



2. Instalación de Software

Copyright (c) 2025 Adrián Quiroga Linares Lectura y referencia permitidas;
reutilización y plagio prohibidos

2.1 Formas de instalación

Podemos clasificar la instalación en dos grandes familias según cómo nos llega el software:

Método	Descripción	Ventajas 	Inconvenientes 
Paquetes Precompilados (Binarios)	El programa ya viene "cocinado" y listo para ejecutarse. Es el estándar actual.	<ul style="list-style-type: none"> • Instalación rápida. • Gestión automática de dependencias. • Actualizaciones sencillas. 	<ul style="list-style-type: none"> • Menor optimización (genérico para todas las CPUs). • Menos personalizable.
Desde Código Fuente (Compilación)	Descargas el código "crudo" y tu ordenador lo traduce a lenguaje máquina.	<ul style="list-style-type: none"> • Máxima optimización (adaptado a tu hardware específico). • Puedes activar/desactivar funciones concretas. 	<ul style="list-style-type: none"> • Complejo y lento. • Gestión manual de errores. • Difícil de desinstalar limpiamente.

2.1.1 Gestores de paquetes

En la mayoría de distribuciones, usamos **Gestores de Paquetes**. Estos programas consultan una base de datos central (repositorios), descargan el software y lo instalan.

Ventajas del sistema de paquetes:

- **Centralización:** Todo el software viene de fuentes oficiales y verificadas (sin virus).

- **Dependencias:** Si instalas el "Programa A" y este necesita la "Librería B", el gestor descarga ambos automáticamente.
- **Mantenimiento:** Un solo comando actualiza *todo* el sistema y sus programas.

El mundo Linux se divide principalmente en dos "dialectos":

1. **Paquetes .deb:** Usados por **Debian**, Ubuntu, Linux Mint, Kali.
2. **Paquetes .rpm:** Usados por **RedHat**, Fedora, CentOS, SUSE.

2.1.3 Gestión de paquetes en Debian

Debian utiliza una estructura de herramientas por capas ("cebolla"). De más bajo nivel a más alto nivel:

Nivel bajo: **dpkg** (Debian Package)

Es la herramienta base. Gestiona los archivos **.deb** individuales que ya tienes descargados en tu disco.

- **Limitación grave: NO resuelve dependencias.** Si el paquete necesita una librería extra, **dpkg** fallará y te dirá que la busques tú.
- *Uso:* Instalar un **.deb** suelto descargado manualmente.

Nivel Alto: **apt** (Advanced Package Tool)

Es la herramienta inteligente que usamos a diario. Se conecta a internet (repositorios).

- **La magia:** Resuelve dependencias. Si pides instalar **vlc**, **apt** calculará qué librerías faltan, las descargará de los servidores y llamará a **dpkg** por detrás para instalarlas.
- *Fuentes:* Lee la lista de repositorios desde **/etc/apt/sources.list**.

Herramientas Especiales

- **Tiendas de Software (GUI):** Interfaces gráficas (como GNOME Software) que por debajo ejecutan **apt**.
- **alien:** Una herramienta de "traducción". Permite convertir un paquete **.rpm** (de RedHat) a **.deb** (para Debian).

2.1.4 La nueva Era: Paquetes Universales y Contenedores

Para solucionar el problema de "esto funciona en Fedora pero no en Ubuntu", nacieron los paquetes autocontenidos.

SNAP y FLATPAK

Son paquetes que incluyen **todas sus dependencias dentro**. Llevan sus propias librerías y no usan las del sistema operativo

Característica	Paquete Tradicional (apt)	Paquete Universal (snap/flatpak)
Dependencias	Usa las librerías compartidas del sistema.	Autocontenido: Lleva sus propias librerías dentro.
Seguridad	Acceso total al sistema (generalmente).	Sandboxing: Se ejecutan en una "caja" aislada (contenedor).
Versiones	Depende de la versión de tu Linux.	Siempre la última versión disponible del desarrollador.
Tamaño	Pequeño (reusa librerías).	Grande (mucha redundancia de librerías).
Arranque	Rápido.	Lento (sobre todo el primer arranque).

2.1.5 Gestores de Lenguajes: PIP (Python)

A veces no instalamos un "programa", sino una librería para programar. Cada lenguaje tiene su propio gestor:

PIP es el estándar para instalar librerías del lenguaje Python:

- **pip (o pip2):** Para Python antiguo (versión 2.7, obsoleto).
- **pip3:** Para Python moderno (versión 3.x).

SHELL

```
# Instalar el gestor pip
sudo apt install python3-pip

# Instalar una librería de Python (ej. NumPy para matemáticas)
pip3 install numpy

# Desinstalar
pip3 remove pytorch
```

2.2 dpkg - Debian Package

dpkg (Debian Package) es la herramienta base para gestionar paquetes **.deb**. A diferencia de **apt**, **dpkg no descarga nada de internet ni resuelve dependencias**. Solo trabaja con archivos que ya tienes descargados localmente en tu disco duro.

2.2.1 Estructura de un Paquete DEB

Un archivo **.deb** es un contenedor comprimido (similar a un .zip) que incluye:

1. **Binarios:** El programa ejecutable, librerías y documentación.
2. **Metadatos:** Información de versión, dependencias, arquitectura y scripts de configuración (pre-instalación y post-instalación).

2.2.2 Nomenclatura estándar

Formato: `paquete_versión-build_arquitectura.deb`

Parte	Descripción	Ejemplo
Paquete	Nombre de la aplicación.	<code>ethereum</code>
Versión	Versión del desarrollador.	<code>0.10.11</code>
Build	Revisión/Compilación del empaquetador.	<code>1</code>
Arquitectura	CPU destino (amd64, i386, armhf).	<code>i386</code>
Extensión	Siempre .deb	<code>.deb</code>

2.2.3 Opciones `dpkg`

Acción	Opción Corta	Opción Larga	Argumento necesario
Instalar	<code>-i</code>	<code>--install</code>	Nombre del Fichero (.deb)
Eliminar	<code>-r</code>	<code>--remove</code>	Nombre del Paquete
Purgar	<code>-P</code>	<code>--purge</code>	Nombre del Paquete
Listar	<code>-l</code>	<code>--list</code>	(Opcional) Patrón
Estado	<code>-s</code>	<code>--status</code>	Nombre del Paquete
Ver Ficheros	<code>-L</code>	<code>--listfiles</code>	Nombre del Paquete
Buscar	<code>-S</code>	<code>--search</code>	Ruta de un fichero

2.2.4 Operaciones de instalación y eliminación

Instalación (`-i`)

Instala un paquete que tienes descargado.

- ⚠ **Importante:** Es el único caso donde necesitas escribir el **nombre completo del archivo** (incluyendo `.deb`).
- Dependencias:** Si el paquete necesita librerías que no tienes, `dpkg` fallará (no las descarga automáticamente).

SHELL

```
# Sintaxis: dpkg -i [fichero.deb]
dpkg -i ethereum_0.10.11-1_i386.deb
```

Si falla por dependencias, un truco es ejecutar inmediatamente después `apt-get install -f` para que APT descargue lo que falta.

Eliminación (-r vs -P)

Aquí ya no usamos el nombre del archivo, sino el **nombre del paquete** (ej: `wget`).

1. **Eliminar (remove):** Desinstala el programa, pero **mantiene los ficheros de configuración** en `/etc`. Útil si planeas reinstalarlo en el futuro y no quieres perder tu configuración.

```
dpkg -r wget
```

```

2. **Purgar (`purge`):** Desinstala el programa y **borra también la configuración**. Limpieza total.

```
```bash
dpkg -P wget
```

```

### Reconfiguración

Si un paquete ya instalado da problemas o quieres cambiar sus opciones iniciales (como el idioma o la distribución de teclado):

```
```bash
dpkg-reconfigure nombre_paquete
```

```

SHELL

## 2.2.5 Consultas e información

### Listar Paquetes (-l)

Muestra una tabla con todos los paquetes instalados. Es muy útil entender la primera columna (Estado).

```
dpkg -l "telnet*"

```

SHELL

**Interpretación de la salida (Las siglas `ii`, `un`, etc.):** La primera columna tiene dos o tres letras.

- **1ª Letra (Deseado):** Lo que tú quieres que pase.
  - `i`: Install (Quieres que esté instalado).
  - `r`: Remove (Quieres desinstalarlo).
- **2ª Letra (Actual):** Lo que realmente hay.
  - `i`: Installed (Está instalado correctamente).

- **c**: Config-files (Se borró el programa, pero quedan archivos de configuración).
- **n**: Not-installed (No está instalado).

**El objetivo:** Un paquete sano debe mostrar **ii** (Deseado: Instalar / Actual: Instalado).

## Ver Estado Detallado (-s)

Muestra toda la información técnica (versión, dependencias, tamaño).

SHELL

```
$ dpkg --status wget
Package: wget
Status: install ok installed <-- ESTADO IMPORTANTE
Priority: important
Section: web
Version: 1.10-2
Depends: libc6, libssl0.9.7 <-- DEPENDENCIAS
Description: Retrieves files from the web...
```

## Ver Ficheros de un Paquete (-L)

¿Qué ha instalado este paquete y dónde?

SHELL

```
$ dpkg -L wget
/usr/bin/wget (El ejecutable)
/etc/wgetrc (La configuración)
/usr/share/man/man1/.. (El manual)
```

## Búsqueda Inversa (-s)

¿Tienes un archivo misterioso y no sabes qué paquete lo puso ahí?

SHELL

```
¿A quién pertenece este fichero?
$ dpkg -S /usr/bin/wget
wget: /usr/bin/wget
```

## 2.3 APT - Advanced Packaging Tools

**APT** es la herramienta de gestión de paquetes de alto nivel del proyecto Debian. Mientras que **dpkg** trabaja con archivos locales, **APT** gestiona todo el ciclo de vida del software conectándose a repositorios de internet.

**Sus superpoderes son:**

1. **Resolución de Dependencias:** Si quieres instalar el *Programa A*, y este necesita la *Librería B*, APT descarga e instala ambos automáticamente.
2. **Repositorios Remotos:** Descarga el software desde servidores oficiales (mirrors).
3. **Actualización Integral:** Actualiza todo el sistema operativo con un solo comando.

## 2.3.1 Archivo de configuración

### La lista de fuentes (`sources.list`)

El archivo `/etc/apt/sources.list` es el mapa del tesoro. Le dice a APT **dónde** buscar los paquetes.

**Sintaxis de una línea:** `[Tipo] [URI] [Distribución] [Componentes]`

Ejemplo real desglosado:

```
deb http://ftp.es.debian.org/debian/ bookworm main contrib non-free
```

SHELL

| Campo               | Significado    | Opciones comunes                                                                                                                                               |
|---------------------|----------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Tipo</b>         | ¿Qué bajamos?  | <code>deb</code> (Binarios/Programas listos)<br><code>deb-src</code> (Código fuente para compilar)                                                             |
| <b>URI</b>          | ¿Dónde está?   | <code>http://...</code> , <code>ftp://...</code> , <code>file://...</code>                                                                                     |
| <b>Distribución</b> | ¿Qué versión?  | <code>stable</code> (Estable/Recomendada)<br><code>testing</code> (Pruebas/Más nueva)<br><code>unstable</code> / <code>sid</code> (Desarrollo/Inestable)       |
| <b>Componentes</b>  | ¿Qué licencia? | <code>main</code> (Software libre oficial)<br><code>contrib</code> (Libre pero depende de no-libres)<br><code>non-free</code> (Software privativo/propietario) |

## Configuración Avanzada (`apt.conf`)

Para cambiar el comportamiento de APT (ej. configurar un proxy), se usan estos ficheros:

1. `/etc/apt/apt.conf`: Fichero único (estilo clásico).
2. `/etc/apt/apt.conf.d/`: Directorio modular (estilo moderno). Aquí se crean ficheros pequeños (ej: `01proxy`, `99mysettings`). APT los lee todos en orden numérico. Es más ordenado y fácil de administrar.

### 2.3.2 Comandos y opciones de APT

Históricamente había dos comandos separados: uno para *hacer* cosas (`apt-get`) y otro para *buscar* cosas (`apt-cache`). Modernamente, se recomienda usar `apt`, que unifica a los dos anteriores y añade barras de progreso y colores.

| Acción           | Comando Moderno (Recomendado) | Comando Clásico (Scripts)    | Comando Consulta              |
|------------------|-------------------------------|------------------------------|-------------------------------|
| Actualizar lista | <code>apt update</code>       | <code>apt-get update</code>  | -                             |
| Instalar         | <code>apt install</code>      | <code>apt-get install</code> | -                             |
| Eliminar         | <code>apt remove</code>       | <code>apt-get remove</code>  | -                             |
| Buscar           | <code>apt search</code>       | -                            | <code>apt-cache search</code> |
| Mostrar info     | <code>apt show</code>         | -                            | <code>apt-cache show</code>   |

### 2.3.3 Flujo de Trabajo (Operaciones comunes)

#### Actualizar el catálogo

**¡Vital!** Antes de instalar nada, hay que sincronizar el índice local con el servidor remoto.

```
apt update
```

SHELL

#### Actualizar el sistema

Actualiza los programas instalados a su última versión disponible

```
apt upgrade
```

SHELL

- `apt upgrade`: Actualiza paquetes de forma segura. No borra paquetes existentes.



- `apt dist-upgrade` (o `full-upgrade`): Es más agresivo. Si para actualizar el Navegador X necesita borrar la Librería Y, lo hará. Es necesario para cambios grandes de versión de Debian.

## Instalar paquetes

```
apt install firefox
```

SHELL

## Eliminar paquetes

Hay dos formas de borrar:

1. `remove`: Desinstala el programa pero **deja los archivos de configuración** (útil si crees que volverás a instalarlo).

```
apt remove firefox
```

SHELL

2. `purge`: Desinstala el programa y **borra todo rastro** de configuración.

```
apt purge firefox
```

SHELL

## Limpieza

APT guarda los `.deb` descargados en `/var/cache/apt/archives/`. Con el tiempo, esto ocupa mucho espacio.

- `apt autoremove`: Borra dependencias "huérfanas" (librerías que se instalaron para un programa que ya has borrado y que nadie más usa).
- `apt clean`: Borra los instaladores `.deb` descargados de la caché para liberar disco.

## 2.3.4 Gestión de Fuentes

Solo para usuarios avanzados que necesitan modificar el código. Requiere líneas `deb-src` en el `sources.list`

- `apt-get source paquete`: Descarga el código fuente.
- `apt-get build-dep paquete`: Instala automáticamente todas las herramientas y librerías necesarias para compilar ese paquete.
- `apt-get source --compile paquete`: Descarga y compila directamente creando un `.deb`

## 2.3.5 Comando apt-cache

Mientras que **apt-get** se encarga de *instalar* y *modificar*, **apt-cache** es la herramienta de **consulta e investigación**.

Su función es leer la base de datos local (la caché que descargaste con **apt update**) para darte información sobre los paquetes disponibles, **incluso si no los tienes instalados**.

### Buscar Paquetes (**search**)

Funciona como un buscador (Google) dentro de tu lista de software. Busca el texto que le indiques tanto en el **nombre** como en la **descripción** del paquete.

- **Sintaxis:** **apt-cache search [patrón]**
- **Detalle:** Admite expresiones regulares (Regex) para búsquedas avanzadas.

SHELL

```
$ apt-cache search firefox
Salida de ejemplo:
firefox - Mozilla Firefox web browser
firefox-locale-es - Spanish language pack for Firefox
iceweasel - Web browser based on Firefox (Transitional package)
```

### Ficha Técnica del Paquete (**show**)

Muestra toda la información detallada ("metadatos") de un paquete específico.

- **Uso:** Leer la descripción larga, ver quién es el mantenedor, el tamaño de la descarga o la suma de verificación (SHA256).

SHELL

```
apt-cache show nombre_paquete
```

### Ver Dependencias (**depends**)

Muestra el árbol de relaciones del paquete. Es útil para saber **qué más se instalará** si decides instalar ese paquete.

- **Depends:** Lo que es obligatorio instalar.
- **Suggests:** Lo que te sugiere instalar (opcional).
- **Conflicts:** Con qué programas es incompatible.

SHELL

```
$ apt-cache depends nano
nano
Depends: libc6
Depends: libncursesw6
Depends: libtinfo6
Suggests: spell
```

## #### Política de Versiones (**policy**)

Este es el comando más técnico y útil para diagnosticar problemas. Te dice **de dónde** viene un paquete y **qué versión** tiene prioridad.

Muestra tres datos clave:

1. **Installed:** La versión que tienes instalada ahora (o **(none)** si no lo tienes).
2. **Candidate:** La versión que se instalaría si ejecutaras **install** ahora mismo.
3. **Version Table:** La lista de todos los repositorios que ofrecen este paquete y su prioridad

SHELL

```
$ apt-cache policy firefox
firefox:
 Installed: (none)
 Candidate: 115.0-2
 Version Table:
 115.0-2 500
 500 http://deb.debian.org/debian stable/main amd64 Packages
```

## 2.3.6 Solución de Problemas

### Dependencias Rotas

Si una instalación falla a medias o se va la luz, el sistema de paquetes puede quedar "roto".

SHELL

```
El comando mágico para arreglarlo
apt --fix-broken install
(o su versión antigua: apt-get -f install)
```

Esto intenta descargar lo que falta y configurar lo que quedó a medias.

## El fichero de bloqueo (Lock File)

APT usa un archivo "cerrojo" en `/var/lib/dpkg/lock` para asegurar que **solo una persona instala cosas a la vez**.

**Síntoma:** Intentas usar apt y dice: `E: Could not get lock /var/lib/dpkg/lock - open (11: Resource temporarily unavailable)`

### Causas:

1. Tienes otra terminal abierta instalando algo.
2. El sistema de actualizaciones automáticas (`unattended-upgrades`) está trabajando en segundo plano.
3. Se fue la luz durante una instalación y el fichero no se borró.

### Solución:

1. Esperar (lo más recomendable).
2. Verificar si hay procesos apt corriendo: `ps aux | grep apt`.
3. (Último recurso) Borrar el fichero manualmente si estás 100% seguro de que nada está corriendo

## 2.4 Instalación desde el código fuente

Este método consiste en descargar el código "crudo" creado por el programador y transformarlo (compilarlo) en un programa ejecutable adaptado específicamente a tu ordenador.

### 2.4.1 Flujo de trabajo estándar

#### Descarga del código (Tarballs)

El código suele venir empaquetado y comprimido para ocupar menos espacio. Las extensiones nos dicen cómo tratarlo:

- **.tar:** Empaquetado (agrupado en un solo fichero) pero sin comprimir.
- **.tar.gz / .tgz:** Empaquetado y comprimido con Gzip (el más común).
- **.tar.bz2 / .tbz:** Empaquetado y comprimido con Bzip2 (comprime más, tarda más).

#### Desempaquetado

Usamos el comando `tar` (Tape ARchiver).

- Descomprimir .tar.gz: `tar -xzvf archivo.tar.gz`
- Descomprimir .tar.bz2: `tar -xjvf archivo.tar.bz2`

**Nota:** Tras descomprimir, siempre se crea un directorio nuevo. Debemos entrar en él: `cd nombre-directorio`.

## Leer la instrucciones ( `README` / `INSTALL` )

Dentro del directorio siempre hay ficheros de texto con instrucciones vitales. **Léelos antes de seguir.** Te dirán qué dependencias necesitas instalar antes de empezar.

## Configuración ( `.configure` )

Ejecutamos el script `./configure` que viene con el código (generado por herramientas como `autoconf`).

- **¿Qué hace?** Chequea tu sistema. Verifica si tienes el compilador (gcc), si te faltan librerías y si tu entorno es compatible.
- **Resultado:** Si todo va bien, crea un fichero llamado `Makefile`. Este fichero es la "receta" personalizada para tu PC.
- **Personalización:** Puedes decirle dónde instalar el programa usando la opción `--prefix`.

```
./configure --prefix=/home/usuario/mi_programa
```

SHELL

## Compilación ( `make` )

El comando `make` lee el `Makefile` generado en el paso anterior y empieza a compilar.

- `make`: Compila el código (traduce C/C++ a binario). Tarda desde segundos hasta horas.
- `make clean`: Si la compilación falla o quieres empezar de cero, esto borra todos los archivos temporales creados.

## ### Instalación ( `make install` )

Copia los binarios finales y las librerías a sus destinos definitivos (normalmente `/usr/bin`, `/usr/lib`, etc.).

- **Permisos:** Si instalas en directorios del sistema, necesitas `sudo make install`. Si usaste `--prefix` a tu carpeta, no hace falta.

## 2.4.2 Tipos de Ejecutables y Librerías

Una vez compilado el programa, obtenemos un **binario**. Dependiendo de cómo se haya "enlazado" (linked), puede ser de dos tipos:

| Tipo                         | Descripción                                                                                                      | Ventajas/Desventajas                                                                                                                                                               |
|------------------------------|------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Estático</b><br>(Static)  | El ejecutable lleva <b>todas</b> las librerías que necesita metidas dentro de sí mismo. Es un bloque único.      | <div>✓ Funciona en cualquier sistema.</div> <div>✗ Ocupa mucho espacio en disco y RAM.</div> <div>✗ Difícil de actualizar (hay que recompilar todo).</div>                         |
| <b>Dinámico</b><br>(Dynamic) | El ejecutable es pequeño. Cuando se inicia, pide al sistema operativo las librerías que necesita ( <b>.so</b> ). | <div>✓ Ocupa muy poco espacio.</div> <div>✓ Si actualizas una librería del sistema, todos los programas mejoran.</div> <div>✗ Si falta una librería, el programa no arranca.</div> |

### 2.4.3 Gestión de Librerías Dinámicas (**ldd**)

Como los programas dinámicos dependen de archivos externos, a veces fallan porque no los encuentran. Para diagnosticar esto usamos **ldd** (List Dynamic Dependencies).

#### Verificar dependencias

El comando muestra qué librerías pide el programa y dónde las ha encontrado el sistema.

**Formato:** **librería\_necesaria => librería\_encontrada (dirección\_memoria)**

```
$ ldd /bin/ls
 libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x0000...)
 libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x0000...)
 ...

Problemas comunes ("Not found")
Si ves esto, el programa no arrancará:
```shell
libgdal.so.20 => Not found
```

SHELL

Posibles causas y soluciones:

1. **La librería no está instalada:** Tienes que instalar el paquete que la contiene (usando `apt`).
2. **La librería está en una ruta rara:** El sistema busca en `/lib` y `/usr/lib`. Si tu librería está en `/opt/mi_app/lib`, el sistema no la ve.
 - *Solución:* Añadir esa ruta a la variable de entorno `LD_LIBRARY_PATH`.
 - *Ejemplo:* `export LD_LIBRARY_PATH=/opt/mi_app/lib:$LD_LIBRARY_PATH`
3. **Error de Versión (Version Mismatch):** El programa pide `lib.so.3` y tú tienes `lib.so.4`.
 - *Solución "sucia" (Symlink):* Engañar al programa creando un enlace simbólico.

SHELL

```
# Hacemos creer al sistema que la versión 4 es la 3
ln -s libgda1.so.30.0.4 libgda1.so.30.0.3
```

El cargador dinámico (`ld.so`)

Es el programa "invisible" del sistema operativo que se ejecuta cada vez que abres una aplicación. Su trabajo es leer la lista de dependencias del ejecutable, buscar esos archivos `.so` en el disco duro y cargarlos en la RAM para que el programa funcione.