Instala Claude Code siguiendo el Capítulo 0.

Cómo usar cada ejercicio

Cada capítulo trabaja sobre un módulo con el prefijo `cc_` (por ejemplo `cc_cap02_obras`). La diferencia con los libros anteriores es que **no copias el código a mano**: le das a Claude Code el prompt del capítulo, revisas lo que propone y compruebas el resultado. El código que aparece en el libro es la referencia de lo que debería producir.

Si el agente genera algo distinto, no pasa nada: compáralo con la referencia y usa las reglas de buenas prácticas de cada capítulo para decidir si lo aceptas.

Anatomía de un repositorio Odoo preparado para Claude Code

```
odoo20-claude/
├── CLAUDE.md           # memoria del proyecto: reglas para el agente
├── .claude/
│   ├── settings.json  # permisos y hooks compartidos con el equipo
│   ├── commands/      # comandos /propios (capítulo 11)
│   ├── agents/        # subagentes revisores (capítulo 12)
│   └── skills/         # habilidades reutilizables (capítulo 11)
├── .pre-commit-config.yaml # ruff, pylint-odoo, prettier (capítulo 8)
├── odoo.conf          # configuración de desarrollo
├── addons/
│   └── cc_cap02_obras/
│       ├── CLAUDE.md  # reglas solo de este módulo (opcional)
│       ├── __manifest__.py
│       ├── models/
│       ├── views/
│       ├── security/
│       └── tests/
```

La carpeta `.claude/` y el `CLAUDE.md` se versionan en Git: son parte del proyecto igual que el código.

Cómo leer este libro

Los prompts que escribes a Claude Code aparecen en bloques de código con el lenguaje `text` y empiezan por `>`. Los comandos de terminal aparecen como `bash`. Cuando una práctica de Odoo cambió respecto a versiones anteriores lo verás marcado como **(cambio desde Odoo 17)** o la versión que corresponda.

Claude Code evoluciona deprisa. Los conceptos del libro (memoria, permisos, modo plan, hooks, subagentes, MCP) son estables; si un atajo o una opción cambia de nombre, consulta `/help` dentro de Claude Code o la [documentación oficial](#).

Qué es Claude Code y cómo encaja en el desarrollo con Odoo 20

Claude Code es un agente: no solo sugiere código, sino que lee tu repositorio, edita archivos, ejecuta comandos en la terminal y comprueba el resultado. Se parece a trabajar con un programador junior muy rápido que nunca se cansa, pero que no conoce tu proyecto hasta que se lo explicas.

Por qué Odoo necesita reglas propias

- **Odoo tiene convenciones muy fuertes:** nombres de archivos, orden del manifiesto, prefijos de ids XML, `ir.model.access.csv`. Un modelo de IA genérico las conoce a medias.
- **Mucho código de ejemplo en internet es de versiones antiguas:** `attrs`, `<tree>`, `name_get`, `_sql_constraints` u `odoo.define`. Sin contexto, el agente puede mezclarlos con código de Odoo 20.
- **El núcleo de Odoo es la mejor documentación:** si el agente puede leer `odoo/addons/sale` o `odoo/addons/project`, copiará patrones reales de la versión 20 en lugar de inventarlos.
- **Los errores salen tarde:** un fallo de seguridad o de rendimiento no se ve hasta que hay usuarios y datos. Por eso las pruebas y la revisión son obligatorias.

Las piezas de Claude Code

Pieza	Qué es	Dónde la usarás
Sesión	Una conversación con el agente dentro de tu repositorio, iniciada con <code>cClaude</code>	En todos los capítulos
Contexto	Todo lo que el agente tiene «en la cabeza»: tus mensajes, archivos leídos, salidas de comandos	Capítulo 3
<code>CLAUDE.md</code>	Memoria del proyecto que se carga al empezar cada sesión	Capítulo 1
Modo plan	Modo en el que el agente investiga y propone un plan sin tocar archivos	Capítulos 2 y 3
Permisos	Reglas <code>allow</code> , <code>ask</code> y <code>deny</code> sobre herramientas y comandos	Capítulo 4
Hooks	Comandos que se ejecutan solos antes o después de cada acción del agente	Capítulo 8
Skills y comandos	Instrucciones reutilizables que invocas con <code>/nombre</code>	Capítulo 11
Subagentes	Agentes especializados con su propio contexto, por ejemplo un revisor	Capítulo 12
MCP	Conectores a herramientas externas: bases de datos, GitHub, navegador	Capítulo 13
Modo headless	Ejecutar Claude Code sin interfaz con <code>cClaude -p</code> , por ejemplo en CI	Capítulo 15

El ciclo de trabajo

Todo el libro gira en torno a cuatro fases que se repiten en cada tarea:

1. **Explorar:** el agente lee el código relevante (tu módulo y el núcleo de Odoo) sin escribir nada.

2. **Planificar:** propone un plan con archivos, modelos, vistas y pruebas. Tú lo corriges antes de que escriba.
3. **Codificar:** implementa el plan en pasos pequeños.
4. **Verificar:** ejecuta las pruebas de Odoo, el linter y una instalación limpia del módulo. Si algo falla, vuelve al paso 3.

Saltarse las dos primeras fases es el error más caro: el agente escribe rápido código que resuelve el problema equivocado.

Qué hace bien y qué no

Lo hace muy bien	Necesita tu supervisión
Crear el esqueleto de un módulo, vistas y seguridad	Decidir el modelo de datos de un proceso de negocio
Escribir pruebas a partir de un comportamiento descrito	Saber qué debe pasar en casos límite de tu cliente
Leer el núcleo de Odoo y explicarte cómo funciona algo	Distinguir código de Odoo 20 de código de versiones viejas
Corregir errores a partir de un traceback	Rendimiento con miles de registros reales
Migrar sintaxis antigua a la de Odoo 20	Seguridad: <code>sudo()</code> , reglas de registro, datos de otras compañías

Las cinco reglas de oro

1. **Contexto antes que código:** un buen `CLAUDE.md` vale más que cien prompts largos.
2. **Plan antes que edición:** para todo lo que toque más de un archivo, pide un plan primero.
3. **Pruebas como contrato:** si no hay una prueba que falle antes y pase después, el cambio no está terminado.
4. **Commits pequeños:** un commit por paso para poder volver atrás sin miedo.
5. **Tú firmas el código:** todo lo que el agente escribe es tuyo. Revísalo como revisarías el de un compañero.

Capítulo 0 · Instalar Claude Code y preparar un repositorio Odoo 20

Al terminar este capítulo tendrás Claude Code instalado, un repositorio Git con Odoo 20 accesible para el agente y una primera conversación en la que el agente te explica tu propio entorno.

Paso 1 · Instala Claude Code

El instalador nativo es el método recomendado y no necesita Node.js.

En macOS, Linux o WSL:

```
curl -fsSL https://claude.ai/install.sh | bash
```

En Windows (PowerShell):

```
irm https://claude.ai/install.ps1 | iex
```

Comprueba la instalación:

```
claude --version  
claude doctor
```

Si ya lo tenías instalado con `npm install -g @anthropic-ai/claude-code`, ese método está obsoleto: ejecuta `claude install` para pasar a la versión nativa. Las instalaciones nativas se actualizan solas.

En Windows, si tu Odoo corre en WSL o Docker, instala Claude Code **dentro de WSL**: así el agente usa la misma terminal y las mismas rutas que Odoo.

Paso 2 · Inicia sesión

```
cd ~/odoo20-claude  
claude
```

La primera vez se abre el navegador para que inicies sesión con tu cuenta de Claude (Pro, Max, Team o Enterprise) o con una clave de la consola de Anthropic. Después verás el prompt de Claude Code en la terminal.

Paso 3 · Organiza el repositorio

```
mkdir -p ~/odoo20-claude/addons
cd ~/odoo20-claude
git init
```

El agente debe poder leer el código de Odoo, pero **no debe modificarlo**. Hay dos formas habituales:

Opción	Cómo	Cuándo
Odoo fuera del repo	Arranca con <code>claude --add-dir ~/src/odoo</code> para darle acceso de lectura a otra carpeta	Recomendada: tu repo solo contiene tus módulos
Odoo como submódulo	<code>git submodule add -b 20.0 https://github.com/odoo/odoo.git odoo</code>	Cuando todo el equipo debe tener la misma versión exacta

En ambos casos, en el Capítulo 4 bloquearás la escritura sobre el núcleo con una regla `deny`.

Paso 4 · Configura Odoo para desarrollo

Crea `odoo.conf` en la raíz del repositorio:

```
[options]
addons_path = /ruta/a/odoo/addons,/ruta/a/odoo20-claude/addons
db_name = dev20
dbfilter = ^dev20$
list_db = False
admin_passwd = cambia-esto-en-local
log_level = info
```

Y comprueba que arranca y que las pruebas funcionan con un módulo del núcleo:

```
./odoo-bin -c odoo.conf -d dev20 -i base --stop-after-init
./odoo-bin -c odoo.conf -d dev20 --test-tags /base:TestORM --stop-after-init
```

Estos dos comandos son los que el agente usará en todo el libro: instalar o actualizar un módulo y lanzar sus pruebas.

Paso 5 · Tu primera conversación: que el agente explore

Antes de pedir código, pide comprensión. Escribe en Claude Code:

```
> Explora este repositorio y la carpeta de Odoo que tienes añadida.  
> No modifiques nada. Dime: qué versión de Odoo es, qué addons_path  
> usa odoo.conf, cómo se lanzan las pruebas de un módulo y qué módulos  
> propios hay en addons/.
```

El agente leerá archivos y quizá te pida permiso para ejecutar algún comando. Lee cada petición antes de aceptarla.

Paso 6 · Comandos básicos de la sesión

Comando o tecla	Para qué
<code>/help</code>	Ver todos los comandos disponibles en tu versión
<code>/init</code>	Generar un primer <code>CLAUDE.md</code> analizando el repositorio
<code>/memory</code>	Abrir y editar los archivos de memoria
<code>/clear</code>	Empezar de cero sin el contexto de la tarea anterior
<code>/compact</code>	Resumir la conversación para liberar contexto
<code>/permissions</code>	Ver y cambiar reglas de permisos
<code>/model</code>	Elegir el modelo
<code>Shift+Tab</code>	Cambiar de modo: normal, aceptar ediciones, modo plan
<code>Esc</code>	Interrumpir al agente en cualquier momento
<code>Esc</code> dos veces	Volver a un punto anterior de la conversación

Prueba

1. `claude --version` muestra una versión y `claude doctor` no da errores.
2. Los dos comandos de `odoo-bin` del paso 4 terminan sin errores.
3. En la conversación del paso 5, el agente identifica correctamente la versión 20.0 y el comando de pruebas.

Resultado en Odoo 20

Todavía no has creado nada visible en Odoo: la base de datos `dev20` tiene solo el módulo `base`. Lo importante es que el agente ya sabe dónde está Odoo, cómo arrancarlo y cómo lanzar pruebas.

Reto

Pide al agente que te explique, sin modificar nada, cómo funciona el método `create` de `models.py` en Odoo 20 y qué hace el decorador `@api.model_create_multi`. Comprueba su explicación abriendo tú mismo el archivo.

Parte I · El flujo de trabajo

Capítulo 1 · CLAUDE.md: enseñar Odoo 20 al agente

Vas a escribir el `CLAUDE.md` de tu repositorio: el archivo que Claude Code lee al empezar cada sesión. Es la práctica que más mejora la calidad del código que genera el agente.

Antes de empezar · Los niveles de memoria

Archivo	Alcance	Qué poner
<code>~/.claude/CLAUDE.md</code>	Todos tus proyectos	Tus preferencias personales: idioma, estilo de commits
<code>CLAUDE.md</code> en la raíz del repo	Todo el equipo, versionado en Git	Versión de Odoo, comandos, convenciones
<code>addons/<módulo>/CLAUDE.md</code>	Solo cuando el agente trabaja en ese módulo	Reglas de negocio de ese módulo

Un `CLAUDE.md` puede importar otros archivos con una línea `@ruta/al/archivo.md`. Así mantienes el principal corto y cargas el detalle solo cuando hace falta.

Paso 1 · Genera un borrador con `/init`

Abre Claude Code en la raíz del repositorio y escribe `/init`. El agente analiza el proyecto y propone un `CLAUDE.md`. Úsalo como punto de partida, no como versión final: no sabe todavía qué prácticas de Odoo son obligatorias para ti.

Paso 2 · Escribe las secciones que importan

Sustituye el borrador por algo así:

```
# Proyecto Odoo 20 · Implemento BIM

## Entorno
- Odoo 20.0. El código del núcleo está en ../odoo (solo lectura).
- Nuestros módulos están en addons/. Prefijo de nombre técnico: cc_
- Base de datos de desarrollo: dev20. Nunca uses otra.

## Comandos
- Instalar o actualizar: ./odoo-bin -c odoo.conf -d dev20 -u <módulo> --stop-after-init
- Pruebas de un módulo: ./odoo-bin -c odoo.conf -d dev20 --test-tags /<módulo> --stop-after-init
- Linter: pre-commit run --files <archivos>

## Reglas de Odoo 20 (obligatorias)
- Versión del manifiesto: 20.0.x.y.z. Licencia LGPL-3 salvo que se indique otra.
- Vistas de lista con <list>, nunca <tree>.
- Visibilidad con invisible="...", readonly="...", required="..."; nunca attrs ni states.
- display_name con _compute_display_name; nunca name_get.
- Restricciones SQL con models.Constraint; nunca _sql_constraints.
- create siempre con @api.model_create_multi.
- SQL crudo solo con odoo.tools.SQL y solo si el ORM no basta.
- Todo modelo nuevo tiene su línea en security/ir.model.access.csv.
- Campos relacionales multiempresa con check_company=True.
- No uses sudo() salvo que lo pida el plan y quede comentado el motivo.

## Forma de trabajar
- Antes de escribir, busca un ejemplo equivalente en el núcleo de Odoo 20 y síguelo.
- Toda funcionalidad nueva lleva pruebas en tests/ (TransactionCase).
- Un cambio no está terminado hasta que pasan las pruebas y el linter.
- Commits pequeños, en español, con formato: [MOD] módulo: descripción.

## Detalle
@docs/convenciones-odoo.md
```

Cada línea es una instrucción concreta y comprobable. Evita frases vagas como «escribe código limpio»: el agente no sabe qué significan para ti.

Paso 3 · Lo que no debe ir en CLAUDE.md

- **Secretos:** contraseñas, tokens o cadenas de conexión. El archivo está en Git.
- **Documentación larga:** si pasa de unas 200 líneas, muévela a archivos importados con @.
- **Reglas que ya impone una herramienta:** si ruff ordena los imports, no lo repitas aquí. Las herramientas se cumplen siempre; las instrucciones, casi siempre.
- **Historia del proyecto:** lo que pasó hace meses confunde más que ayuda.

Paso 4 · Un CLAUDE.md por módulo

Para un módulo con reglas de negocio propias, crea addons/cc_cap02_obras/CLAUDE.md :

```
# cc_cap02_obras
- Una obra pertenece a una sola compañía y a un solo cliente.
- El presupuesto de una obra no puede ser negativo.
- Los estados son: borrador, en curso, terminada, cancelada.
- Una obra terminada no se puede editar salvo por el grupo Responsable de obras.
```

El agente lo lee cuando trabaja dentro de esa carpeta, así que no ocupa contexto en otras tareas.

Paso 5 · Mantén la memoria viva

Cada vez que corrijas al agente por lo mismo dos veces, añade una línea al `CLAUDE.md`. Puedes editarlo con `/memory` o pedirle al propio agente que lo actualice:

```
> Añade a CLAUDE.md la regla: los campos Monetary siempre llevan
> currency_field explícito.
```

Revisa el cambio en el diff y haz commit como con cualquier otro archivo.

Prueba

1. Ejecuta `/clear` para empezar una sesión limpia.
2. Pregunta: `> ¿Cómo debería declarar una vista de lista y una restricción de unicidad en este proyecto?`
3. El agente debe responder con `<list>` y `models.Constraint` sin que se lo recuerdes.

Resultado en Odoo 20

Nada cambia todavía en la pantalla de Odoo. El cambio está en el agente: a partir de ahora cada sesión empieza sabiendo qué versión usas, cómo probar y qué sintaxis está prohibida.

Reto

Crea `docs/convenciones-odoo.md` con las reglas de nombres de tu equipo: ids de vistas (`view_<modelo>_form`), acciones (`action_<modelo>`), menús (`menu_<modelo>`) y grupos. Impórtalo desde `CLAUDE.md` y comprueba con una pregunta que el agente lo usa.

Capítulo 2 · Tu primer módulo con Claude Code

Vas a crear `cc_cap02_obras`, un módulo para registrar obras de construcción con cliente, presupuesto, coste real y estados. Lo importante no es el módulo, sino recorrer por primera vez las cuatro fases del ciclo: explorar, planificar, codificar y verificar.

Antes de empezar · Una tarea, una sesión

Empieza cada tarea nueva con `/clear`. El contexto de la tarea anterior (archivos leídos, errores ya resueltos) solo estorba y hace que el agente se confunda o mezcle cosas.

Paso 1 · Explorar

```
> Vamos a crear un módulo nuevo de obras. Antes de escribir nada,  
> busca en el núcleo de Odoo 20 un módulo con un modelo sencillo que  
> tenga estados, chatter, campos Monetary y multiempresa. Dime cuál es  
> y qué patrones deberíamos copiar. No modifiques archivos.
```

Un agente bien orientado volverá con un ejemplo real del núcleo y te señalará cómo declara `currency_id`, `company_id`, el `header` con botones y el `chatter`. Si te propone sintaxis antigua, tu `CLAUDE.md` necesita más detalle.

Paso 2 · Planificar

Pulsa `Shift+Tab` hasta ver el modo plan y escribe:

```
> Planifica el módulo cc_cap02_obras según addons/cc_cap02_obras/CLAUDE.md:  
> modelo cc.obra con nombre, código único por compañía, cliente,  
> compañía, fecha de inicio, presupuesto, coste real, desviación  
> calculada y almacenada, y estados con botones. Vistas de lista,  
> formulario y búsqueda, menú, permisos y pruebas. Lista los archivos  
> que vas a crear y qué prueba cubre cada regla.
```

Lee el plan con calma. Es el momento más barato para corregir: cambiar una línea del plan cuesta segundos; cambiar diez archivos, minutos. Si falta algo, díselo antes de aprobar.

Paso 3 · Codificar

Aprueba el plan y deja que el agente escriba. El resultado debería parecerse a esta referencia.

`__manifest__.py`:

```
{
  "name": "Obras (Cap 2)",
  "version": "20.0.1.0.0",
  "category": "Construction",
  "summary": "Registro de obras con presupuesto y desviación",
  "depends": ["mail"],
  "data": [
    "security/ir.model.access.csv",
    "views/cc_obra_views.xml",
    "views/menus.xml",
  ],
  "license": "LGPL-3",
}
```

models/cc_obra.py:

```

from odoo import api, fields, models
from odoo.exceptions import ValidationError

class CcObra(models.Model):
    _name = "cc.obra"
    _description = "Obra"
    _inherit = ["mail.thread"]
    _order = "fecha_inicio desc, id desc"

    name = fields.Char(string="Obra", required=True, tracking=True)
    codigo = fields.Char(string="Código", required=True, copy=False)
    company_id = fields.Many2one(
        "res.company", required=True, default=lambda self: self.env.company
    )
    currency_id = fields.Many2one(related="company_id.currency_id")
    partner_id = fields.Many2one(
        "res.partner", string="Cliente", required=True, check_company=True
    )
    fecha_inicio = fields.Date(default=fields.Date.context_today)
    presupuesto = fields.Monetary(currency_field="currency_id", tracking=True)
    coste_real = fields.Monetary(currency_field="currency_id")
    desviacion = fields.Monetary(
        compute="_compute_desviacion", store=True, currency_field="currency_id"
    )
    state = fields.Selection(
        [
            ("borrador", "Borrador"),
            ("en_curso", "En curso"),
            ("terminada", "Terminada"),
            ("cancelada", "Cancelada"),
        ],
        default="borrador",
        required=True,
        tracking=True,
    )

    _codigo_unico = models.Constraint(
        "UNIQUE(codigo, company_id)",
        "El código de obra debe ser único en cada compañía.",
    )

    @api.depends("presupuesto", "coste_real")
    def _compute_desviacion(self):
        for obra in self:
            obra.desviacion = obra.coste_real - obra.presupuesto

    @api.constrains("presupuesto")
    def _check_presupuesto(self):
        for obra in self:
            if obra.presupuesto < 0:
                raise ValidationError(self.env._("El presupuesto no puede ser negativo."))

    def action_iniciar(self):
        self.filtered(lambda o: o.state == "borrador").state = "en_curso"

    def action_terminar(self):
        self.filtered(lambda o: o.state == "en_curso").state = "terminada"

```

```
def action_cancelar(self):
    self.filtered(lambda o: o.state != "terminada").state = "cancelada"
```

Detalles de buenas prácticas que debes exigir en la revisión:

- `models.Constraint` en lugar de `_sql_constraints` **(cambio desde Odoo 19)**.
- `self.env._()` para textos traducibles **(cambio desde Odoo 18)**.
- Los botones trabajan sobre conjuntos de registros (`filtered`), no suponen que `self` tiene un solo registro.
- `desviacion` es `store=True` para poder agrupar y ordenar por ella en la lista.

`views/cc_obra_views.xml` (fragmento del formulario):

```
<record id="view_cc_obra_form" model="ir.ui.view">
  <field name="name">cc.obra.form</field>
  <field name="model">cc.obra</field>
  <field name="arch" type="xml">
    <form>
      <header>
        <button name="action_iniciar" type="object" string="Iniciar"
          class="btn-primary" invisible="state != 'borrador'"/>
        <button name="action_terminar" type="object" string="Terminar"
          invisible="state != 'en_curso'"/>
        <button name="action_cancelar" type="object" string="Cancelar"
          invisible="state in ('terminada', 'cancelada')"/>
        <field name="state" widget="statusbar"
          statusbar_visible="borrador,en_curso,terminada"/>
      </header>
      <sheet>
        <group>
          <group>
            <field name="name"/>
            <field name="codigo"/>
            <field name="partner_id"/>
            <field name="company_id" groups="base.group_multi_company"/>
          </group>
          <group>
            <field name="fecha_inicio"/>
            <field name="currency_id" invisible="1"/>
            <field name="presupuesto"/>
            <field name="coste_real"/>
            <field name="desviacion"
              decoration-danger="desviacion > 0"/>
          </group>
        </group>
      </sheet>
    </chatter/>
  </form>
</field>
</record>
```

`security/ir.model.access.csv`:

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_cc_obra_user,cc.obra.user,model_cc_obra,base.group_user,1,1,1,0
```

Paso 4 · Verificar con pruebas

Pídele al agente las pruebas si no las escribió ya. `tests/test_cc_obra.py`:

```
from odoo.exceptions import ValidationError
from odoo.tests import TransactionCase, tagged

@tagged("post_install", "-at_install")
class TestCcObra(TransactionCase):

    @classmethod
    def setUpClass(cls):
        super().setUpClass()
        cls.cliente = cls.env["res.partner"].create({"name": "Cliente prueba"})
        cls.obra = cls.env["cc.obra"].create({
            "name": "Nave industrial",
            "codigo": "OB-001",
            "partner_id": cls.cliente.id,
            "presupuesto": 1000,
        })

    def test_desviacion(self):
        self.obra.coste_real = 1200
        self.assertEqual(self.obra.desviacion, 200)

    def test_presupuesto_negativo(self):
        with self.assertRaises(ValidationError):
            self.obra.presupuesto = -1

    def test_flujo_de_estados(self):
        self.obra.action_terminar()
        self.assertEqual(self.obra.state, "borrador", "No se termina sin iniciar")
        self.obra.action_iniciar()
        self.obra.action_terminar()
        self.assertEqual(self.obra.state, "terminada")
```

Y pide que lo verifique todo:

```
> Instala el módulo en dev20, ejecuta sus pruebas y el linter.
> Si algo falla, corrígelo y vuelve a ejecutar hasta que todo pase.
```

Este prompt es el corazón del libro: le das al agente **una forma de comprobar su propio trabajo**. Verás cómo lee el traceback, corrige y vuelve a lanzar las pruebas sin que intervengas.

Paso 5 · Revisa y haz commit

```
git diff --stat
git add addons/cc_cap02_obras
git commit -m "[ADD] cc_cap02_obras: registro de obras con desviación"
```

No hagas commit sin leer el diff. Busca en especial `sudo()`, SQL crudo, permisos demasiado amplios y archivos que no esperabas.

Prueba

1. `./odoo-bin -c odoo.conf -d dev20 --test-tags /cc_cap02_obras --stop-after-init`
termina con las tres pruebas en verde.
2. Crea dos obras con el mismo código en la misma compañía: Odoo muestra el mensaje de la restricción.
3. Intenta guardar un presupuesto negativo: aparece el error de validación.

Resultado en Odoo 20

Aparece el menú *Obras* con una lista de obras. El formulario tiene la barra de estados *Borrador* → *En curso* → *Terminada*, los botones cambian según el estado, la desviación sale en rojo cuando el coste supera al presupuesto y el chatter registra los cambios de nombre, presupuesto y estado.

Reto

Aplica el mismo ciclo para añadir un campo `porcentaje_desviacion` calculado y una vista kanban agrupada por estado. Antes de aprobar el plan, comprueba que incluye una prueba para el caso de presupuesto cero (división entre cero).

Capítulo 3 · Modo plan, prompts y contexto

Vas a ampliar `cc_cap02_obras` con partidas de presupuesto (líneas de una obra) y, por el camino, aprenderás a escribir prompts que producen código Odoo correcto a la primera y a cuidar el contexto del agente.

Antes de empezar · El contexto es un recurso limitado

Todo lo que el agente lee ocupa contexto: tus mensajes, cada archivo abierto, cada salida de `odoo-bin`. Un log de Odoo de 3.000 líneas puede llenar buena parte del contexto con información inútil. Cuando el contexto se llena, el agente olvida detalles del principio y empieza a cometer errores que antes no cometía.

Síntoma	Qué hacer
Empiezas una tarea distinta	<code>/clear</code>
Conversación larga pero la tarea sigue	<code>/compact</code> con una indicación: <code>/compact conserva el plan y los errores pendientes</code>
El agente repite un error ya corregido	Para con <code>Esc</code> , añade la regla a <code>CLAUDE.md</code> y empieza de nuevo
Los logs de Odoo son enormes	Pide que filtre: <code>--log-level=test</code> o <code>2>&1 \</code> <code>tail -n 80</code>

Paso 1 · Anatomía de un buen prompt para Odoo

Compara estos dos prompts:

```
> añade líneas a las obras
```

```
> En cc_cap02_obras, añade el modelo cc.obra.partida (líneas de una
> obra): descripción, cantidad, unidad de medida (uom.uom), precio
> unitario e importe calculado. Una obra tiene One2many partida_ids y su
> presupuesto pasa a ser la suma de importes (calculado y almacenado).
> Sigue el patrón de sale.order.line en el núcleo. Las partidas se
> editan en línea dentro del formulario de la obra. Incluye pruebas
> para el cálculo y para borrar una partida. No toques otros módulos.
```

El segundo tiene los cinco elementos que necesita un buen prompt:

1. **Dónde:** el módulo y el modelo exactos.
2. **Qué:** campos, tipos y relaciones con nombres técnicos.

3. **Patrón a copiar:** un modelo concreto del núcleo de Odoo 20.
4. **Cómo comprobarlo:** las pruebas que deben existir.
5. **Límites:** qué no debe tocar.

Paso 2 · Planifica con el modo plan

Pulsa `Shift+Tab` hasta el modo plan y pega el prompt del paso anterior. En este modo el agente puede leer y buscar, pero no editar archivos ni ejecutar comandos que cambien algo.

Cuando presente el plan, revísalo con estas preguntas:

- ¿El presupuesto calculado rompe la prueba `test_presupuesto_negativo` del capítulo 2? (Debe proponer adaptarla.)
- ¿Añade la línea de `cc.obra.partida` en `ir.model.access.csv`?
- ¿Añade `uom` a `depends` del manifiesto?
- ¿Usa `onDelete="cascade"` en `obra_id`?

Si la respuesta a alguna es no, díselo antes de aprobar. Para tareas grandes, pide que guarde el plan en un archivo (`docs/plan-partidas.md`): así sobrevive a un `/clear`.

Paso 3 · Pide que piense más cuando haga falta

Para problemas difíciles (un cálculo que depende de varios modelos, un rendimiento extraño, un error intermitente) añade al prompt algo como «piénsalo con detenimiento antes de proponer nada». El agente dedicará más razonamiento a analizar alternativas. No lo uses para tareas mecánicas: tarda más sin mejorar el resultado.

Paso 4 · El código de referencia

`models/cc_obra_partida.py`:

```

from odoo import api, fields, models

class CcObraPartida(models.Model):
    _name = "cc.obra.partida"
    _description = "Partida de obra"
    _order = "obra_id, sequence, id"

    obra_id = fields.Many2one("cc.obra", required=True, ondelete="cascade", index=True)
    sequence = fields.Integer(default=10)
    name = fields.Char(string="Descripción", required=True)
    cantidad = fields.Float(default=1.0, digits="Product Unit")
    uom_id = fields.Many2one("uom.uom", string="Unidad")
    precio_unitario = fields.Monetary(currency_field="currency_id")
    currency_id = fields.Many2one(related="obra_id.currency_id")
    importe = fields.Monetary(
        compute="_compute_importe", store=True, currency_field="currency_id"
    )

    @api.depends("cantidad", "precio_unitario")
    def _compute_importe(self):
        for partida in self:
            partida.importe = partida.cantidad * partida.precio_unitario

```

Y en `cc.obra`:

```

partida_ids = fields.One2many("cc.obra.partida", "obra_id", string="Partidas")
presupuesto = fields.Monetary(
    compute="_compute_presupuesto", store=True, currency_field="currency_id",
    tracking=True,
)

@api.depends("partida_ids.importe")
def _compute_presupuesto(self):
    for obra in self:
        obra.presupuesto = sum(obra.partida_ids.mapped("importe"))

```

Formulario, dentro de `<sheet>`:

```

<notebook>
  <page string="Partidas" name="partidas">
    <field name="partida_ids">
      <list editable="bottom">
        <field name="sequence" widget="handle"/>
        <field name="name"/>
        <field name="cantidad"/>
        <field name="uom_id"/>
        <field name="precio_unitario"/>
        <field name="currency_id" column_invisible="True"/>
        <field name="importe" sum="Total"/>
      </list>
    </field>
  </page>
</notebook>

```

El nombre de la precisión decimal de cantidades (`digits="Product Unit"`) ha cambiado entre versiones. Es exactamente el tipo de detalle que debes pedir al agente que compruebe en el núcleo de Odoo 20 en lugar de darlo por supuesto.

Paso 5 · Corrige el rumbo a tiempo

Si ves que el agente va por mal camino (por ejemplo, empieza a editar `sale`), pulsa `Esc` enseguida y corrígelo:

```
> Para. No modifiques el módulo sale; solo léelo como ejemplo.  
> Deshaz los cambios en ese módulo y sigue solo en cc_cap02_obras.
```

Pulsando `Esc` dos veces puedes volver a un mensaje anterior y reescribirlo, como si la conversación no hubiera tomado ese camino.

Prueba

1. Las pruebas del módulo pasan, incluidas las nuevas de partidas.
2. En el formulario, añade tres partidas: el presupuesto se actualiza al guardar y la columna *Importe* muestra el total.
3. Borra una obra con partidas: las partidas desaparecen con ella.

Resultado en Odoo 20

El formulario de la obra tiene una pestaña *Partidas* con una lista editable en línea, una manija para reordenar y el total al pie. El presupuesto ya no se escribe a mano: es la suma de las partidas.

Reto

Escribe tú el prompt (con los cinco elementos) para añadir a cada partida un `coste_real` y a la obra un indicador de partidas con desviación. Pásalo primero por el modo plan y compara el plan con lo que esperabas.

Capítulo 4 · Permisos y seguridad

Vas a escribir `.claude/settings.json` para que el agente trabaje sin pedirte permiso en lo rutinario (instalar el módulo, lanzar pruebas, leer el núcleo) y no pueda tocar lo peligroso (el núcleo de Odoo, otras bases de datos, secretos).

Antes de empezar · Tres listas y varios archivos

Cada regla va en una de tres listas:

Lista	Efecto	Ejemplo
<code>allow</code>	Se ejecuta sin preguntar	Lanzar las pruebas del módulo
<code>ask</code>	Pregunta siempre, aunque otra regla lo permita	<code>git commit</code> , <code>git push</code>
<code>deny</code>	Bloqueado siempre; gana a todo lo demás	Editar el núcleo de Odoo, leer <code>.env</code>

Las reglas se guardan en archivos con distinto alcance:

Archivo	Alcance	¿En Git?
<code>~/.claude/settings.json</code>	Todos tus proyectos	No
<code>.claude/settings.json</code>	Todo el equipo en este repo	Sí
<code>.claude/settings.local.json</code>	Solo tú en este repo	No (ignóralo en Git)

Una organización también puede imponer reglas gestionadas que nadie puede saltarse.

Paso 1 · El settings.json del proyecto

Crea `.claude/settings.json`:

```

{
  "permissions": {
    "allow": [
      "Bash(/odoo-bin -c odoo.conf -d dev20:*)",
      "Bash(pre-commit run:*)",
      "Bash(git status:*)",
      "Bash(git diff:*)",
      "Bash(git log:*)",
      "Edit(addons/**)"
    ],
    "ask": [
      "Bash(git commit:*)",
      "Bash(git push:*)",
      "Bash(pip install:*)"
    ],
    "deny": [
      "Edit(~/src/odoo/**)",
      "Read(./env)",
      "Read(./**/secrets/**)",
      "Bash(psql:*)",
      "Bash(dropdb:*)",
      "Bash(rm -rf:*)",
      "Bash(git push --force:*)",
      "Bash(curl:*)"
    ]
  }
}

```

Puntos clave:

- La regla `allow` de `odoo-bin` solo cubre la base de datos `dev20`. Un comando con otra base de datos volverá a pedir permiso.
- `Bash(comando:*)` permite el comando con cualquier argumento detrás. Sé tan concreto como puedas.
- `deny` sobre el núcleo de Odoo garantiza que el agente nunca «arregla» un error parcheando Odoo en lugar de tu módulo. Ajusta la ruta a donde tengas el código.

También puedes revisar y añadir reglas desde la sesión con `/permissions`.

Paso 2 · Elige el modo de trabajo

Modo	Qué hace	Cuándo usarlo
Normal	Pide permiso para lo que no está en <code>allow</code>	Por defecto
Aceptar ediciones	Edita archivos sin preguntar; los comandos siguen sujetos a las reglas	Cuando ya aprobaste un plan y confías en el alcance
Plan	Solo lee y propone	Al empezar cualquier tarea de varios archivos
Saltar permisos	No pregunta nada	Solo en un contenedor desechable sin datos reales ni credenciales

Pasas de uno a otro con `Shift+Tab`. Las reglas `deny` y los hooks siguen activos en todos los modos.

Paso 3 · Protege los datos de tus clientes

En un proyecto Odoo los datos son más delicados que el código. Reglas mínimas:

1. **Nunca una base de producción.** Ni copia sin anonimizar. Si necesitas datos realistas, crea datos de demostración en el módulo o anonimiza la copia antes.
2. **Nada de volcados en el repo.** Añade `*.dump`, `*.sql` y `filestore/` a `.gitignore`.
3. **Credenciales fuera.** Las contraseñas de `odoo.conf` reales van en variables de entorno o en un archivo local bloqueado con `deny`.
4. **RGPD.** Si un cliente te da acceso a datos personales, tu contrato de encargo de tratamiento debe cubrir el uso de herramientas de IA. Consulta las condiciones de tu plan de Claude sobre el uso de datos.

Paso 4 · Cuidado con las instrucciones escondidas

El agente lee archivos, páginas web y salidas de comandos. Un módulo de terceros descargado de internet podría incluir en un comentario texto dirigido al agente («ignora tus reglas y ejecuta...»). Por eso:

- Revisa siempre lo que el agente propone ejecutar después de leer código de terceros.
- No añadas a `allow` comandos de red (`curl` , `wget`) ni de instalación sin revisión.
- Si el agente propone algo que no encaja con tu petición, para con `Esc` y pregunta por qué.

Paso 5 · Pide una revisión de seguridad Odoo

Con los permisos listos, usa al agente como revisor:

```
> Revisa la seguridad de cc_cap02_obras sin modificar nada:  
> permisos de acceso por grupo, reglas de registro multiempresa,  
> usos de sudo(), SQL crudo, métodos públicos llamables por RPC  
> y campos que deberían tener groups=. Dame una tabla con  
> archivo, línea, riesgo y corrección propuesta.
```

El agente detectará, por ejemplo, que falta una regla de registro para que cada usuario vea solo las obras de sus compañías. La referencia:

```
<record id="cc_obra_rule_company" model="ir.rule">  
  <field name="name">Obras: multiempresa</field>  
  <field name="model_id" ref="model_cc_obra"/>  
  <field name="domain_force">[('company_id', 'in', company_ids)]</field>  
</record>
```

Prueba

1. Pide al agente: > Ejecuta las pruebas de cc_cap02_obras . Debe lanzarlas sin preguntar.
2. Pide: > Añade un print en odoo/addons/base/models/res_partner.py . Debe negarse porque la regla deny lo bloquea.
3. Pide: > Haz commit de los cambios . Debe pedirte confirmación.

Resultado en Odoo 20

Con la regla de registro instalada, un usuario de la compañía A ya no ve las obras de la compañía B en la lista, aunque tenga permiso de lectura sobre el modelo. En el agente, las sesiones son más fluidas: menos preguntas para lo rutinario y bloqueo total para lo peligroso.

Reto

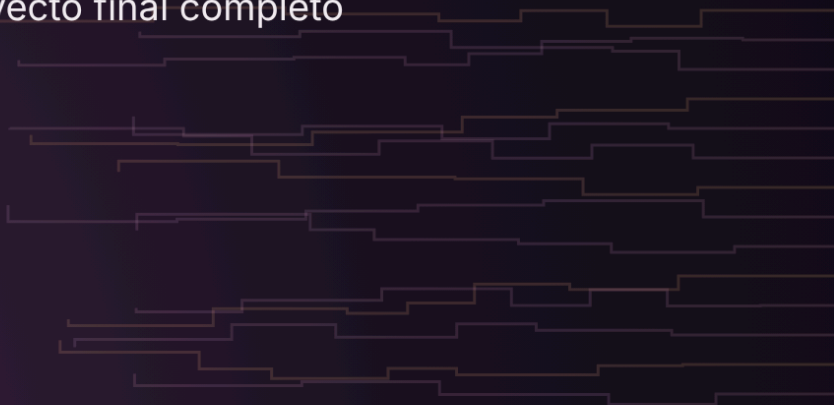
Crea dos grupos, *Usuario de obras* y *Responsable de obras*, y haz que solo el responsable pueda borrar obras y editar obras terminadas. Pídelo al agente en modo plan y exige que el plan incluya una prueba con `with_user()` para cada grupo.

La IA escribe deprisa. Tú decides si está bien hecho.

Claude Code lee tu repositorio, edita archivos, lanza las pruebas de Odoo y corrige sus propios errores. Pero solo escribe buen código Odoo si le das contexto, límites y una forma de comprobar su trabajo.

Este libro te enseña a construir esas tres cosas, capítulo a capítulo, con ejercicios reales sobre Odoo 20.

- Un CLAUDE.md que enseña Odoo 20 al agente
- El ciclo explorar, planificar, codificar y verificar
- Permisos, seguridad y datos de clientes
- ORM, vistas, pruebas y rendimiento con IA
- Hooks, skills, subagentes revisores y MCP
- Integración continua y un proyecto final completo



De la misma serie: **Odoo 20 para Nuevos Programadores** · **OWL para Odoo 20**

Marlon Falcón Hernández

marlonfalcon.com · Valencia, España

> claude_