

---

## Guía de Desarrollo en Odoo: Comandos, Estructura y Vistas

### Parar/iniciar odoo desde sge

```
sudo service odoo stop
```

```
sudo service odoo start
```

### Conectarse a usuarios de la Maquina Virtual

```
ssh -p 2222 sge@127.0.0.1
```

```
ssh -p 2222 odoo@127.0.0.1
```

### Crear Modulo en odoo

```
odoo scaffold NombreModulo ./modules
```

### Cargar Modulo en odoo

```
odoo -d BaseDatos -u NombreModulo
```

### Carpeta Imágenes

```
-static
```

```
-description
```

```
-icon.png
```

### 1. Principales Comandos de Odoo

Para trabajar con Odoo, especialmente en entornos de desarrollo Linux o mediante terminal, se utilizan los siguientes comandos fundamentales:

- **Creación de la estructura base:** `odoo scaffold NombreModulo ./modules`
- Este comando genera automáticamente la carpeta del módulo con todos los directorios y ficheros necesarios para empezar a programar.
- **Gestión de directorios y permisos (Linux/Docker):**
  - `mkdir modules`: Crea un directorio para almacenar tus módulos personalizados.
  - `chmod 777 -R <ruta_modulo>`: Otorga permisos completos a la carpeta del módulo para poder editarla fácilmente, especialmente útil en contenedores Docker.
- **Configuración y ejecución del servidor:**

- odoos --addons-path="" --save: Añade directorios de módulos al PATH de Odoo y guarda la configuración en el archivo .odoorc.
- odoos -d examen -u david el servidor Odoo e **instala o actualiza** un módulo específico en la base de datos indicada.
- odoos --dev=all: Lanza Odoo en **modo desarrollo**, permitiendo que los cambios en ficheros XML y Python se apliquen sin necesidad de reiniciar el servicio o actualizar el módulo manualmente.
- **Gestión del servicio del sistema:**
  - sudo service odoo stop: Detiene el servicio de Odoo si se está ejecutando en segundo plano.
  - sudo systemctl disable/enable odoo.service: Deshabilita o habilita el inicio automático de Odoo con el sistema.

## 2. Estructura Básica de un Módulo

Un módulo de Odoo se organiza en una carpeta con el nombre del módulo y contiene archivos específicos que el framework reconoce para cargar la lógica y la interfaz:

1. **\_\_manifest\_\_.py**: Es el archivo principal de metadatos. Define el nombre del módulo, dependencias (depends), archivos XML de datos y vistas a cargar (data), categoría y versión.
2. **\_\_init\_\_.py**: Archivo de inicialización de Python que indica qué otros ficheros o subcarpetas con código Python deben importarse.
3. **models/**: Carpeta que contiene la lógica de negocio. Incluye archivos .py donde se definen las clases que heredan de models.Model para mapearse en la base de datos mediante el ORM.
4. **views/**: Carpeta para los archivos XML que describen la interfaz de usuario (menús, acciones y tipos de vistas).
5. **security/**: Contiene archivos como ir.model.access.csv, que definen los permisos de lectura, escritura y borrado para los diferentes grupos de usuarios.
6. **static/**: Carpeta opcional para recursos estáticos como imágenes, archivos CSS o Javascript.

## 3. Tipos de Vistas y su Implementación

Las vistas definen cómo se muestran los datos del modelo al usuario. Se declaran en archivos XML dentro del modelo ir.ui.view.

### A. Vista de Lista (List/Tree)

Muestra múltiples registros en formato de tabla. Se implementa con la etiqueta <list> (anteriormente <tree>).

- **Atributos útiles:**

- decoration-<estilo>="condición": Cambia el color de la fila (ej. decoration-info, decoration-danger) según una condición.
- editable="top" o "bottom": Permite editar los registros directamente en la lista sin abrir el formulario.
- sum="Etiqueta": Muestra el sumatorio total de una columna numérica.

## B. Vista de Formulario (Form)

Permite crear o editar un registro individual. Se recomienda usar contenedores para organizar el diseño:

- <sheet>: Crea un contenedor central para que el formulario no ocupe toda la pantalla.
- <group>: Alinea automáticamente los campos y sus etiquetas.
- <notebook> y <page>: Crean pestañas para organizar formularios complejos.
- **Widgets:** Permiten cambiar la apariencia de un campo (ej. widget="image", widget="progressbar", widget="many2many\_tags").

## C. Vista Kanban

Muestra los registros como tarjetas visuales, permitiendo un diseño muy flexible mediante el motor de plantillas **QWeb**. Se utiliza la etiqueta <kanban> y requiere definir plantillas HTML dentro de <templates>.

## D. Vista de Calendario (Calendar)

Muestra los registros en un calendario. Requiere al menos un campo de fecha de inicio.

- **Implementación:** <calendar string="Título" date\_start="campo\_fecha">.

## E. Vista de Gráfico (Graph)

Permite visualizar datos numéricos en gráficos de tipo tarta, barras o líneas.

- **Implementación:** <graph string="Título" type="bar"> y define campos con type="row" (ejes) y type="measure" (datos).

## F. Vista de Búsqueda (Search)

Define los filtros y opciones de agrupación que aparecen en la barra superior de las vistas de lista.

- <filter>: Crea filtros rápidos con un dominio específico (ej. domain="['campo', '=', True])" o agrupaciones mediante el contexto.

**Analogía para entender la estructura:** Desarrollar en Odoo es como **construir un edificio modular**: el scaffold es el plano básico; el \_\_manifest\_\_ es la lista de materiales y permisos; los models son los cimientos de la base de datos; y las views son la decoración y las ventanas que permiten a los habitantes (usuarios) interactuar con la casa.

Para entender cómo Odoo gestiona los datos, es fundamental comprender su **capa ORM (Object Relational Mapping)**, que permite definir relaciones entre modelos de Python que se traducen automáticamente en tablas de la base de datos PostgreSQL.

A continuación, se explican los tres tipos de relaciones principales utilizando el ejemplo de una **biblioteca**:

### 1. ManyZone (Muchos a Uno)

Es la relación más simple y común. Indica que **muchos registros de un modelo se relacionan con un único registro de otro modelo**. En la base de datos, esto genera una **clave ajena (foreign key)** en la tabla del modelo que declara el campo.

- **Ejemplo en la biblioteca:** Muchos **Libros** pertenecen a un único **Autor**.
- **Implementación:** En el modelo `biblioteca.libro`, definiríamos el autor así: `autor_id = fields.ManyZone('biblioteca.autor', string="Autor")`.

### 2. One2many (Uno a Muchos)

Es la relación **inversa al ManyZone**. Este campo no existe físicamente en la tabla de la base de datos; funciona como un campo calculado que realiza una consulta SELECT sobre las claves ajenas del otro modelo. **Para que un One2many funcione, es obligatorio que exista un ManyZone en el modelo relacionado**.

- **Ejemplo en la biblioteca:** Un **Autor** tiene una lista de muchos **Libros**.
- **Implementación:** En el modelo `biblioteca.autor`, para ver todos sus libros: `libro_ids = fields.One2many('biblioteca.libro', 'autor_id', string="Libros Escritos")`. (*Aquí, 'autor\_id' es el nombre del campo ManyZone que definimos en el modelo Libro*).

### 3. Many2many (Muchos a Muchos)

Se utiliza cuando **múltiples registros de un modelo pueden estar relacionados con múltiples registros de otro**. Para gestionar esto, Odoo crea automáticamente una **tabla intermedia** con las claves ajenas de ambos modelos, evitando redundancias.

- **Ejemplo en la biblioteca:** Un **Libro** puede tratar sobre varias **Categorías** (ej. Misterio y Aventura), y una **Categoría** puede incluir muchos **Libros**.
- **Implementación:** En el modelo `biblioteca.libro`: `categoria_ids = fields.Many2many('biblioteca.categoría', string="Categorías")`.
- **Control avanzado:** Si necesitas que la tabla intermedia tenga un nombre específico o evitar conflictos entre relaciones, puedes definir los atributos `relation` (nombre de la tabla), `column1` y `column2`.

---

**Resumen visual de las relaciones:**

Tipo de Relación	Persistencia en Base de Datos	Ejemplo Biblioteca
<b>Many2one</b>	Crea una columna (Clave ajena)	Libro -> Autor
<b>One2many</b>	Virtual (no crea columna)	Autor -> Lista de Libros
<b>Many2many</b>	Crea una tabla intermedia	Libros <-> Categorías

Para visualizar estos datos en la interfaz de usuario, se suelen usar **widgets** específicos en las vistas: los Many2one suelen aparecer como campos de selección, mientras que los One2many y Many2many se muestran frecuentemente como listas incrustadas o etiquetas (many2many\_tags).

**Metáfora para recordar:** Las relaciones en Odoo son como las **conexiones en un aeropuerto**: un **Many2one** es un pasajero con un solo destino; un **One2many** es la pantalla de la puerta de embarque que muestra a todos los pasajeros de ese vuelo; y un **Many2many** es el área de tránsito, donde muchos pasajeros de diferentes orígenes se mezclan para ir a muchos destinos distintos.