# Flatpak Documentation

*Versión*

**Flatpak Team**

**21 de febrero de 2018**

# Índice general


**1. Contenido** **3**

1.1. Introducción a Flatpak . . . 3
1.2. Elementos de una aplicación Flatpak . . . 6
1.3. Using Flatpak . . . 7
1.4. Building Your First Flatpak . . . 9
1.5. Preparación . . . 12
1.6. Construcción de apps simples . . . 12
1.7. Flatpak Builder . . . 14
1.8. Trabajar con el Sandbox . . . 16
1.9. Distribución de aplicaciones . . . 19
1.10. Available Runtimes . . . 21
1.11. Referencia de comandos . . . 22
1.12. Flatpak-builder Command Reference . . . 22

Estos documentos cubren todo lo que necesita saber para construir y distribuir aplicaciones como Flatpaks. Contienen una introducción de alto nivel a Flatpak, tutoriales, e información esencial sobre cómo desarrollar, construir, y distribuir aplicaciones.

Los documentos están destinados principalmente a desarrolladores de aplicaciones y distribuidores. Su contenido también es relevante a aquellos que tengan un interés general en Flatpak.

Si está buscando información sobre como instalar y usar aplicaciones Flatpak, diríjase al sitio web de Flatpak.

CAPÍTULO 1

# Contenido

## 1.1 Introducción a Flatpak

Flatpak es una tecnología para construir, distribuir, instalar, y lanzar aplicaciones. Está orientada principalmente al escritorio Linux, aunque también puede ser usada como base para la distribución de aplicaciones en otros contextos, tales como sistemas embebidos.

Flatpak ha sido diseñado e implementado con una cantidad de objetivos:

- Permitir que las aplicaciones puedan ser instaladas en cualquier distribución Linux.
- Proporcionar entornos consistentes a las aplicaciones, para facilitar las pruebas y reducir errores.
- Desacoplar las aplicaciones del sistema operativo, para que no dependan de versiones específicas de cada distribución.
- Permitir que las aplicaciones agrupen sus propias dependencias, de forma que puedan usar bibliotecas que no estén provistas por una distribución de Linux, y así puedan depender de versiones específicas o incluso versiones emparchadas de alguna biblioteca.
- Incrementar la seguridad de los escritorios Linux, aislando aplicaciones en *sandboxes*.

Flatpak hace que sea fácil sacar partido de estas características. Si aún no lo ha hecho, se recomienda que pruebe el ejemplo de hola mundo como una forma de ponerse en marcha.

Puede encontrar más información sobre Flatpak en flatpak.org.

### 1.1.1 Cómo funciona

Flatpak puede entenderse a través de un pequeño número de conceptos clave. Éstos también ayudan a explicar la diferencia con los paquetes de software tradicional.

#### *Runtimes*

Los *runtimes* proporcionan las dependencias básicas que las aplicaciones usan. Hay varios runtimes disponibles, desde el más mínimo (pero más estable) Freedesktop, a runtimes más grandes producidos por escritorios como GNOME o KDE. La página de runtimes en flatpak.org proporciona una vista general de los runtimes disponibles actualmente.

Cada aplicación debe ser construida contra un runtime, y este runtime debe ser instalado en el sistema *host* para que la aplicación se pueda lanzar. Los usuarios pueden tener instalados múltiples runtimes diferentes al mismo tiempo, inclusive distintas versiones del mismo runtime.

---

**Truco:** Cada runtime puede pensarse como un sistema de archivos `/usr`. De hecho, cuando se lanza una aplicación, su runtime está montado en `/usr`.

---

#### Bibliotecas agrupadas

Si una aplicación requiere alguna dependencia que no esté en su runtime, puede ser agrupada junto con la aplicación misma. Esto permite que las aplicaciones usen dependencias que no están disponibles en la distribución, o que usen una dependencia en una versión distinta de la instalada en el host.

#### SDKs (Kits de desarrollo de software)

Un SDK es un runtime que incluye las partes "devel" que no son necesarias al momento de lanzar la aplicación, tales como herramientas de construcción y empaquetado, archivos de cabecera, compiladores y *debuggers*. Cada aplicación se construye contra un SDK, que está apareado a un runtime. Este es el runtime que será usado por la aplicación al ser lanzada.

#### Extensiones

Una extensión es un agregado opcional a un runtime o aplicación. Son más comúnmente usados para separar de los runtimes a las traducciones y a la información de *debug*. Por ejemplo, `org.freedesktop.Platform.Locale` puede agregarse al runtime `org.freedesktop.Platform` para sumar traducciones.

#### *Sandboxes*

Con Flatpak, cada app es construida y lanzada en un entorno aislado. Por defecto, la aplicación sólo puede «ver» dos cosas: a sí misma y a su runtime. Se debe garantizar explícitamente el acceso a los archivos de usuario, la red, los *sockets* gráficos, subsistemas del bus y dispositivos. Como se describirá más adelante, Flatpak proporciona varias formas de hacerlo. El acceso a otras cosas, como por ejemplo a otros procesos, es deliberadamente imposible.

### 1.1.2 El comando flatpak

`flatpak` es el comando que se usa para instalar, desinstalar y actualizar runtimes y aplicaciones. También puede usarse para ver qué está instalado actualmente, y tiene comandos para construir y distribuir paquetes de aplicaciones. `flatpak --help` proporciona una lista completa de los comandos dispobibles.

La mayoría de los comandos flatpak son realizados a nivel de sistema por defecto. Para realizar un comando sólo para el usuario actual, use la opción `--user`. Esto permite, por ejemplo, que los runtimes y los paquetes de aplicaciones sean instaladas por cada usuario.

Para más información sobre los comandos de flatpak, vea la *Referencia de comandos*

### 1.1.3 Identificadores

Flatpak identifica cada aplicación, runtime y SDK usando un nombre único, que a veces se emplea como parte de una tripleta nombre/arquitectura/rama.

#### Nombre

Los nombres de Flatpak toman la forma de una dirección de DNS inversa, tal como `com.compañia.App`. El segmento final de esta dirección es el nombre del objeto, y la parte precedente es el dominio al que pertenece. Para poder prevenir conflictos entre nombres, este dominio debe corresponder a una dirección de DNS registrada. Esto significa que se debe usar el dominio de un sitio web, ya sea de una aplicación o de una organización. Por ejemplo, si la aplicación `App` tiene su propio sitio en `app.com`, su nombre Flatpak sería `com.app.App`. Múltiples aplicaciones pueden pertenecer al mismo dominio, tales como `org.organización.App1` y `org.organización.App2`.

Si no cuenta con un dominio registrado para su aplicación, se puede usar un sitio web de terceros para obtener uno. Por ejemplo, Github permite la creación de páginas personales que pueden usarse para este propósito. En este caso, el espacio de nombres `nombre.github.io` puede usarse como la base de un identificador de aplicación `io.github.nombre.App`.

Si la aplicación proporciona un servicio de D-Bus, se espera que el nombre del servicio de D-Bus sea el mismo que el nombre de la aplicación.

#### Tripleta identificadora

Muchos comandos de flatpak sólo requieren el nombre de una aplicación, runtime o SDK. Sin embargo, en algunas circunstancias también es necesario especificar la arquitectura y la rama. Las ramas permiten especificar una versión en particular. Esto se hace usando la tripleta nombre/arquitectura/rama. Por ejemplo: `org.gnome.Sdk/x86_64/3.14` o `org.gnome.Builder/i386/master`.

### 1.1.4 Bajo la alfombra

Flatpak usa una cantidad de tecnologías preexistentes. En general no es necesario estar familiarizado con ellas para poder usar Flatpak, pero en algunos casos esto podría ser útil. Incluyen:

- La utilidad bubblewrap del Project Atomic, que hace que los usuarios sin privilegios puedan configurar y correr contenedores, usando características del *kernel* tales como:
  - Cgroups
  - Namespaces
  - Bind mounts
  - Seccomp rules
- systemd para configurar cgroups para sandboxes
- D-Bus, una forma bien establecida de proporcionar APIs de alto nivel a las aplicaciones.
- El formato OCI de la Open Container Initiative, como un formato conveniente de transporte para paquetes de un solo archivo.
- El sistema OSTree para el versionado y la distribución de árboles de archivos de sistema.
- Metadatos Appstream, para permitir que las aplicaciones flatpak se vean bien en aplicaciones administradoras de software.

## 1.2 Elementos de una aplicación Flatpak

Flatpak espera que las aplicaciones sigan las convenciones del estándar del escritorio Linux. Éstas son suplementadas con una pequeña cantidad de elementos específicos de Flatpak que se usan para distribuir, instalar y lanzar las aplicaciones.

### 1.2.1 Elementos estándar de la aplicación

Las siguientes son algunas de las convenciones del escritorio Linux consideradas y esperadas por Flatpak. Se anima a los desarrolladores de aplicaciones a usarlas.

- AppData, para proporcionar información sobre la aplicación, tal como descripciones y pantallazos, que son usados por las tiendas de apps.
- Iconos de aplicación, especificados por la Especificación de temas de iconos.
- D-Bus, para interacciones con el host.
- Archivos de escritorio, para proporcionar información básica sobre la aplicación.
- PulseAudio, para sonido.
- X11 y Wayland, para la visualización.

### 1.2.2 Estructura de la aplicación

Cuando una aplicación se construye usando flatpak, la salida cuenta con la siguiente estructura:

- `metadata` - Un archivo clave que proporciona información sobre la aplicación.
- `/files` - Los archivos que hacen a la aplicación, incluye el código fuente y los datos.
- `/files/bin` - Binarios de la aplicación.
- `/export` - Archivos a los que el entorno del host necesita acceder, tales como la AppData de la aplicación, el archivo .desktop, icono y archivos de servicios de D-Bus.

Todos los archivos en el directorio de exportación deben tener el ID de la aplicación como prefijo. Por ejemplo:

- `org.gnome.App.appdata.xml`
- `org.gnome.App.desktop`
- `org.gnome.App.png`
- `org.gnome.App.service`

Nombrar a los archivos de esta forma previene conflictos de nombres y asegura que no se sobreescriban las aplicaciones instaladas en el sistema.

Para nombrar archivos exportados de esta forma, se puede renombrar los archivos fuente relevantes, o usar flatpak-builder para renombrar los archivos durante la construcción. Esto último se explica en más detalle en la sección sobre flatpak-builder.

### 1.2.3 Archivos de metadatos

El archivo `metadata` de una aplicación proporciona información que permite a flatpak configurar el sandbox para lanzar la aplicación. Un archivo metadata típico luce así:

```
[Application]
name=org.gnome.gedit
runtime=org.gnome.Platform/x86_64/3.22
sdk=org.gnome.Sdk/x86_64/3.22
command=gedit

[Context]
shared=ipc;network;
sockets=x11;wayland;pulseaudio;
devices=dri;
filesystems=host;

[Environment]
GEDIT_FOO=bar

[Session Bus Policy]
org.extra.name=talk
org.other.name=own
```

Esto especifica el nombre de la aplicación, el runtime que requiere, el SDK con el que fue construida y el comando que se usa para lanzarla. También especifica el acceso a archivos y dispositivos, asigna variables de entorno (por supuesto, dentro del sandbox de la aplicación), y cómo se conecta con el bus de sesión. En las siguientes secciones se incluyen detalles sobre como cambiar estos parámetros de los metadatos.

---

**Nota:** Si bien es más común encontrar archivos de metadatos para las aplicaciones, los runtimes y las extensiones también los tienen.

---

## 1.3 Using Flatpak

This page provides an introduction to the most common commands needed to use Flatpak. It is not intended to be exhaustive or to cover all the options for each command (a full list of all the commands and their options can be found in the command reference).

---

**Nota:** Flatpak commands can be run either per-user or system-wide. All the examples in this guide use the default system-wide behavior.

---

### 1.3.1 Remotes

Remotes are the repositories from which applications and runtimes can be installed.

#### List remotes

To list the remotes that you have configured on your system, run:

```
$ flatpak remotes
```

This gives a list of the existing remotes that have been added. The list indicates whether each remote has been added per-user or system-wide.

#### Add a remote

Adding a remote allows you to search and list its contents, and to install applications from it. The most convenient way to add a remote is by using a `.flatpakrepo` file, which includes both the details of the remote and its GPG key:

```
$ flatpak remote-add --if-not-exists flathub https://dl.flathub.org/repo/flathub.
↪flatpakrepo
```

Here, `flathub` is the local name that is given to the remote. The URL points to the remote's `.flatpakrepo` file. `--if-not-exists` stops the command from producing an error if the remote already exists.

#### Remove a remote

To remove a remote, run:

```
$ flatpak remote-delete flathub
```

In this case, `flathub` is the remote's local name.

### 1.3.2 Installing applications

#### Search

Applications can be found in any of your remotes using the `search` command. For example:

```
$ flatpak search gimp
```

Search will return any applications matching the search terms. Each search result includes the application ID and the remote that the application is in. In this example, the search term is `gimp`.

---

**Nota:** Search will only work for remotes whose application metadata has been updated. This can be done by either running `flatpak update` or `flatpak update --appstream`.

---

#### Install applications

To install an application, run:

```
$ flatpak install flathub org.gimp.GIMP
```

Here, `flathub` is the name of the remote the application is to be installed from, and `org.gimp.GIMP` is the ID of the application.

Sometimes, an application will require a particular runtime, and this will be installed prior to the application.

The details of the application to be installed can also be provided by a `.flatpakref` file, which can be either remote or local. To specify a `.flatpakref` instead of manually providing the remote and application ID, run:

```
$ flatpak install https://flathub.org/repo/appstream/org.gimp.GIMP.flatpakref
```

If the `.flatpakref` file specifies that the application is to be installed from a remote that hasn't already been added, you will be asked whether to add it before the application is installed.

#### Running applications

Once an application has been installed, it can be launched using the `run` command and its application ID:

```
$ flatpak run org.gimp.GIMP
```

### 1.3.3 Managing your applications

#### Updating

To update all your installed applications and runtimes to the latest version, run:

```
$ flatpak update
```

#### List installed applications

To list the applications and runtimes you have installed, run:

```
$ flatpak list
```

Alternatively, to just list installed applications, run:

```
$ flatpak list --app
```

#### Remove an application

To remove an application, run:

```
$ flatpak uninstall org.gimp.GIMP
```

## 1.4 Building Your First Flatpak

This tutorial provides a quick introduction to Flatpak. In it, you will learn how to create a basic Flatpak application, which can be installed and run.

To complete this tutorial, you should have installed Flatpak by following the setup guide. You will also need to haves the `flatpak-builder` tool, which is generally available as a package.

### 1.4.1 1. Install a runtime and the matching SDK

Flatpak requires every app to specify a runtime that it uses for its basic dependencies. Each runtime has a matching SDK (Software Development Kit), which contains all the things that are in the runtime, plus headers and development tools (similar to what is typically found in -devel/-dev packages in Linux distributions). This SDK is required to build apps for the runtime.

In this tutorial we will use the Freedesktop runtime version 1.6. This runtime is provided by the Flathub repository. To add this, run:

```
$ flatpak remote-add --if-not-exists flathub https://flathub.org/repo/flathub.
↪flatpakrepo
```

Then, to install the runtime and the SDK, run:

```
$ flatpak install flathub org.freedesktop.Platform//1.6 org.freedesktop.Sdk//1.6
```

### 1.4.2 2. Create the app

The app that is going to be created for this tutorial is a a simple script. To create it, copy the following to an empty file and save it as *hello.sh*:

```
#!/bin/sh
echo "Hello world, from a sandbox"
```

### 1.4.3 3. Add a manifest

Most Flatpaks are built using the *flatpak-builder* tool. This reads a manifest file which describes the key properties of the application and how it is to be built.

To add a manifest to the hello world app, add the following to an empty file:

```
{
    "app-id": "org.flatpak.Hello",
    "runtime": "org.freedesktop.Platform",
    "runtime-version": "1.6",
    "sdk": "org.freedesktop.Sdk",
    "command": "hello.sh",
    "modules": [
        {
            "name": "hello",
            "buildsystem": "simple",
            "build-commands": [
                "install -D hello.sh /app/bin/hello.sh"
            ],
            "sources": [
                {
                    "type": "file",
                    "path": "hello.sh"
                }
            ]
        }
    ]
}
```

Now save the file alongside *hello.sh* and call it *org.flatpak.Hello.json*.

In a more complex application, the manifest would list multiple modules. The last one would typically be the application itself, and the earlier ones would be dependencies that are bundled with the app because they are not part of the runtime.

### 1.4.4 4. Build the application

Now that the app has a manifest, *flatpak-builder* can be used to build it. This is done by specifying the the manifest file and a target directory:

```
$ flatpak-builder app-dir org.flatpak.Hello.json
```

This command will build each module that is listed in the manifest and install it to the */app* subdirectory, inside the *app-dir* directory.

### 1.4.5 5. Test the build

To verify that the build was successful, the following can be run:

```
$ flatpak-builder --run app-dir org.flatpak.Hello.json hello.sh
```

Congratulations, you've made an app!

### 1.4.6 6. Put the app in a repository

Before you can install and run the app, it first needs to be put in a repository. This is done by passing the *–repo* argument to *flatpak-builder*:

```
$ flatpak-builder --repo=repo --force-clean app-dir org.flatpak.Hello.json
```

This does the build again, and at the end exports the result to a local directory called *repo*. Note that *flatpak-builder* keeps a cache of previous builds in the *.flatpak-builder* subdirectory, so doing a second build like this is very fast.

This second time we passed in *–force-clean*, which means that the previously created *app-dir* directory was deleted before the new build was started.

### 1.4.7 7. Install the app

Now we're ready to add the repository that was just created and install the app. This is done with two commands:

```
$ flatpak --user remote-add --no-gpg-verify tutorial-repo repo
$ flatpak --user install tutorial-repo org.flatpak.Hello
```

The first command adds the repository that was created in the previous step. The second command installs the app from the repository.

Both these commands use the *–user* argument, which means that the repository and the app are added per-user rather than system-wide. This is useful for testing.

Note that the repository was added with *–no-gpg-verify*, since a GPG key wasn't specified when the app was built. This is fine for testing, but for official repositories you should sign them with a private GPG key.

### 1.4.8 8. Run the app

All that's left is to try the app. This can be done with the following command:

```
$ flatpak run org.flatpak.Hello
```

This runs the app, so that it prints *Hello world, from a sandbox*.

## 1.5 Preparación

Prepararse para construir flatpaks es rápido y sencillo. Primero, es necesario tener instalados en el sistema los paquetes `flatpak` y `flatpak-builder`. Están disponibles en la mayoría de las distribuciones, y el sitio web de Flatpak proporciona los detalles para obtenerlos.

Una vez que flatpak ha sido instalado, es necesario elegir un runtime e instalarlo, junto con el SDK correspondiente.

### 1.5.1 Instalación de un SDK

Un SDK es un tipo especial de runtime que se usa para construir aplicaciones. Típicamente, un SDK está apareado con un runtime que será usado por la aplicación al momento de ser lanzada. Por ejemplo el SDK GNOME 3.22 SDK se usa para construir aplicaciones que se lanzan sobre el runtime GNOME 3.22.

El sitio de Flatpak proporciona una lista de los runtimes disponibles. Una vez que haya decidido cual usar, prepararlo es sólo una cuestión de instalarlo junto con su SDK.

Los ejemplos en el resto de la documentación de Flatpak usan el runtime y el SDK de GNOME 3.22. Si todaviá no los ha instalado, descargue la clave GPG del repositorio y luego agregue el repositorio que contiene el runtime y el SDK:

```
$ flatpak remote-add --from gnome https://sdk.gnome.org/gnome.flatpakrepo
```

Ahora puede descargar e instalar el runtime y el SDK:

```
$ flatpak install gnome org.gnome.Platform//3.22 org.gnome.Sdk//3.22
```

El mismo procedimiento puede usarse para instalar cualquier otro runtime y SDK.

### 1.5.2 Echando un vistazo

Si esta es la primera vez que usa Flatpak, es un buen momento para intentar instalar una aplicación y echar un vistazo «bajo la alfombra». Para hacer esto, necesita instalar un repositorio que contenga aplicaciones. Podemos hacerlo usando el repositorio gnome-apps para instalar gedit:

```
$ flatpak remote-add --from gnome-apps https://sdk.gnome.org/gnome-apps.flatpakrepo
$ flatpak install gnome-apps org.gnome.gedit
```

Ahora puede usar el siguiente comando para obtener un shell en “devel sandbox”:

```
$ flatpak run --devel --command=bash org.gnome.gedit
```

Esto abre un entorno que tiene al paquete de la aplicación montado en `/app`, y al SDK con el que fue construida montado en `/usr`. Puede explorar estos dos directorios para ver cómo luce un típico flatpak, y también qué incluye el SDK.

## 1.6 Construcción de apps simples

La utilidad `flatpak` proporciona un conjunto simple de comandos para construir y distribuir aplicaciones. Estos permiten crear nuevos Flatpaks, en los que se pueden construir aplicaciones nuevas o existentes.

Esta sección describe cómo construir una aplicación simple que no requiere ninguna dependencia adicional fuera del runtime contra el que es construida. Para poder completar los ejemplos, debería haber completado antes los pasos de la Preparación.

### 1.6.1 Creación de una app

Para crear una aplicación, el primer paso es usar el comando `build-init`. Esto crea un directorio en el cual la aplicación puede ser construida, con la estructura de directorios correcta y el archivo de metadatos que contiene información sobre la app. El formato de build-init es:

```
$ flatpak build-init DIRECTORY APPNAME SDK RUNTIME [BRANCH]
```

- DIRECTORIO es el nombre del directorio que será creado para contener a la aplicación
- NOMBREAPP es el nombre de D-Bus de la aplicación
- SDK es el nombre del SDK que será usado para construir la aplicación
- RUNTIME es el nombre del runtime runtime que será requerido por la aplicaación
- RAMA es típicamente la versión del SDK y del runtime que será usada

Por ejemplo, para construir la aplicación Diccionario de GNOME usando el SDK de GNOME 3.22, el comando se vería así:

```
$ flatpak build-init dictionary org.gnome.Dictionary org.gnome.Sdk org.gnome.Platform␣
↪3.22
```

You can try this command now. In the next step we will build an application inside the resulting `dictionary` directory.

### 1.6.2 Construcción

`flatpak build` se usa para construir una aplicación usando un SDK. Este comando se usa para proporcionar acceso al sandbox. Por ejemplo, el siguiente comando creará un archivo dentro del directorio /app del sandbox, en el directorio files:

```
$ flatpak build dictionary touch /app/some_file
```

Se recomienda eliminar el archivo antes de continuar.

El comando build permite que las aplicaciones existentes que hayan sido hechas con la tradicional rutina configure, make, make install sean construidas dentro de flatpak. Puede probar esto usando el Diccionario de GNOME. Primero, descargue los archivos fuente, extráigalos y posiciónese dentro del directorio creado:

```
$ wget https://download.gnome.org/sources/gnome-dictionary/3.20/gnome-dictionary-3.20.
↪0.tar.xz
$ tar xvf gnome-dictionary-3.20.0.tar.xz
$ cd gnome-dictionary-3.20.0/
```

Luego puede usar el comando build para construir e instalar el código fuente dentro del directorio dictionary que fue creado anteriormente:

```
$ flatpak build ../dictionary ./configure --prefix=/app
$ flatpak build ../dictionary make
$ flatpak build ../dictionary make install
$ cd ..
```

Dado que esto corre en el sandbox, el compilador y otras herramientas del SDK son usadas para construir e instalar, en vez de usarse las del sistema host.

### 1.6.3 Completar la construcción

Una vez que la aplicación fue construida, necesita correr el comando `build-finish` para especificar el acceso a las distintas partes del host, tales como la red y los sockets de gráficos. Este comando también es usado para especificar el comando que será llamado para lanzar la app (esto se hace modificando el archivo metadata), y para crear el directorio de exportación de la aplicación. Por ejemplo:

```
$ flatpak build-finish dictionary --socket=x11 --share=network --command=gnome-
↪dictionary
```

En este punto ya se ha construido exitosamente un flatpak listo para ser lanzado. Para probar el app, necesita exportar el Diccionario a un repositorio. Agregue ese repositorio y luego instale y corra la app:

```
$ flatpak build-export repo dictionary
$ flatpak --user remote-add --no-gpg-verify --if-not-exists tutorial-repo repo
$ flatpak --user install tutorial-repo org.gnome.Dictionary
$ flatpak run org.gnome.Dictionary
```

Esto exporta la app, crea un repositorio llamado tutorial-repo, instala la aplicación Diccionario para este solo usuario, y finalmente lanza la aplicación.

## 1.7 Flatpak Builder

La mayoría de las aplicaciones requieren dependencias adicionales que no están provistas por los runtimes. Flatpak permite que estas dependencias se agrupen como parte de la aplicación misma. Para hacer esto, cada dependencia debe ser construida dentro del directorio build de la aplicación. La herramienta `flatpak-builder` automatiza este proceso que suele tomar múltiples pasos, haciendo posible construir todos los módulos de la aplicación con un solo comando.

flatpak-builder supports a variety of build systems, including autotools, cmake, cmake-ninja, meson, a simple one called «simple» which allows to provide a series of commands to run, and the so called Build API.

Todas las entidades json están explicadas en la página man de `flatpak-builder`.

### 1.7.1 Manifiestos

La entrada de flatpak-builder es un archivo JSON que describe los parámetros para construir una aplicación, así como los de cada módulo a ser agrupado. Este archivo se llama el manifiesto. Las fuentes de los módulos pueden ser de varios tipos, incluyendo archivadores `.tar` o `.zip`, repositorios Git or Bzr, archivos de parches o comandos de shell que se corren.

El manifiesto del Diccionario de GNOME es corto, porque el único módulo que contiene es la aplicación misma:

```
{
  "app-id": "org.gnome.Dictionary",
  "runtime": "org.gnome.Platform",
  "runtime-version": "3.22",
  "sdk": "org.gnome.Sdk",
  "command": "gnome-dictionary",
  "finish-args": [
     "--socket=x11",
     "--share=network"
  ],
  "modules": [
    {
```

```
      "name": "gnome-dictionary",
      "sources": [
        {
          "type": "archive",
          "url": "https://download.gnome.org/sources/gnome-dictionary/3.20/gnome-
↪dictionary-3.20.0.tar.xz",
          "sha256": "efb36377d46eff9291d3b8fec37baab2355f9dc8bc7edb791b6a625574716121"
        }
      ]
    }
  ]
}
```

Como se puede ver, este manifiesto incluye la información básica sobre la aplicación antes de especificar un único archivo .tar a ser descargado y construido. Manifiestos más complejos incluyen una secuencia de módulos.

### 1.7.2 Limpieza

Luego de la construcción, flatpak-builder procede a la fase de limpieza. Esta puede ser usada para eliminar encabezados y documentación de desarrollo, entre otras cosas. Dos propiedades del archivo manifiesto son usadas para esto. Primero, se puede incluir una lista de patrones de nombres de archivos:

```
"cleanup": [ "/include", "/bin/foo-*", "*.a" ]
```

La segunda propiedad es una lista de comandos que se corren durante la fase de limpieza:

```
"cleanup-commands": [ "sed s/foo/bar/ /bin/app.sh" ]
```

Las propiedades de limpieza también pueden asignarse a nivel de módulos, en cuyo caso la comparación se hará sólo para los archivos que fueron creados para ese módulo en particular.

### 1.7.3 Renombrado de archivos

Files that are exported by a flatpak must be prefixed using the application ID. If an application's source files are not named using this convention, flatpak-builder allows them to be renamed as part of the build process. To rename application icons, desktop files and AppData files, use the `rename-icon`, `rename-desktop-file` and `rename-appdata-file` properties.

### 1.7.4 Separando los tantos

By default, flatpak-builder splits off translations and debug information into separate *.Locale* and *.Debug* extensions. These "standard" extension points are then added to the application's metadata file. You can turn this off with the `separate-locales` and `no-debuginfo` keys, but there shouldn't be any reason for it.

When flatpak-builder exports the build into a repository, it automatically includes the *.Locale* and *.Debug* extensions. If you do the exporting manually, don't forget to include them.

### 1.7.5 Ejemplo

Para probar flatpak-builder usted mismo, cree un archivo llamado `org.gnome.Dictionary.json` y pegue el manifiesto JSON del Diccionario de arriba. Luego corra el siguiente comando:

```
$ flatpak-builder --repo=repo dictionary2 org.gnome.Dictionary.json
```

Esto hará lo siguiente:

- Crear un nuevo directorio llamado dictionary2 (equivalente a usar *flatpak build-init*).
- Descargar y verificar el código fuente de Dictionario.
- Construir e instalar el código fuente, usando el SDK en lugar del sistema host.
- Finalizar la construcción, asignando permisos (en este caso dando acceso a X y a la red).
- Crear un nuevo repositorio llamado repo (si aún no existe) y exportar el resultado de la construcción dentro del mismo.

flatpak-builder will also do some other useful things, like creating a separately installable debug runtime (called `org.gnome.Dictionary.Debug` in this case) and a separately installable translation runtime (called `org.gnome.Dictionary.Locale`).

Si ha completado el tutorial en Construcción de apps simples, puede actualizar la aplicación instalada con la nueva versión que acaba de construir y exportar con flatpak-builder:

```
$ flatpak --user update org.gnome.Dictionary
```

Otherwise, you need to add the repo to Flatpak and install the application. To do so:

```
$ flatpak --user remote-add --no-gpg-verify --if-not-exists tutorial-repo repo
$ flatpak --user install tutorial-repo org.gnome.Dictionary
```

Para comprobar que la aplicación ha sido actualizada con éxito, puede comparar el commit sha256 de la aplicación instalada con el ID del commit que imprimió flatpak-builder:

```
$ flatpak info org.gnome.Dictionary
$ flatpak info org.gnome.Dictionary.Locale
```

Y finalmente, puede correr la nueva versión del app Diccionario:

```
$ flatpak run org.gnome.Dictionary
```

### 1.7.6 Manifiestos de ejemplo

Hay disponible un manifiesto completo para el Diccionario de GNOME construido desde Git, además de manifiestos para un amplio rango de otras aplicaciones de GNOME.

## 1.8 Trabajar con el Sandbox

Uno de los objetivos principales de Flatpak es incrementar la seguridad en los sistemas de escritorio, aislando a las aplicaciones de las demás. Esto se logra usando sandboxing y significa que, por defecto, un Flatpak tiene un acceso extremadamente limitado al entorno del host. Esto incluye:

- Sin acceso a ningún archivo del host con excepción del runtime, de la app y de `~/.var/app/$APPID`. Sólo se permite acceso de escritura al último de estos.
- Sin acceso a la red.
- Sin acceso a ningún nodo de dispositivo (aparte de `/dev/null`, etc).

- Sin acceso a los procesos por fuera del sandbox.
- Llamadas al sistema limitadas. Por ejemplo, las apps no pueden usar tipos de socket de red no estándar, o hacer ptrace de otros procesos.
- Acceso limitado a la instancia del D-Bus de sesión - una app puede ser dueña sólo su propio nombre en el bus.
- Sin acceso a los servicios del host como X, el D-Bus del sistema, o PulseAudio.

La mayoría de las aplicaciones necesitarán acceso a algunos de estos recursos para poder ser usadas, y Flatpak proporciona una cantidad de formas de dar a una aplicación acceso a los mismos.

Si bien no hay restricciones a cuáles permisos del sandbox una aplicación puede acceder, como buena práctica, se recomienda usar el mínimo número de permisos posible. Ciertos permisos, tales como el acceso total al bus del sistema (usando la opción `--socket=system-bus` ) deberían ser evitados a toda costa.

### 1.8.1 Configurar los permisos del sandbox

Usar el comando `build-finish` es la manera más simple de configurar los permisos del sandbox. Como se vio en un ejemplo anterior, esto puede ser usado para dar acceso a los sockets de gráficos y a la red:

```
$ flatpak build-finish dictionary2 --socket=x11 --share=network --command=gnome-
↪dictionary
```

Estos argumentos se traducen a varias propiedades dentro del archivo de metadatos de la aplicación:

```
[Application]
name=org.gnome.Dictionary
runtime=org.gnome.Platform/x86_64/3.22
sdk=org.gnome.Sdk/x86_64/3.22
command=gnome-dictionary

[Context]
shared=network;
sockets=x11;
```

build-finish permite agregar un amplio rango de recursos en una aplicación. Estas opciones también pueden pasarse a flatpak-builder como propiedades `finish-args`.

La tabla de abajo proporciona una vista general de varios permisos del sandbox. La lista completa también puede verse usando `flatpak build-finish --help`.

---

**Nota:** Hasta que exista un backend compatible con el sandbox, el acceso a dconf necesita activarse usando las siguientes opciones:

```
--filesystem=xdg-run/dconf
--filesystem=~/.config/dconf:ro
--talk-name=ca.desrt.dconf
--env=DCONF_USER_CONFIG_DIR=.config/dconf
```

---

### 1.8.2 Portales

Los portales son un mecanismo a través del cual las aplicaciones pueden interactuar con el entorno del host desde adentro de un sandbox. De esta forma, se dan utilidades adicionales para interactuar con datos, archivos y servicios sin necesidad de agregar permisos de sandbox.

Los *toolkits* de interfaces pueden implementar soporte para portales. Si una aplicación usa uno de estos toolkits, no se requiere ningún trabajo adicional para acceder a los mismos.

Ejemplos de capacidades a las que se puede acceder a través de portales incluyen:

- Inhabilitar la sesión del usuario para cerrarla, suspenderla, o cambiar de usuario.
- Información del estado de la red
- Notificaciones
- Abrir una URI
- Abrir archivos con un diálogo selector de archivos nativo
- Imprimir
- Tomar capturas de pantalla

Las aplicaciones que no usen un toolkit con soporte para portales pueden dirigirse a la documentación de la API de xdg-desktop-portal para información de cómo acceder a los mismos.

### 1.8.3 Sobreescribir los permisos del sandbox

Cuando se desarrolla una aplicación, puede ser útil sobreescribir la configuración de un sandbox de Flatpak. Hay varias maneras de hacerlo. Una es sobreescribirla usando `flatpak run`, que acepta los mismos parámetros que `build-finish`. Por ejemplo, esto hará que la aplicación Diccionario vea su directorio home:

```
$ flatpak run --filesystem=home --command=ls org.gnome.Dictionary ~/
```

También puede usarse `flatpak override` para sobreescribir permanentemente los permisos de una aplicación:

```
$ flatpak --user override --filesystem=home org.gnome.Dictionary
$ flatpak run --command=ls org.gnome.Dictionary ~/
```

Además es posible quitar permisos usando el mismo método. Puede usar el siguiente comando para ver qué ocurre si se elimina el acceso al sistema de archivos, por ejemplo:

```
$ flatpak run --nofilesystem=home --command=ls org.gnome.Dictionary ~/
```

### 1.8.4 Permisos útiles del sandbox

Flatpak proporciona una lista de opciones para controlar los permisos del sandbox. Los siguientes son algunos de los más útiles:

| | |
|---|---|
| `--filesystem=host` | Acceso a todos los archivos |
| `--filesystem=home` | Acceso al directorio home |
| `--filesystem=home:ro` | Acceso al directorio home, de sólo lectura |
| `--filesystem=/some/dir --filesystem=~/other/dir` | Acceso a directorios específicos |
| `--filesystem=xdg-download` | Acceso al directorio Descargas de XDG |
| `--nofilesystem=...` | Hacer una excepción para alguna de las anteriores |
| `--socket=x11 --share=ipc` | Mostrar ventanas usando X11[1] |
| `--device=dri` | Renderizado OpenGL |
| `--socket=wayland` | Mostrar ventanas usando Wayland |
| `--socket=pulseaudio` | Reproducir sonidos usando PulseAudio |
| `--share=network` | Acceso a la red[2] |
| `--talk-name=org.freedesktop.secrets` | Comunicarse con un servicio por nombre en el bus de sesión |
| `--system-talk-name=org.freedesktop.GeoClue2` | Comunicarse con un servicio por nombre en el bus del sistema |
| `--socket=system-bus` | Acceso ilimitado a todos los D-Bus |

#### Notas al pie

## 1.9 Distribución de aplicaciones

Flatpak proporciona varias formas de distribuir aplicaciones. El método principal es alojando un repositorio. Esto es relativamente simple, aunque hay algunos detalles importantes a tener en cuenta, y permite que se distribuyan actualizaciones de las aplicaciones.

También es posible distribuir Flatpaks como un solo archivo autocontenido, lo que puede ser útil en algunas situaciones.

### 1.9.1 Hospedar un repositorio

Las secciones anteriores de esta guía describen cómo generar repositorios usando `build-export` o `flatpak-builder`. El repositorio OSTree resultante puede ser hospedado en un servidor web para consumo de los usuarios.

#### Detalles importantes

Los repositorios OSTree usan archive-z2, lo que significa que contienen un solo archivo por cada archivo de la aplicación. Esto implica que las operaciones *pull* harán una gran cantidad de peticiones HTTP. Dado que las nuevas peticiones son lentas, es importante activar el *keep-alive* de HTTP en el servidor web.

OSTree tiene una característica llamada deltas estáticos. Estos son archivos únicos en el repositorio que contienen todos los datos necesarios para pasar de una revisión a otra (o de nada a una revisión). La creación de estos deltas tomará más espacio en el servidor, pero hará que las descargas sean mucho más rápidas. Esto puede hacerse con la opción `build-update-repo --generate-static-deltas`.

[1] `--share=ipc` means that the sandbox shares IPC namespace with the host. This is not necessarily required, but without it the X shared memory extension will not work, which is very bad for X performance.

[2] Dar acceso a la red también garantiza acceso a todos los servicios que escuchan en sockets abstractos de Unix (dada la forma en que los espacios de nombre de red funcionan), y estos no tienen chequeos de permisos. Esto desafortunadamente afecta por ej. al servidor X y al bus de la sesión, quienes escuchan en sockets abstractos por defecto. Una distribución segura debería desactivarlos y usar sólo sockets regulares.

### Firmas GPG

OSTree usa GPG para verificar la identidad de los repositorios. Esto requiere que todos los commits a un repositorio usen una firma GPG, así también como cuando se modifique el archivo sumario de un repositorio.

Para hacerlo, se debe pasar una clave GPG a los comandos `build-update-repo` y `build-export`, así como también a `flatpak-builder` si está siendo usado para modificar o crear un repositorio. Si todavía no tiene una clave, es fácil generar una.) Por ejemplo:

```
$ flatpak build-export --gpg-sign=KEYID --gpg-homedir=PATH REPOSITORY DIRECTORY
```

Aquí `--gpg-homedir` es opcional, y permite especificar el directorio home de la clave a ser usada.

Si bien en general no se recomienda, es posible desactivar la verificación GPG de los repositorios OSTree. Para hacerlo, la opción `--no-gpg-verify` se puede usar cuando se agregue un remoto. La verificación GPG también puede desactivarse en un remoto existente usando `flatpak remote-modify`.

Notar que es necesario convertirse en root para poder actualizar un remoto que no tenga verificación de GPG activada.

### Referenciar repositorios

Una forma conveniente de dirigir a los usuarios al repositorio que contiene su aplicación es proporcionando un archivo `.flatpakrepo` que puedan descargar e instalar. Para instalar un archivo `.flatpakrepo` manualmente, use el comando:

```
$ flatpak remote-add --from foo.flatpakrepo
```

Un archivo `.flatpakrepo` típico luce así:

```
[Flatpak Repo]
Title=GEdit
Url=http://sdk.gnome.org/repo-apps/
GPGKey=mQENBFUUCGcBCAC/K9WeV4xCaKr3...
```

Si su repositorio contiene una sola aplicación, puede ser más conveniente usar un archivo .flatpakref, que contenga la información suficiente para agregar el repositorio e instalar la aplicación al mismo tiempo. Para instalar un `.flatpakref` manualmente, usar el comando:

```
$ flatpak install --from foo.flatpakref
```

Un archivo `.flatpakref` típico luce así:

```
[Flatpak Ref]
Title=GEdit
Name=org.gnome.gedit
Branch=stable
Url=http://sdk.gnome.org/repo-apps/
IsRuntime=False
GPGKey=mQENBFUUCGcBCAC/K9WeV4xCaKr3...
RuntimeRepo=https://sdk.gnome.org/gnome.flatpakrepo
```

Notar que la clave GPGKey en estos archivos contienen la clave GPG codificada en base64, la cual puede obtener con el siguiente comando:

```
$ base64 --wrap=0 < foo.gpg
```

### 1.9.2 Empaquetado en un solo archivo

Hospedar un repositorio es la forma preferida de distribuir una aplicación, pero a veces un solo archivo autocontenido que se pueda hacer disponible en un sitio web o enviado adjunto en un email es más conveniente. Flatpak brinda capacidad para esto con los comandos `build-bundle` y `build-import-bundle` que convierten una aplicación en un repositorio en un archivo autocontenido y viceversa:

```
$ flatpak build-bundle [OPTION...] LOCATION FILENAME NAME [BRANCH]
$ flatpak build-import-bundle [OPTION...] LOCATION FILENAME
```

Por ejemplo, para crear un archivo autocontenido llamado *dictionary.flatpak* que contenga a la aplicación Diccionario de GNOME desde un repositorio ubicado en ~/repositories/apps, correr:

```
$ flatpak build-bundle ~/repositories/apps dictionary.flatpak org.gnome.Dictionary
```

Para importar el autocontenido en un repositorio de otra máquina, correr:

```
$ flatpak build-import-bundle ~/my-apps dictionary.flatpak
```

Notar que los autocontenidos tienen algunas desventajas comparados con los repositorios. Por ejemplo, es mucho más conveniente distribuir actualizaciones con un repositorio hospedado, dado que los usuarios simplemente necesitan correr `flatpak update`.

## 1.10 Available Runtimes

This page provides information about available Flatpak runtimes. It is primarily intended as information for application developers and distributors.

There are currently three main runtimes available: Freedesktop, GNOME and KDE. These are all hosted on Flathub.

### 1.10.1 Freedesktop

The Freedesktop runtime is the standard runtime that can be used for any application and contains a set of essential libraries and services, including D-Bus, GLib, PulseAudio, X11 and Wayland.

Available Freedesktop runtimes:

| ID | Description |
|---|---|
| org.freedesktop.Platform | Runtime |
| org.freedesktop.Platform.Locale | Runtime translations (extension) |
| org.freedesktop.Sdk | SDK |
| org.freedesktop.Sdk.Debug | SDK debug information (extension) |
| org.freedesktop.Sdk.Locale | SDK translations (extension) |
| org.freedesktop.Sdk.Docs | SDK documentation (extension) |

### 1.10.2 GNOME

The GNOME runtime is appropriate for any application that uses the GNOME platform. It is based on the Freedesktop runtime and adds the GNOME platform, including:

- Clutter
- Gjs

- GObject Introspection
- GStreamer
- GVFS
- Libnotify
- Libsecret
- LibSoup
- PyGObject
- Vala
- WebKitGTK

Available GNOME runtimes:

| ID | Description |
|---|---|
| org.gnome.Platform | Runtime |
| org.gnome.Platform.Locale | Runtime translations (extension) |
| org.gnome.Sdk | SDK |
| org.gnome.Sdk.Debug | SDK debug information (extension) |
| org.gnome.Sdk.Locale | SDK translations (extension) |
| org.gnome.Sdk.Docs | SDK documentation (extension) |

### 1.10.3 KDE

The KDE runtime is also based on the Freedesktop runtime and adds Qt and KDE Frameworks. It is appropriate for any application that makes use of the KDE platform and most Qt-based applications.

Available KDE runtimes:

| ID | Description |
|---|---|
| org.kde.Platform | Runtime |
| org.kde.Platform.Locale | Runtime translations (extension) |
| org.kde.Sdk | SDK |
| org.kde.Sdk.Debug | SDK debug information (extension) |
| org.kde.Sdk.Locale | SDK translations (extension) |
| org.kde.Sdk.Docs | SDK documentation (extension) |

## 1.11 Referencia de comandos

## 1.12 Flatpak-builder Command Reference