educación IT
CODE YOUR FUTURE

Package Managers

¿Qué son las Librerías?

Son **colecciones de funciones y recursos reutilizables** que facilitan el desarrollo de *software* y permiten a los desarrolladores **reutilizar bloques de código** para ahorrar tiempo y simplificar la implementación de funcionalidades específicas.

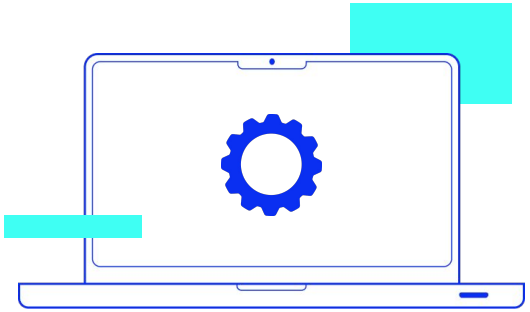
- Facilitan el desarrollo y mantenimiento de *software*.
- Aumentan la eficiencia y reducen errores.

Entender las principales librerías de un lenguaje es crucial para dominarlo por completo.

Introducción a los Administradores de paquetes

Son herramientas esenciales en el desarrollo de *software*, permiten la **instalación, actualización y gestión de librerías y dependencias** de manera eficiente.

- Simplifican la instalación y actualización de librerías.
- Aseguran compatibilidad y versiones correctas de las dependencias.



Antes de los administradores de paquetes	Después de los administradores de paquetes
En el pasado, era **necesario, de forma manual, descargar y configurar cada librería desde la web.**	**La gestión de dependencias se simplifica enormemente.** Basta con unos pocos comandos para **instalar y actualizar bibliotecas de manera automatizada.** De esta manera, se garantiza la compatibilidad y se evitan errores comunes.
Ese proceso era tedioso y presentaba algunos **desafíos:** • Dependencia manual de librerías. • Riesgos de incompatibilidad y versiones incorrectas.	**Ventajas:** • Gestión automatizada de dependencias. • Instalación y actualización sencilla de librerías. • Resolución automática de conflictos y versiones.

Cada tecnología tiene su administrador de paquetes característico

	JavaScript (Node.js)	Usa ***Node Package Manager (npm)*** **y Yarn** para manejar paquetes JavaScript.
	.NET	Usa **NuGet.**
	Python	Utiliza **pip** y **Conda** para paquetes y entornos de Python.
	Java	Emplea **Apache Maven** y **Gradle Build Tool.**

npm

Administrador de paquetes en JavaScript

En el ecosistema de JavaScript, *npm* es uno de los administradores de paquetes más populares.

- Es el **administrador de paquetes predeterminado para Node.js** (entorno de ejecución para JavaScript construido sobre el motor de JavaScript V8 de Google Chrome).
- Posee un gran repositorio de paquetes: ***npm registry*.**

> Link al sitio oficial.

Instalación

Comando de instalación:

```
npm install <package>.
```

Uso

npm se incluye en la instalación de Node.js.

1. Para comenzar a trabajar con un proyecto usando *npm*, primero debes **inicializarlo** con el comando:

   ```
   npm init
   ```

   Este comando creará un archivo ***package.json*** en el que se detallan las dependencias del proyecto y otras configuraciones importantes.

2. Una vez que el proyecto está configurado, puedes **gestionar sus dependencias** con el comando:

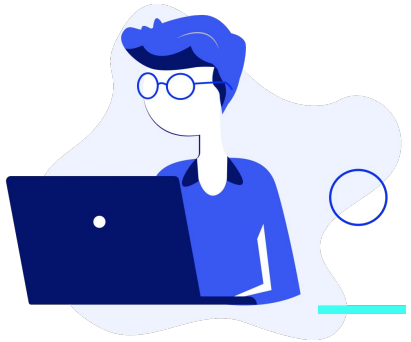
   ```
   npm install
   ```

   Este comando descarga las dependencias necesarias y las instala en la carpeta ***node_modules*** del proyecto.

Ejemplo:

```
# Iniciar un nuevo proyecto con npm init
npm init

# Instalación de lodash usando npm npm
install lodash
```



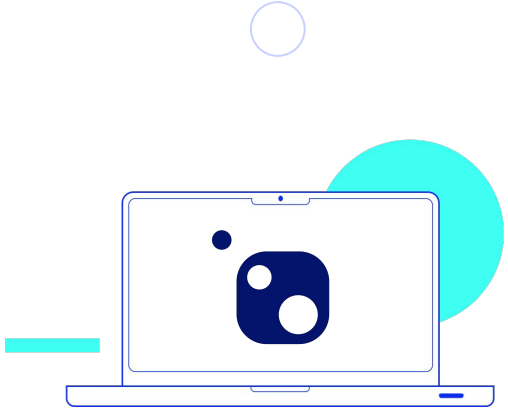
NuGet

Administrador de paquetes en C# (.NET)

NuGet es el administrador de paquetes más utilizado en el **ecosistema .NET**. Permite agregar, actualizar y gestionar librerías en **proyectos C#**.

Características

- Administrador de paquetes oficial para .NET.
- Repositorio de paquetes: NuGet Gallery.
- Integrado en Visual Studio y .NET CLI.



Instalación

Se puede instalar una librería también con el comando:

```
dotnet
```

Ejemplo:

```
# Instalación de Newtonsoft.Json usando NuGet CLI nuget install Newtonsoft.Json

# Instalación usando .NET CLI
dotnet add package Newtonsoft.Json
```

pip y Conda

Administradores de paquetes en Python

En el ecosistema de **Python,** *pip* y *Conda* son los administradores de paquetes más utilizados. Permiten una **gestión eficiente de dependencias y entornos.**

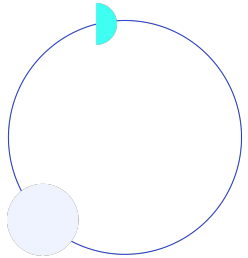
 pip	 Conda
■ Administrador de paquetes predeterminado. ■ Repositorio de paquetes: ***Python Package Index (PyPI)***. ■ Comando de instalación: `pip install <package>`	■ Administrador de paquetes y entornos para Python. ■ Comando de instalación: `conda install <package>`

Instalación y uso

El administrador de paquetes estándar se instala, automáticamente, junto con Python. Permite la instalación de paquetes de manera sencilla y eficiente.

- **Dependencias globales:** Las dependencias instaladas con pip se gestionan de manera global en el sistema, lo que puede causar conflictos entre proyectos si no se manejan correctamente.
- **Uso de entornos virtuales:** Se recomienda el uso de entornos virtuales como ***venv*** (incluido con Python) o ***conda*** (a través de Anaconda) para aislar y gestionar las dependencias de manera independiente para cada proyecto.

```
# Instalación de una biblioteca usando pip
pip install nombre_de_la_biblioteca
```



Maven y Gradle

Administradores de paquetes en Java

Apache Maven y *Gradle Build Tool* son los administradores de paquetes más populares en el ecosistema de Java. **Facilitan la gestión de dependencias y la automatización de *builds.***

Apache Maven	Gradle Build Tool
■ Basado en la convención sobre la configuración. ■ Utiliza un archivo ***pom.xml*** para gestionar dependencias y configuración del proyecto. ■ Comando de instalación: `mvn install`	■ Flexible y basado en *scripts.* ■ Utiliza archivos ***build.gradle*** para gestionar dependencias y tareas. ■ Comando de instalación: `gradle build`

Instalación y uso

Maven maneja las dependencias descargando automáticamente los artefactos desde los repositorios remotos.

1. Para agregar una dependencia en Maven se debe incluir el bloque **<dependency>** en el archivo ***pom.xml*** con el artefacto que deseas agregar.
2. Después de agregar la dependencia Maven **detectará** automáticamente los cambios y comenzará a **descargar las dependencias** especificadas.

Es posible utilizar comandos como:

```
mvn clean install
```

para compilar y asegurarse de que todas las dependencias estén descargadas y configuradas.

Ejemplo:

```
<!-- Ejemplo de pom.xml para Maven -->
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.12.0</version>
</dependency>


#Luego desde la consola

mvn clean install
```

Resumen de gestión de dependencias

JavaScript (Node.js)

Se utiliza ***package.json*** junto con **npm** o **Yarn** para especificar y gestionar dependencias.

Python

Se utiliza ***requirements.txt*** para listar las dependencias del proyecto, instalables con **pip.**

C#

Se utiliza **NuGet** para gestionar las dependencias del proyecto, configurando los paquetes en el archivo ***csproj***.

Java

Se utiliza ***pom.xml*** en proyectos **Apache Maven** o ***build.gradle*** en proyectos **Gradle Build Tool** para definir y gestionar las dependencias.