

apt vs apt-get vs aptitude

Generalmente los usuarios de distribuciones DEB como Debian, Devuan, Ubuntu, Mint, LMDE, etc, nos preguntamos qué diferencia tiene usar el viejo apt-get, o aptitude, o el nuevo apt, y nos resulta difícil encontrar respuestas acertadas. Aquí algunas notas al respecto.

Conceptos previos

La infraestructura de administración de paquetes en Debian es simple y potente, y ha distinguido al proyecto desde sus inicios. En distros basadas en DEB, el gestor de paquetes estándar es `dpkg`, y permite instalar, desinstalar, en fin, administrar paquetes `.deb` en el sistema operativo.

Ahora bien, los paquetes en general cuentan con una serie de dependencias, esto es, paquetes que deben ser instalados previamente para poder instalar y/o usar la aplicación. Estas dependencias se gestionan mediante **APT, el Advanced Package Tool**, una herramienta global de alto nivel que incluye a `dpkg` como backend, y que le agrega las funcionalidades de gestión de repositorios externos para dependencias y actualizaciones.

APT soporta diferentes *front-ends* o aplicaciones que hacen uso de sus servicios (API), y pueden ser para línea de comandos, interfaz gráfica, etc.

En sus inicios los usuarios se acostumbraron a utilizar un front-end llamado `apt-get`. A través de este front-end se pueden instalar paquetes, desinstalar, eliminar, etc. `apt-get` es parte de una constelación de herramientas que incluye a `apt-cache` para las búsquedas, por ejemplo.

No todos los usuarios se sienten cómodos con esta herramienta (bastante primitiva en algunos aspectos). Por ello, desde Debian Jessie (v8.0) y Ubuntu 16.04 se puede utilizar el gestor `apt` en su lugar.

`apt` consolida las características de `apt-get`, `apt-cache`, y otros comandos similares, y simplifica mucho el trabajo con la paquetería DEB mediante opciones y modificadores más amigables.

`apt-get` sigue siendo soportada por la mayoría de las distros DEB (y por ese motivo muchos usuarios no ven motivación en aprender `apt`), y disponer de las dos herramientas, tres con `aptitude`, suele ser motivo de confusión y de consulta por parte de mis alumnos. Aclaro que no es necesaria dicha migración, pero `apt` facilita algunas cosas, y `apt-get` sigue siendo muy útil para otras.

Diferencias en funcionalidades

`apt` no garantiza la compatibilidad hacia atrás con `apt-get`, pero prácticamente todos los comandos tienen su equivalente.

Por su parte, `aptitude` gestiona el marcado de paquetes también, y dispone de una interfaz opcional de curses que a muchos les facilita el trabajo en la terminal.

Veamos ahora algunas equivalencias entre comandos de estos tres front-ends. La palabra **app** hace referencia al nombre de una aplicación en el repositorio de Debian, como por ejemplo *libreoffice*, *firefox* y *vlc*.

<u>Función</u>	<u>apt-get</u>	<u>aptitude</u>	<u>apt</u>
<i>Instalar paquete</i>	apt-get install app	aptitude install app	apt install app
<i>Desinstalar un paquete</i>	apt-get remove app	aptitude remove app	apt remove app
<i>Eliminar un paquete y su configuración</i>	apt-get purge app	aptitude purge app	apt purge app
<i>Actualizar el repositorio</i>	apt-get update	aptitude update	apt upate
<i>Actualizar paquetes (sin eliminar ni reinstalar)</i>	apt-get upgrade	aptitude safe-upgrade	apt upgrade (*)
<i>Actualizar paquetes (eliminando y reinstalando si es necesario)</i>	apt-get dist-upgrade	aptitude full-upgrade	apt full-upgrade
<i>Eliminar dependencias innecesarias</i>	apt-get autoremove	aptitude autoremove	apt autoremove
<i>Buscar pakeutes</i>	apt-cache search app	aptitude search app	apt search app
<i>Mostrar info de un paquete</i>	apt-cache show app	aptitude show app	apt show app
<i>Mostrar estado y candidato de instalación</i>	apt-cache policy app	aptitude policy app	apt policy app
<i>Mostrar las fuentes a repositorios</i>	apt-cache policy	aptitude policy	apt policy
<i>Edición de repositorios fuente</i>	-	-	apt edit-

Listar paquetes por criterio

`dpkg --get-selections
> lista.txt`

`dpkg --get-selections
> lista.txt`

`apt list`

(*) Corresponde a `apt-get upgrade --install new-pkgs`

Errores de diseño en apt-get

`apt` corrige algunos errores de la implementación original de `apt-get`, no bugs, errores de diseño de la aplicación (y sí, tiene un montón de años ya, las cosas mejoran :D). Por ejemplo, además de usar `apt-get` originalmente para editar paquetes Debian, el sistema utilizaba `apt-cache` para mostrar la información de los paquetes. El nuevo `apt` combina ambas funcionalidades en ambos comandos y lo estructura todo mejor. `apt` provee características de la mayoría de los comandos comunes de `apt-get` y `apt-cache`, agregando información muy útil que antes requería opciones adicionales.

Otra mejora se puede ver en el comando `apt-get update`. Este comando, al igual que `aptitude update`, solamente reportan si pueden actualizar las listas de paquetes de los repositorios o no, mientras que `apt update` agrega la cantidad de paquetes que pueden ser actualizados, y sugiere usar `apt list --upgradable` para listar los actualizables.

```
juncotic@juncotic-lubuntu: ~
juncotic@juncotic-lubuntu:~$
juncotic@juncotic-lubuntu:~$ sudo apt-get update          ##### APT-GET
Get:1 http://security.ubuntu.com/ubuntu disco-security InRelease [97,5 kB]
Hit:2 http://archive.ubuntu.com/ubuntu disco InRelease
Hit:3 http://archive.ubuntu.com/ubuntu disco-updates InRelease
Get:4 http://archive.ubuntu.com/ubuntu disco-backports InRelease [88,8 kB]
Fetched 186 kB in 2s (94,4 kB/s)
Reading package lists... Done
juncotic@juncotic-lubuntu:~$
juncotic@juncotic-lubuntu:~$
juncotic@juncotic-lubuntu:~$ sudo apt update             ##### APT
Get:1 http://security.ubuntu.com/ubuntu disco-security InRelease [97,5 kB]
Hit:2 http://archive.ubuntu.com/ubuntu disco InRelease
Hit:3 http://archive.ubuntu.com/ubuntu disco-updates InRelease
Get:4 http://archive.ubuntu.com/ubuntu disco-backports InRelease [88,8 kB]
Fetched 186 kB in 2s (99,8 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
117 packages can be upgraded. Run 'apt list --upgradable' to see them.
juncotic@juncotic-lubuntu:~$
```

Diferencias entre update de apt-get y apt

```
juncotic@juncotic-lubuntu: ~  
  
juncotic@juncotic-lubuntu:~$  
juncotic@juncotic-lubuntu:~$  
juncotic@juncotic-lubuntu:~$ sudo apt list --upgradable  
Listing... Done  
apparmor/disco-updates 2.13.2-9ubuntu6.1 amd64 [upgradable from: 2.13.2-9ubuntu6]  
apt-utils/disco-updates 1.8.1 amd64 [upgradable from: 1.8.0]  
apt/disco-updates 1.8.1 amd64 [upgradable from: 1.8.0]  
aptdaemon/disco-updates 1.1.1+bzr982-0ubuntu21.1 all [upgradable from: 1.1.1+bzr982-0ubuntu21]  
base-files/disco-updates 10.1ubuntu9.1 amd64 [upgradable from: 10.1ubuntu9]  
bash/disco-updates 5.0-3ubuntu1.1 amd64 [upgradable from: 5.0-3ubuntu1]  
console-setup-linux/disco-updates 1.178ubuntu12.1 all [upgradable from: 1.178ubuntu12]  
console-setup/disco-updates 1.178ubuntu12.1 all [upgradable from: 1.178ubuntu12]  
cups-bsd/disco-updates 2.2.10-4ubuntu2 amd64 [upgradable from: 2.2.10-4]  
cups-client/disco-updates 2.2.10-4ubuntu2 amd64 [upgradable from: 2.2.10-4]  
cups-common/disco-updates 2.2.10-4ubuntu2 all [upgradable from: 2.2.10-4]  
cups-core-drivers/disco-updates 2.2.10-4ubuntu2 amd64 [upgradable from: 2.2.10-4]  
cups-daemon/disco-updates 2.2.10-4ubuntu2 amd64 [upgradable from: 2.2.10-4]  
cups-ipp-utils/disco-updates 2.2.10-4ubuntu2 amd64 [upgradable from: 2.2.10-4]  
cups-ppdc/disco-updates 2.2.10-4ubuntu2 amd64 [upgradable from: 2.2.10-4]  
cups-server-common/disco-updates 2.2.10-4ubuntu2 all [upgradable from: 2.2.10-4]  
cups/disco-updates 2.2.10-4ubuntu2 amd64 [upgradable from: 2.2.10-4]  
debconf-i18n/disco-updates 1.5.71ubuntu1 all [upgradable from: 1.5.71]  
debconf/disco-updates 1.5.71ubuntu1 all [upgradable from: 1.5.71]
```

Lista de paquetes: `apt list --upgradable`

Otra mejora de `apt` es que muestra la barra de progreso de instalación en las actualizaciones de software, lo que resulta súper útil para saber cuánto resta el proceso completo:

```
juncotic@juncotic-lubuntu: ~  
  
vlc-plugin-visualization  
112 upgraded, 5 newly installed, 0 to remove and 0 not upgraded.  
Need to get 0 B/315 MB of archives.  
After this operation, 345 MB of additional disk space will be used.  
Do you want to continue? [Y/n] y  
Extracting templates from packages: 100%  
Preconfiguring packages ...  
(Reading database ... 258907 files and directories currently installed.)  
Preparing to unpack .../base-files_10.1ubuntu9.1_amd64.deb ...  
Warning: Stopping motd-news.service, but it can still be activated by:  
  motd-news.timer  
Unpacking base-files (10.1ubuntu9.1) over (10.1ubuntu9) ...  
Setting up base-files (10.1ubuntu9.1) ...  
motd-news.service is a disabled or a static unit, not starting it.  
(Reading database ... 258907 files and directories currently installed.)  
Preparing to unpack .../bash_5.0-3ubuntu1.1_amd64.deb ...  
Unpacking bash (5.0-3ubuntu1.1) over (5.0-3ubuntu1) ...  
Setting up bash (5.0-3ubuntu1.1) ...  
update-alternatives: using /usr/share/man/man7/bash-builtins.7.gz to provide /usr/share/man/man7/builtins.7.gz (builtins.7.gz) in auto mode  
(Reading database ... 258907 files and directories currently installed.)  
Preparing to unpack .../dpkg_1.19.6ubuntu1.1_amd64.deb ...  
Unpacking dpkg (1.19.6ubuntu1.1) over (1.19.6ubuntu1) ...  
  
Progress: [ 2%] [#.....]
```

`apt upgrade` y la barra de progreso

Esta barra de progreso también aparece cuando eliminamos paquetes con `apt remove` o `apt purge`.

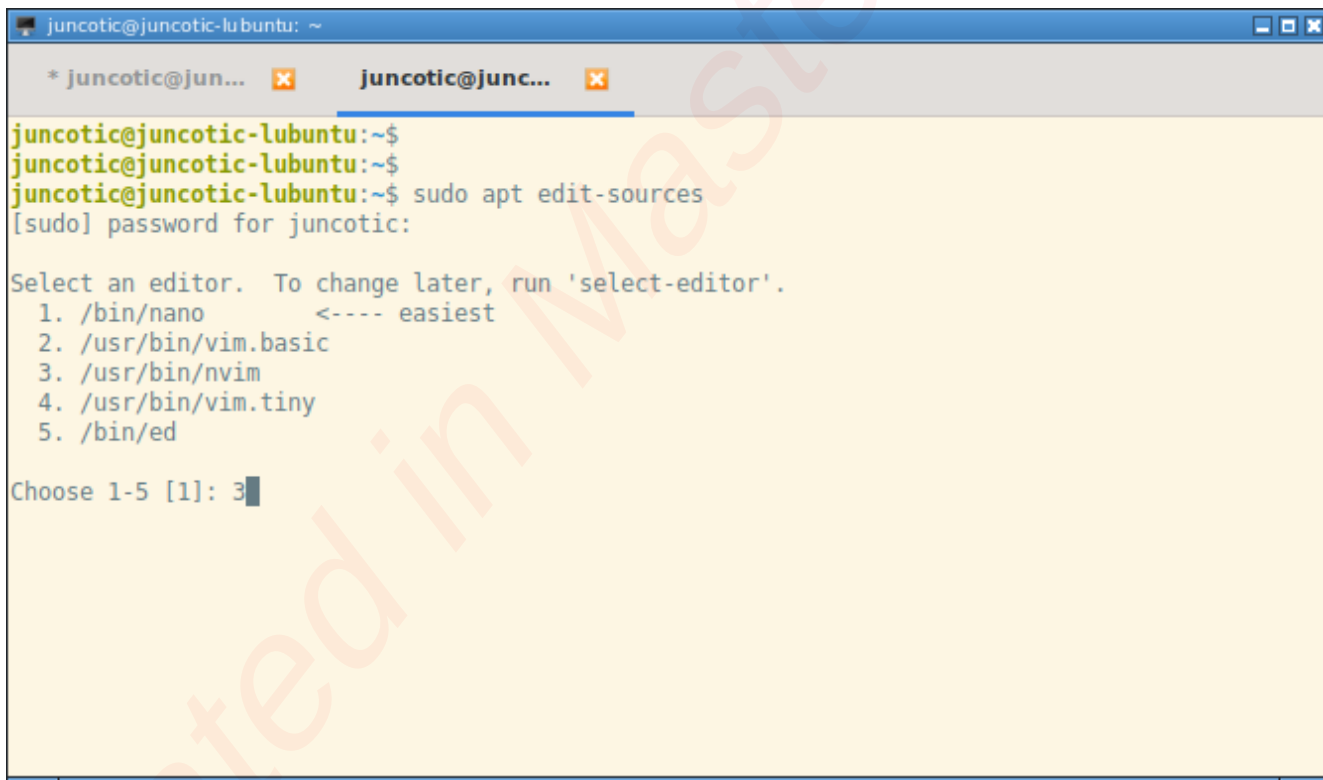
Nuevos comandos

La salida del comando `apt show` está ordenada alfabéticamente y no muestra información poco importante que sí agregaba `apt-cache show`. El viejo `apt-get dist-upgrade` es cambiado por el nuevo `apt full-upgrade`, un nombre más significativo para las tareas que realiza.

Por su parte, los comandos `apt list` y `apt edit-sources` son nuevos en la suite `apt`. `apt list` permite listar todos los paquetes del repositorio, y con modificadores como `--installed` o `--upgradable` podemos ver qué paquetes tenemos instalados, y cuáles son los candidatos para actualización. La lista de paquetes instalados anteriormente podía verse con `dpkg -l` (igual se mantiene este comando).

La opción `edit-sources` edita por defecto el archivo `/etc/apt/sources.list` con el editor predeterminado en línea de comandos, mientras que si tenemos listas de repositorios en archivo dentro de `/etc/apt/sources.list.d/` podemos pasar el nombre de la lista para poder editarla directamente de la misma forma.

En mi caso no tenía ningún editor predeterminado en terminal, y es por esto que el comando me da varias opciones:



```
juncotic@juncotic-lubuntu: ~  
* juncotic@junc... x juncotic@junc... x  
juncotic@juncotic-lubuntu:~$  
juncotic@juncotic-lubuntu:~$  
juncotic@juncotic-lubuntu:~$ sudo apt edit-sources  
[sudo] password for juncotic:  
  
Select an editor. To change later, run 'select-editor'.  
1. /bin/nano          <---- easiest  
2. /usr/bin/vim.basic  
3. /usr/bin/nvim  
4. /usr/bin/vim.tiny  
5. /bin/ed  
  
Choose 1-5 [1]: 3
```

Yo elegí `neovim` para editar, y automáticamente me abre el archivo `/etc/apt/sources.list` con este editor:

```
juncotic@juncotic-lubuntu: ~  
juncotic@junc... x juncotic@junc... x  
sources.list buffers  
7 deb http://archive.ubuntu.com/ubuntu/ disco main restricted  
8 # deb-src http://archive.ubuntu.com/ubuntu/ disco main restricted  
9  
10 ## Major bug fix updates produced after the final release of Lubuntu.  
11 ## Have you noticed a regression? Please report it!  
12 ## https://wiki.ubuntu.com/StableReleaseUpdates#Regressions  
13 deb http://archive.ubuntu.com/ubuntu/ disco-updates main restricted  
14 # deb-src http://archive.ubuntu.com/ubuntu/ disco-updates main restricted  
15  
16 ## Software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu team.  
17 ## Also, please note that software in Universe WILL NOT receive any review or  
18 ## updates from the Ubuntu security team directly. Updates in this repository  
19 ## are provided by volunteers, but most come from Debian.  
20 deb http://archive.ubuntu.com/ubuntu/ disco universe  
21 # deb-src http://archive.ubuntu.com/ubuntu/ disco universe  
22 deb http://archive.ubuntu.com/ubuntu/ disco-updates universe  
23 # deb-src http://archive.ubuntu.com/ubuntu/ disco-updates universe  
24  
25 ## Software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu team,  
NORMAL /etc/apt/sources.list debsources utf-8[unix] 12% 7/56 : 1
```

Y aptitude??

Aptitude agrega algunos otros comandos, y por sobre todo, la principal diferencia que tiene tanto con `apt-get` como con `apt` es su interfaz de `ncurses` al ejecutarlo sin argumentos:

aptitude

Yo en general suelo instalar por línea de comandos, por lo que esta interfaz no me resulta útil, así que a `aptitude` también lo he dejado de lado bastante, en favor de `apt`. No obstante, para quienes no se entiendan del todo con la terminal, `aptitude` con su interfaz ncurses es bastante intuitiva para listar, instalar, eliminar, etc, cualquier paquete del sistema... para los de la vieja escuela, esta interfaz es similar al viejo front-end `dselect` que corría sobre `apt-get`.

`Aptitude` sin embargo tiene algunas cosas interesantes, más allá de la interfaz ncurses. `Aptitude` tiene un sistema de resolución de dependencias basada en puntajes que le permite al usuario elegir entre una solución u otra, al estilo de «No quiero esa parte de la solución, pero sí quiero mantener esta otra parte en el siguiente intento de resolución». `Apt` busca la mejor forma de solucionar las dependencias en un solo intento, y en general eso me ha servido bien en mi trabajo.

Lista de comandos disponibles

Tanto `apt` como `apt-get` y `aptitude` disponen de una ayuda rápida con el modificador `--help` que nos da una lista de los comandos más utilizados y opciones que soporta cada uno:

apt-get

```
juncotic@juncotic-lubuntu: ~  
* juncotic@junc... x juncotic@junc... x  
juncotic@juncotic-lubuntu:~$  
juncotic@juncotic-lubuntu:~$  
juncotic@juncotic-lubuntu:~$ sudo apt-get --help  
apt 1.8.1 (amd64)  
Usage: apt-get [options] command  
       apt-get [options] install|remove pkg1 [pkg2 ...]  
       apt-get [options] source pkg1 [pkg2 ...]  
  
apt-get is a command line interface for retrieval of packages  
and information about them from authenticated sources and  
for installation, upgrade and removal of packages together  
with their dependencies.  
  
Most used commands:  
update - Retrieve new lists of packages  
upgrade - Perform an upgrade  
install - Install new packages (pkg is libc6 not libc6.deb)  
reinstall - Reinstall packages (pkg is libc6 not libc6.deb)  
remove - Remove packages  
purge - Remove packages and config files  
autoremove - Remove automatically all unused packages  
dist-upgrade - Distribution upgrade, see apt-get(8)  
dselect-upgrade - Follow dselect selections  
build-dep - Configure build-dependencies for source packages  
clean - Erase downloaded archive files  
autoclean - Erase old downloaded archive files  
check - Verify that there are no broken dependencies  
source - Download source archives  
download - Download the binary package into the current directory  
changelog - Download and display the changelog for the given package  
  
See apt-get(8) for more information about the available commands.  
Configuration options and syntax is detailed in apt.conf(5).  
Information about how to configure sources can be found in sources.list(5).  
Package and version choices can be expressed via apt_preferences(5).  
Security details are available in apt-secure(8).  
This APT has Super Cow Powers.  
juncotic@juncotic-lubuntu:~$
```

aptitude


```
juncotic@juncotic-lubuntu: ~  
juncotic@junc... x juncotic@junc... x  
juncotic@juncotic-lubuntu:~$  
juncotic@juncotic-lubuntu:~$  
juncotic@juncotic-lubuntu:~$  
juncotic@juncotic-lubuntu:~$ sudo aptitude --help  
aptitude 0.8.11  
Usage: aptitude [-S fname] [-u|-i]  
       aptitude [options] <action> ...  
  
Actions (if none is specified, aptitude will enter interactive mode):  
  
install      Install/upgrade packages.  
remove       Remove packages.  
purge        Remove packages and their configuration files.  
hold         Place packages on hold.  
unhold       Cancel a hold command for a package.  
markauto     Mark packages as having been automatically installed.  
unmarkauto   Mark packages as having been manually installed.  
forbid-version Forbid aptitude from upgrading to a specific package version.  
update       Download lists of new/upgradable packages.  
safe-upgrade Perform a safe upgrade.  
full-upgrade Perform an upgrade, possibly installing and removing packages.  
build-dep    Install the build-dependencies of packages.  
forget-new   Forget what packages are "new".  
search       Search for a package by name and/or expression.  
show         Display detailed info about a package.  
showsrc      Display detailed info about a source package (apt wrapper).  
versions     Displays the versions of specified packages.  
clean        Erase downloaded package files.  
autoclean    Erase old downloaded package files.  
changelog    View a package's changelog.  
download     Download the .deb file for a package (apt wrapper).  
source       Download source package (apt wrapper).  
reinstall    Reinstall a currently installed package.  
why          Explain why a particular package should be installed.  
why-not      Explain why a particular package cannot be installed.  
  
add-user-tag Add user tag to packages/patterns.  
remove-user-tag Remove user tag from packages/patterns.
```

apt

```
juncotic@juncotic-lubuntu: ~  
juncotic@junc... x juncotic@junc... x  
juncotic@juncotic-lubuntu:~$  
juncotic@juncotic-lubuntu:~$  
juncotic@juncotic-lubuntu:~$ sudo apt --help  
apt 1.8.1 (amd64)  
Usage: apt [options] command  
  
apt is a commandline package manager and provides commands for  
searching and managing as well as querying information about packages.  
It provides the same functionality as the specialized APT tools,  
like apt-get and apt-cache, but enables options more suitable for  
interactive use by default.  
  
Most used commands:  
list - list packages based on package names  
search - search in package descriptions  
show - show package details  
install - install packages  
reinstall - reinstall packages  
remove - remove packages  
autoremove - Remove automatically all unused packages  
update - update list of available packages  
upgrade - upgrade the system by installing/upgrading packages  
full-upgrade - upgrade the system by removing/installing/upgrading packages  
edit-sources - edit the source information file  
  
See apt(8) for more information about the available commands.  
Configuration options and syntax is detailed in apt.conf(5).  
Information about how to configure sources can be found in sources.list(5).  
Package and version choices can be expressed via apt_preferences(5).  
Security details are available in apt-secure(8).  
This APT has Super Cow Powers.  
juncotic@juncotic-lubuntu:~$
```

Como puede verse, `apt` es bastante más simple que los anteriores, y hay que destacar que `apt-get` es una de las utilidades que incluye `apt`, `apt-cache` también tiene su opción `--help` con sus comandos particulares.

Además, como puede verse, `aptitude` incluye varias opciones para marcado de paquetes, congelado de versiones y demás. Estas tareas no son tan comunes, y no se incluyen en el comando `apt`, pero sí pueden trabajarse con el comando `apt-mark` como una extensión:

```
juncotic@juncotic-lubuntu: ~  
juncotic@junc... x juncotic@junc... x  
juncotic@juncotic-lubuntu:~$  
juncotic@juncotic-lubuntu:~$ sudo apt-mark --help  
apt 1.8.1 (amd64)  
Usage: apt-mark [options] {auto|manual} pkg1 [pkg2 ...]  
  
apt-mark is a simple command line interface for marking packages  
as manually or automatically installed. It can also be used to  
manipulate the dpkg(1) selection states of packages, and to list  
all packages with or without a certain marking.  
  
Most used commands:  
auto - Mark the given packages as automatically installed  
manual - Mark the given packages as manually installed  
minimize-manual - Mark all dependencies of meta packages as automatically installed.  
hold - Mark a package as held back  
unhold - Unset a package set as held back  
showauto - Print the list of automatically installed packages  
showmanual - Print the list of manually installed packages  
showhold - Print the list of packages on hold  
  
See apt-mark(8) for more information about the available commands.  
Configuration options and syntax is detailed in apt.conf(5).  
Information about how to configure sources can be found in sources.list(5).  
Package and version choices can be expressed via apt_preferences(5).  
Security details are available in apt-secure(8).  
juncotic@juncotic-lubuntu:~$
```

Programando shell scripts con APT

En cuanto a la programación de scripts, acabo de leer el `man 8 apt` y me encuentro con una nota más que interesante para todos los que hacen la pregunta ¿sigo con `apt-get`? ¿migro a `apt`?:

```
juncotic@juncotic-lubuntu: ~
* juncotic@junc... x juncotic@junc... x

package names as well as options to list installed (--installed), upgradeable
(--upgradeable) or all available (--all-versions) versions.

edit-sources (work-in-progress)
edit-sources lets you edit your sources.list(5) files in your preferred
texteditor while also providing basic sanity checks.

SCRIPT USAGE AND DIFFERENCES FROM OTHER APT TOOLS
The apt(8) commandline is designed as an end-user tool and it may change behavior
between versions. While it tries not to break backward compatibility this is not
guaranteed either if a change seems beneficial for interactive use.

All features of apt(8) are available in dedicated APT tools like apt-get(8) and apt-
cache(8) as well. apt(8) just changes the default value of some options (see
apt.conf(5) and specifically the Binary scope). So you should prefer using these
commands (potentially with some additional options enabled) in your scripts as they
keep backward compatibility as much as possible.

SEE ALSO
apt-get(8), apt-cache(8), sources.list(5), apt.conf(5), apt-config(8), The APT User's
guide in /usr/share/doc/apt-doc/, apt_preferences(5), the APT Howto.

DIAGNOSTICS
apt returns zero on normal operation, decimal 100 on error.

Manual page apt(8) line 87/123 95% (press h for help or q to quit)
```

man apt

`apt` es una interfaz de línea de comandos diseñada como una herramienta de **usuario final** y puede tener diferencias y cambios entre las versiones, principalmente en los valores por defecto para algunas opciones. Esto dificulta la **compatibilidad hacia atrás**. Como todas las características de `apt` se encuentran en `apt-get` y `apt-cache`, y éstos comandos sí se mantienen más estables en su comportamiento, la recomendación es seguir usando `apt-get` para programación de scripts.

Conclusiones

He tratado de montar una fuente de referencias para entender *grosso modo* las diferencias entre el viejo `apt-get` / `apt-cache`, el nuevo `apt`, y el paquete alternativo `aptitude`.

`apt-get` y `apt-cache` poseen una gran cantidad de funcionalidades, un exceso de lo que un usuario promedio puede utilizar. Además, como se mencionó, muchas funcionalidades están dispersas entre `apt-get` y `apt-cache`. `apt` viene a solucionar este problema, y provee las opciones necesarias para gestionar los paquetes y actualizaciones en el sistema DEB.

`apt-get` no es un paquete desactualizado/*deprecated*, simplemente es un juego de herramientas de más **bajo nivel** que sigue y seguirá estando disponible. `apt` es una abstracción de algunas de estas herramientas, con un CLI más intuitivo y fácil de utilizar.

Personalmente suelo utilizar `apt` para uso diario en la gestión de paquetes en distros DEB, y `apt-get` para algunos usos en shell scripts. `Aptitude` por su parte no suelo utilizarlo últimamente, en su lugar uso `apt`.

Instalación de aplicaciones

Estándar de jerarquía del sistema de archivos

Antes de instalar software no administrado por nuestra distribución conviene que tengamos una idea de como se organiza el sistema de archivos en Linux para que entendamos dónde se instala el software. Linux sigue una norma estándar llamada [Estándar de jerarquía del sistema de archivos](#) que define los directorios del sistema y la localización de los distintos tipos de archivos.

En Unix todos los archivos y directorios aparecen bajo el directorio raíz (/). Es habitual que dentro del directorio raíz existen tres jerarquías en las que se distribuyen los archivos y directorios de los programas:

- **/, jerarquía primaria.** De ella cuelgan el resto de jerarquías. Los archivos incluidos directamente en esta jerarquía son sólo los esenciales para el sistema, como por ejemplo los comandos: `cp`, `ls` o `mkdir`.
- **/usr/, jerarquía secundaria.** Contiene la mayoría de aplicaciones del sistema. En las distribuciones Linux es la jerarquía en las que los programas de uso común son instalados, como por ejemplo el LibreOffice o el entorno de usuario Gnome.
- **/usr/local/, jerarquía terciaria.** Contiene la mayoría de las aplicaciones que instalamos sin la mediación de la distribución.

Dentro de cada una de las jerarquías hay varios directorios en los que se distribuyen los archivos de las aplicaciones dependiendo del tipo de archivo. Por ejemplo, los ejecutables se encuentran en los directorios *bin* y las librerías en *lib*. En mi ordenador el ejecutable `cp`, que es esencial para el sistema se encuentra en `/bin/cp`, el editor de textos `gedit` que ha sido instalado por la distribución y no es esencial para el sistema se encuentra en `/usr/bin/gedit` y el alineador de secuencias `bwa` que yo he instalado manualmente sin utilizar el gestor de paquetes se encuentra en `/usr/local/bin/bwa`.

A diferencia de otros sistemas operativos en los sistemas Linux las aplicaciones normalmente no están contenidas en un sólo directorio. Los ejecutables están en *bin*, las librerías de los que dependen en *lib*, etc. Cuando se instalan programas estáticos, que incluyen todas sus librerías, suelen instalarse en `/opt/`. `opt/` sigue un modelo similar al *Archivos de programa* de Windows. No es muy común que los programas se instalen de esta forma en *Linux*, pero algunos programas como el *IDE java eclipse* sí suelen instalarse en `/opt`

Instalando programas

README e INSTALL

Cuando queramos instalar aplicaciones que no esten en repositorios o que la aplicacion que queremos está pero necesitamos una version más nueva, tendremos que ir programa por programa instalandolo. No existe una sola forma de instalarlos, pero si que todos los programas suelen tener algun tipo de documento en el que se explica como instalar el programa. Normalmente este documento suele llamarse README o INSTALL, aunque no hay ninguna regla que diga que esos son los nombres, por lo que lo primero que tenemos que hacer cuando nos descargemos la aplicacion es buscar este archivo o alguno en el que intuyamos en el que pueda estar la informacion acerca de como instalarlo.

Instalando programas compilados

Una vez hemos visto dónde debemos instalar los programas que administremos sin la ayuda de la distribución (en `/usr/local`) vamos a ver como instalaríamos el mapeador de secuencias cortas [bowtie](#). En su página de [descargas](#) encontramos los siguientes ficheros:

- bowtie-1.1.2-src.zip
- bowtie-1.1.2-macos-x86_64.zip
- bowtie-1.1.2-linux-x86_64.zip
- bowtie-1.1.2-mingw-x86_64.zip

Los ficheros de interés en un sistema Linux son los marcados como linux-x86_64 y src. El primeros incluyen el programa compilado y el segundo el código fuente listo para compilar. Normalmente si se nos ofrece el programa precompilado para nuestra arquitectura podemos simplemente copiar el programa a `/usr/local`. En este caso el programa lo han compilado para la arquitecturas x86_64. x86_64 se refiere a los microprocesadores *Intel* y *AMD* de 64 bits.

Esta arquitectura no tiene que coincidir con nuestro procesador sino con la versión de la distribución que tenemos instalada. Por ejemplo, Ubuntu y Debian tienen versiones para 32 y 64 bits, pero en un ordenador con un microprocesador de 64 bits podemos optar por instalar la versión de la distribución de 32 bits. La arquitectura del programa compilado debe coincidir con la de la distribución que tengamos instalada, no con la de nuestro microprocesador. Para poder saber la arquitectura de nuestro sistema operativo, podemos usar el comando `uname` :

```
~$ uname -a
Linux txindoki 4.6.0-1-amd64 #1 SMP Debian 4.6.4-1 (2016-07-18) x86_64 GNU/Linux
```

Supongamos que queremos instalar la versión de 64 bits del programa precompilado. Una vez que hemos descargado el fichero zip (que en muchas ocasiones será en realidad un tar.gz), lo descomprimos. En el directorio descomprimido encontramos los ejecutables, en este caso el *bowtie* y otros archivos adicionales, en este caso *scripts* en el directorio *scripts* y ficheros de ejemplos en los directorios *indexes*, *genomes* y *reads*. Es posible que entre todos estos ficheros haya un *README* o un *INSTALL* si es así debemos leerlos antes de continuar.

La forma más sencilla de instalar el programa es mover el directorio completo a `/usr/local/bowtie/`. Una vez hecho deberemos añadir los directorios en los que hay ejecutables a nuestro `$PATH`. Sino lo hacemos no podremos ejecutar el programa a no ser que incluyamos la ruta completa al ejecutables `/usr/local/bowtie/bowtie`.

Para incluir el directorio con los ejecutables en la variable *PATH* debemos modificar la variable con la siguiente orden::


```
~$ export PATH=$PATH:/usr/local/bowtie/:/usr/local/bowtie/scripts/
```

Una vez ejecutada esta orden ya podríamos ejecutar el bowtie en la terminal. El problema de este método es que cada vez que nos salimos de la sesión del terminal esta modificación del \$PATH se pierde. Para que el \$PATH sea el que deseamos siempre que entremos al sistema el comando anterior debe ser ejecutado, esto podemos conseguirlo incluyéndolo en el fichero `.bashrc` situado en nuestra \$HOME. `.bashrc` se ejecuta automáticamente cada vez que entramos en un *shell* y se utiliza para adaptar las variables de entorno, como por ejemplo \$PATH, a nuestras necesidades.

Ejercicios: 1. Instala la ultima version de ncbi-blast+ 2. Instala la ultima version de [tophat](#)

Instalando un programa desde el código fuente

Cuando no tenemos la posibilidad de instalarnos el programa ni desde la distribución ni usando binarios compilados, podemos optar por instalar un programa partiendo desde el código fuente, para ello primero tenemos que compilarlo. Compilar significa traducir el código en texto escrito por humanos en un código que el ordenador entienda; código binario. Este suele ser un caso bastante habitual en las aplicaciones bioinformáticas, para los casos en los que los desarrolladores no han creado binarios compilados para nuestra arquitectura.

Normalmente los ficheros con el código fuente con ficheros comprimidos *tar.gz*, pero en el ejemplo que nos ocupa es un fichero comprimido *zip*. Una vez descomprimido normalmente encontraremos un fichero *INSTALL* o *README* si es así debemos leerlos antes de continuar.

El caso más habitual es que entre los archivos descomprimidos haya un *shell script* ejecutable llamado *configure*. Si es así el procedimiento a seguir suele ser ejecutar la secuencia de comandos `./configure`, `make`, `make install`. *configure* verificará que nuestro sistema dispone de todas las librerías y utilidades necesarias para instalar el programa y creará una serie de ficheros *Makefile*. Una vez terminado el *configure* y creados los *Makefile* satisfactoriamente podemos ejecutar el comando `make` y el programa se compilará para nuestra arquitectura. El *configure* también suele ser el encargado de determinar en qué jerarquía del sistema de archivos se instalará el programa. Lo habitual es que estos programas compilados manualmente se instalen en la tercera jerarquía, `/usr/local/`.

Para poder compilar los programas necesitamos tener instaladas una serie de aplicaciones. En Ubuntu por defecto estas aplicaciones no se instalan en el sistema. Para instalarlas debemos instalar el paquete `build-essential`:

```
~$ sudo apt-get install build-essential
```

Este comando instalará, entre otras cosas el compilador *gcc* y los *headers* de las librerías básicas del sistema. Los *headers* se encuentran en los paquetes *dev* (*devel* en *RedHat* y derivados) y son necesarios para poder compilar programas que requieran las librerías. Para compilar un programa que utilice una librería no es suficiente con tener instalada la librería, necesitamos instalar el paquete *dev* con los *headers*. Si el *configure* se queja de que alguna librería no está instalada lo más habitual es que no tengamos el correspondiente paquete *dev* instalado.

Una vez tenemos todas las librerías requeridas instaladas el compilador podrá compilar el programa cuando ejecutemos el `make`. Este proceso debe terminar sin errores, si algo falla debemos leer la salida del `make` y buscar el fallo. Normalmente el fallo suele deberse a la falta de alguna librería. Si el fallo es más complicado lo más recomendable es buscar primero en *google* y si no encontramos la solución preguntar a los desarrolladores del software por el problema.

Una vez compilado sólo queda ejecutar *make install* para que el software se instale. Como lo habitual es que el programa esté destinado a ser instalado en */usr/local/* para poder ejecutar el *make install* deberemos tener privilegios de administrador.

```
~$ sudo make install
```

Con esto el programa debe estar listo para ser ejecutado puesto que el binario ejecutable debe haber sido copiado por el comando *make install* a algún directorio del *\$PATH*, normalmente a */usr/local/bin*.

Ejercicios 1. Instala [samtools](#) 2. Instala [bwa](#) 3. Instala [freebayes](#) 4. Instala [RAxML](#) 5. Instala la última versión de [PyMOL](#)

Instalando programas Java.

Java es un lenguaje de programación de propósito general. Es un lenguaje muy usado en programas de bioinformática porque es un lenguaje multiplataforma. Escribes una vez el código y lo puedes usar en los distintos sistemas operativos.

Cual es el truco? Tienes que tener instalado en el sistema el intérprete de java. Este intérprete será especial para cada sistema operativo, pero podrá ejecutar cualquier código java. Existen diferentes intérpretes de java, algunos son software libre, y suelen estar en la distribución y luego tenemos el java de Oracle.

Para poder ejecutar un programa de java, necesitaremos un *Java runtime environment (JRE)*. Podemos instalar varios:

Java libres

Una forma de tener java en nuestro sistema es mediante las versiones libres que nos ofrecen las distribuciones. Hay muchas variantes y distintas versiones para cada variante, por lo que si no sabemos que versión instalar lo mejor es instalar la que ellos ponen por defecto.

```
~$ java
El programa «java» puede encontrarse en los siguientes paquetes:
* default-jre
* gcj-5-jre-headless
* openjdk-8-jre-headless
* gcj-4.8-jre-headless
* gcj-4.9-jre-headless
* openjdk-9-jre-headless
Intente: sudo apt install <paquete seleccionado>

~$ sudo apt install default-jre
```

Java de oracle

Oracle es la empresa dueña de java. Ellos son los que lo desarrollan y por lo tanto su intérprete de java suele ser el de referencia. No es software libre y por lo tanto las distribuciones no lo pueden distribuir, pero existen formas sencillas de instalarlos. En ubuntu tenemos un PPA en el que podemos instalar un instalador que nos instalará el java de oracle. Para instalarlo:

```
~$ sudo add-apt-repository ppa:webupd8team/java
~$ sudo apt-get update
~$ sudo apt-get install oracle-java8-installer
```

Como siempre lo primero que tenemos que hacer es leernos la documentacion donde nos diga como ejecutarlo. Lo más usual es que el programa se distribuya como un fichero .jar. P.ej trimmomatic:

```
~$ java -jar Trimmomatic.jar
```

Otros programas en cambio se distribuyen con un ejecutable que se puede directamente ejecutar: p.ej: Fastqc:

```
~$ fastqc
```

Ejercicios: 1. Instala [trimmomatic](#)

1. Instala [VarScan](#)
2. Instala [FastQC](#), ¿Cómo es el ejecutable para lanzar el fastq? ¿Podrías modificarlo?

Python

Python es otro lenguaje de programación de propósito general. Es un lenguaje interpretado, por lo que no hay que traducir el código fuente en código binario. De eso ya se encargará el intérprete de python cada vez que ejecutamos algo.

Normalmente las distribuciones de linux suelen instalar python por defecto. Ahora mismo hay dos versiones de python conviviendo juntas; python2 y python3. Se pueden instalar las dos simultáneamente y tendremos que usar la versión que nuestro programa/librería necesite. Cuando instalamos una aplicación en python solo lo instalaremos para la versión que estemos usando en ese momento. Las librerías son únicas para esa versión. Para saber cuál de las 2 versiones estamos usando podemos usar el comando `which`

Tenemos varios métodos para instalar aplicaciones de python. 1. Instalarlos desde los repositorios.

1. Python tiene un repositorio de software llamado [pypi](#) donde muchos desarrolladores “cuelgan” sus programas. Para poder instalar estos programas necesitamos instalar un programa que se llama `pip`. Con pip podremos administrar los paquetes del repositorio pypi. `~$ sudo apt-get install python-pip ... ~$ pip install nombre_programa`
2. Utilizando el script instalador que suelen tener los programas. Una vez descomprimido, entrados en el directorio y ejecutamos la siguiente orden: `~$ tar zxvf programa_python.tar.gz ... ~$ cd programa_python ~$ python setup.py install`
3. Si el programa es un solo ejecutable, lo podemos instalar como cualquier otro ejecutable.

Virtualenv

Python nos permite, en aquellos entornos en los que no tenemos permisos para instalar fuera de nuestra HOME, crear un entorno virtual en el directorio que le digamos. Así, podremos instalar las aplicaciones que necesitamos sin hacer cambios en el sistema. Con el comando `virtualenv` crearemos un entorno virtual en python2. Y con el comando `pyenv` crearemos un entorno virtual en python3:

```
user@virtual:~$ virtualenv pyenv2
Running virtualenv with interpreter /usr/bin/python2
New python executable in /home/user/pyenv2/bin/python2
Also creating executable in /home/user/pyenv2/bin/python
Installing setuptools, pkg_resources, pip, wheel...done.
```

Una vez creados el entorno virtual, vemos que tenemos 1 nuevo directorio: pyenv2. Para usar el entorno virtual tendremos que activarlo:

```
user@virtual:~$ source pyenv2/bin/activate
user@virtual:~$ source pyenv2/bin/activate
(pyenv2) user@virtual:~$ which python
/home/user/pyenv2/bin/python
```

Ejercicios: 1. Instala [Biopython](#) en el sistema

1. Instala [pandas](#) en el sistema
2. Instala [ngs_crums](#) en el sistema.
3. Crea un entorno virtual 3en python2 y repite los ejercicios 1 y 3 pero instaladolo en el entorno virtual
4. Instala la [ultima version de python disponible](#)

R

R es un entorno y lenguaje de programación con un enfoque al análisis estadístico. Para poder ejecutar cualquier librería o programa en R lo primero que tenemos que instalar es el intérprete de R. La mejor forma es hacerlo desde un paquete de la distribución:

```
~$ sudo apt-get install r-base
```

R utiliza un repositorio llamado cran para distribuir los paquetes. Estos paquetes se pueden instalar desde la propia *shell* de R utilizando el comando `install.packages(nombre_paquete)`:

```
~$ R

>install.packages('ggplot2')
```

En R versiones concretas de librerías dependen de versiones concretas del intérprete de R, por lo que si necesitamos una versión concreta de una librería de R, tendremos que instalar la versión de R de la que dependa. En el caso que necesitemos dos versiones de R diferentes, lo tendremos que instalar a mano, ya que las distribuciones no suelen permitir la instalación de diferentes versiones simultáneamente.

Ejercicios: 1. Instalar la librería XML de R

1. Instala [\[bioconductor\]](#)
2. Instalar [\[la última versión de R\]](#) manteniendo la que hemos instalado con la distribución.

Programas que se pueden instalar

- C programs
 - bwa
 - samtools
 - bedtools
 - freebayes
 - Blast
 - bowtie y bowtie2
 - valvet
 - trinity
 - repeatmasker -> estos son chungos pero puede servir como ejercicio final
 - repeatexplorer -> estos son chungos pero puede servir como ejercicio final
 - phylip
 - RAxML -> compilarlo parece divertido . Hay que saber el procesador que tienes.
 - phym1
- java:
 - fastqc
 - picard-tools
 - trimmomatic
 - igv
- python
 - Biopython
 - matplotlib
- repeatmasker
- emboos
- staden
- hmmer
- clustalw

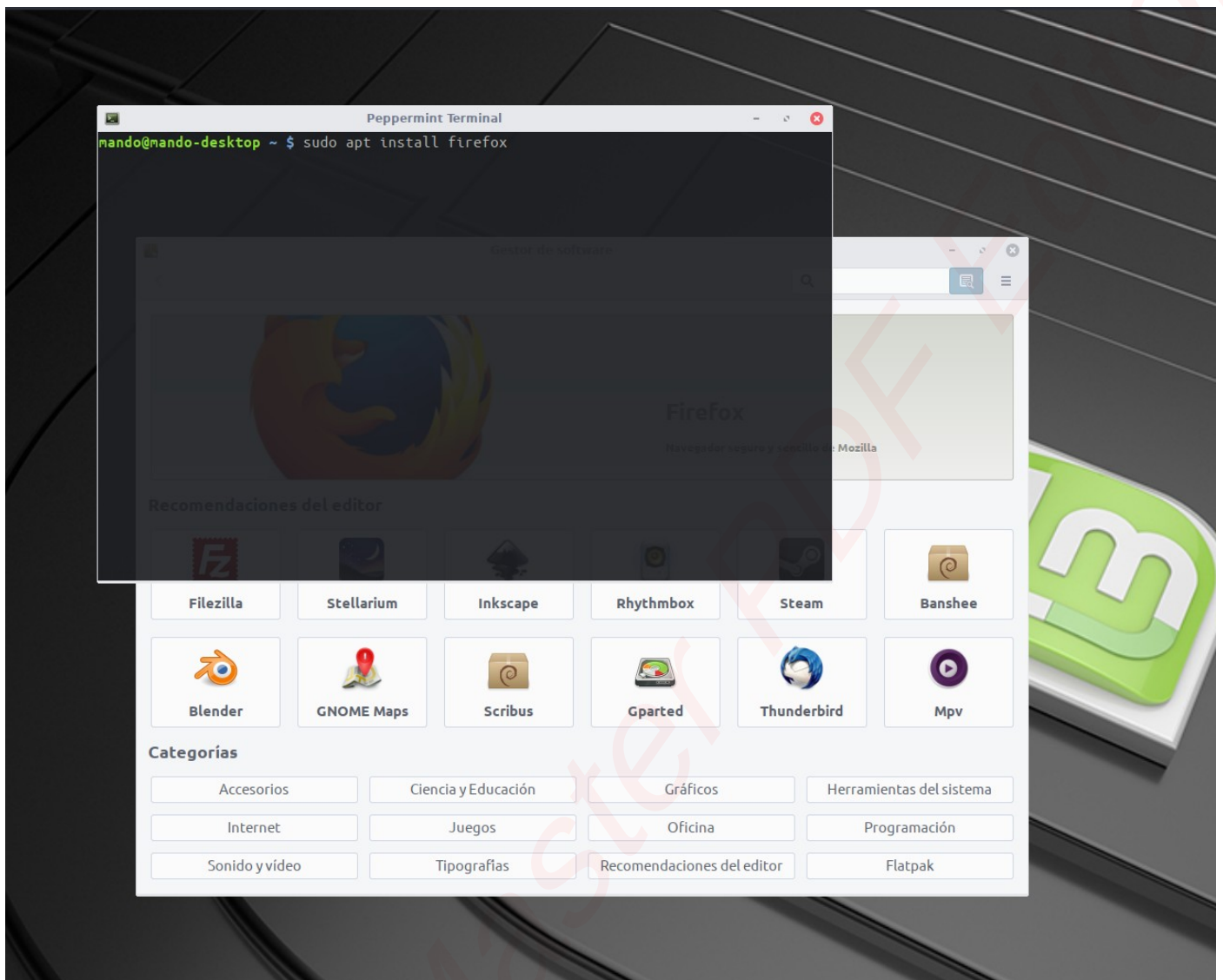
Instalar software en Linux usando la Terminal y otros métodos

Instalar software en Linux usando la Terminal es más sencillo de lo que te imaginas. Sin embargo, los comandos utilizados pueden variar un poco dependiendo del Gestor de Paquetes en tu sistema operativo.

En cualquier caso, a continuación te mostraremos las diferentes formas de instalar software en Linux, usando la Terminal y otras herramientas. Antes de eso, hablemos primero de los gestores de paquetes.

¿Qué es un Sistema de Gestión de Paquetes?

Básicamente es un conjunto de herramientas y formatos de archivos, que se utiliza para instalar, actualizar y desinstalar software en Linux. Actualmente los dos sistemas de gestión de paquetes más comunes son Red Hat y Debian.



En cuanto a Fedora, Mandriva, CentOS y otras distribuciones, utilizan el sistema rpm de Red Hat, (archivos .rpm). Por su parte, [Ubuntu](#), Linux Mint, Peppermint, entre otros, utilizan el sistema dpkg de Debian, (archivos .deb).

Sin embargo, la principal diferencia entre estos dos gestores de paquetes es la forma en que se instala y se mantiene el software.

Instalar software en Linux Mint, Ubuntu, Peppermint y otros (Debian)

Realmente hay dos formas de instalar programas en Linux desde la Terminal usando Debian. Es decir, puedes usar el programa apt para instalar el software desde un repositorio.

También tienes la opción de utilizar el programa **dpkg** para instalar aplicaciones desde archivos **.deb**. Es decir, puedes usar cualquiera de las dos según lo requieras.

Instalar un programa en Linux usando apt:

```
sudo apt install nombre_del_software
```

Desinstalar un programa en Linux:

```
sudo apt remove nombre_del_software
```

Y si solo quieres actualizar el software instalado en Linux, primero tendrás que actualizar el repositorio de aplicaciones. Para ello se utiliza el siguiente comando:

```
sudo apt update
```

Después que se actualice el repositorio, puedes actualizar cualquier programa que lo requiera con este comando:

```
sudo apt upgrade
```

Si únicamente deseas actualizar una aplicación, entonces utilizas este comando:

```
sudo apt update nombre_del_software
```

También puedes instalar un programa en Linux que hayas descargado en formato **.deb**. Para ello solo utiliza:

```
sudo dpkg -i nombre_del_software.deb
```

Instalar programas en Linux usando Red Hat

Hay dos comandos que se pueden utilizar en Red Hat para instalar software en Linux desde la Terminal: yum y dnf. Lo puedes hacer así:

```
sudo yum install nombre_del_software
```

```
sudo dnf install nombre_del_software
```

Además, el software en formato .rpm también se puede instalar usando el comando rpm:

```
sudo rpm -i nombre_de_software
```

Si deseas eliminar programas no deseados o que no uses:

```
sudo yum remove nombre_del_software
```

```
sudo dnf remove nombre_del_software
```

Y para actualizar los programas de Linux ya instalados se utilizan los siguientes comandos:

```
yum update
```

```
sudo dnf upgrade --refresh
```

¿Hay otra forma de instalar software en Linux sin usar la Terminal?

Ciertamente la hay, y de hecho es el método para instalar y eliminar programas en Linux preferido por muchos. Principalmente aquellos usuarios que recién comienzan su experiencia en este sistema operativo.

El **Gestor de Software** o Administrador de Software, es la aplicación desarrollada internamente en Linux Mint. Evidentemente su objetivo es ofrecer un entorno simple e intuitivo para gestionar todos los paquetes de software disponibles.

La ventaja principal del Gestor de Software es su **comodidad y facilidad de uso**. Es decir, aquí encontrarás todos los programas para Linux, desglosados en categorías, de acuerdo a su uso previsto.

No solo eso, también puedes acceder a descripciones, reseñas y calificaciones para cada uno de los programas. Además, usualmente hay una sección donde aparecen las aplicaciones más destacadas, no solo las recomendaciones del editor.

Por lo tanto, si quieres instalar aplicaciones en Linux desde el Gestor de Software, solo debes hacer lo siguiente:

- Lo primero es navegar hasta el programa que deseas instalar.
- Puedes encontrar la aplicación accediendo a la categoría correspondiente, o también puedes buscar la función de búsqueda.
- Cuando encuentres la aplicación, simplemente haz clic en el botón "Instalar".
- Después el Gestor de Software se encargará de todo lo demás.

Después, solo debes esperar unos pocos segundos y podrás acceder al programa desde el menú de Linux. Y si lo que deseas es desinstalar una aplicación, el procedimiento es el mismo. Sin embargo, esta vez se mostrará el botón "Eliminar", ya que se trata de un programa ya instalado.

Gestor de Paquetes Synaptic

Esta es **otra forma de instalar software en Linux**, aunque no es del todo recomendable para usuarios principiantes. Además, la herramienta permite instalar, actualizar o desinstalar paquetes de software en el sistema.

Por lo general lo encuentras en la sección Sistema, del Menú de Linux. Y ya sea que desees instalar, actualizar o eliminar un programa, tendrás que marcar los paquetes indicados. Después de esto solo tienes que hacer clic en el botón "Aplicar".

Instalador de paquetes GDebi

Finalmente también es posible instalar software en Linux a través de la herramienta **GDebi**. Es muy fácil de usar, aunque solo funciona con los paquetes de software en formato .deb. Además y a diferencia de las otras opciones, GDebi no requiere de repositorios, ni tampoco de una [conexión a internet](#).

En otras palabras, la instalación del software se realiza de forma local. En consecuencia solo debes hacer doble clic sobre el paquete de software descargado. Después de esto se te mostrará una pequeña ventana con la descripción del programa y otros detalles.

Para instalar el programa simplemente debes hacer clic en "Instalar paquete".

equivalencias apt-get, pacman y zypper (Debian, Arch, OpenSuse)

Todo linuxero que se precie instala con frecuencia paquetes en modo consola/terminal, para eso existen los potentes y rápidos programas de gestión de paquetes en modo texto, tanto en Debian (apt-get), Arch (pacman) u OpenSuse (zypper).

Veamos sus equivalencias principales:

APT-GET – DEBIAN / UBUNTU y derivadas:

`sudo apt-get update` (actualizar la base de datos de los repositorios)

`sudo apt-get upgrade` (actualizar el sistema)

`sudo apt-get install paquete` (instala el paquete)

`sudo apt-get remove paquete` (desinstala el paquete)

`sudo apt-cache search paquete` (buscar un paquete)

sudo apt-cache show paquete (muestra información del paquete)



PACMAN – ARCH / MANJARO y derivadas:

sudo pacman -Syu (actualizar la base de datos de los repositorios y actualizar el sistema)

sudo pacman -S paquete (instala el paquete)

sudo pacman -R paquete (desinstala el paquete)

sudo pacman -Rs paquete (desinstala el paquete y sus dependencias no útiles para el sistema)

sudo pacman -Ss paquete (busca un paquete específico)

sudo pacman -Sw paquete (descarga el paquete pero no lo instala)

sudo pacman -U /ruta/paquete.pkg.tar.gz (instala un paquete desde una carpeta local)

sudo pacman -Q (muestra la lista de todos los paquetes instalados en el sistema)

sudo pacman -Scc (borra todos los paquetes guardados en la cache de pacman en: /var/cache/pacman/pkg)



ZYPPER – OPENSUSE y derivadas:

sudo zypper update (actualizar la base de datos de los repositorios y actualizar el sistema)

sudo zypper in paquete (instala el paquete)

sudo zypper rm paquete (desinstala el paquete)



para más información `man apt-get` , `man pacman` , `man zypper`

comandos emerge de Portage en Gentoo, explicados

Portage es otro de los grandes gestores de paquetes de por ejemplo Gentoo. Presenta similitudes con los Ports de BSD y es compatible con POSIX y el entorno python. Además también lo emplea FreeBSD. Para instalar un paquete con él:

emerge es el comando para controlar **Portage**, el ya legendario sistema de administración de paquetes de **Gentoo**. Por eso en Gentoo hablamos de “**emerge**” un paquete de Portage, que significa: descargar sus fuentes, compilarlo e instalarlo en el sistema.

He aquí los principales comandos de emerge seguidos de una breve explicación. El directorio principal de configuración de Portage está en: `/etc/portage/` donde están sus archivos de configuración:

make.conf (archivo de configuración principal de Portage)

package.use (donde se pueden definir las “USE flags” para paquetes individuales)

package.mask (donde se pueden “enmascarar” paquetes para que no se instalen o actualicen)

package.unmask (donde se pueden desenmascarar paquetes enmascarados en Portage para permitir que se instalen)

package.accept keywords (donde autorizar paquetes inestables)

sincronizar Portage:

`sudo emerge --sync` actualiza el árbol de Portage que está en: `/usr/portage/`

`sudo emerge-webrsync` actualiza el árbol de Portage desde la última instantánea de la web de Gentoo

buscar en Portage:

`emerge -s paquete` busca el paquete `--search`
`emerge -S palabra` busca también en las descripciones `--searchdesc`

instalar paquetes:

`emerge -p paquete` muestra las dependencias del paquete sin instalarlo `--pretend`
`sudo emerge -a paquete` instala el paquete, (-a pide confirmación antes de hacerlo `--ask`)
`sudo emerge -f paquete` descarga el paquete de fuentes pero no lo instala `--fetchonly`

Portage guarda las fuentes en: `/usr/portage/distfiles/`

desinstalar paquetes:

`sudo emerge -Ca paquete` desinstala el paquete y sus dependencias (Portage no mira si las dependencias las necesita otro paquete, tampoco desinstala los archivos de configuración (`--unmerge --ask`))

actualización básica del sistema:

`sudo emerge -ua world` actualiza el sistema (no necesariamente las dependencias (`--update --ask @world`))
`sudo emerge -uaD world` actualiza el sistema incluidas todas las dependencias (`--update --ask --deep @world`)

actualización avanzada del sistema:

`sudo emerge -uaD --with-bdeps=y world` actualiza el sistema incluidas todas las dependencias + ("build dependencies") (`--update --ask --deep --with-bdeps=y @world`)

`sudo emerge -uaDN --with-bdeps=y world` todo lo anterior + revisa por si hay cambios USE.

(`--update --ask --deep --newuse --with-bdeps=y @world`)

desinstalar dependencias huérfanas en 3 pasos:

`sudo emerge -uaDN world` (`--update --ask --deep --newuse @world`)
`sudo emerge --depclean`
`sudo revdep-rebuild`



Cómo instalar tarballs:

Los paquetes que se instalan directamente desde la fuente se empaquetan con la primitiva, pero aun útil y eficiente, herramienta Tar (de ahí el nombre **tarball**) y luego se comprimen empleando algún tipo de formato comprimido.

Algunos paquetes de este tipo vienen con ficheros en su interior tipo .jar, .bin, .rpm,..., en ese caso tan solo hay que desempaquetar y emplear el procedimiento correcto para el binario que alberga. Pero por lo general es **código fuente** que hay que compilar e instalar.

Veamos como. Lo primero, cuando trabajamos **desde la consola**, es situarnos en el directorio donde se encuentra el paquete con el que queremos trabajar. Para ello utilizamos la herramienta "cd". Por ejemplo, si te has descargado un paquete y lo tienes en la carpeta Descargas, teclea en el terminal:

```
1cd Descargas
```

Y el **prompt** cambiará con esa ruta para indicarte que estás dentro de este directorio del sistema. También debes recordar que necesitas privilegios para ejecutar ciertas acciones como `./configure`, `make`, o `make install...` que veremos a continuación.

Instalar tar.gz o tgz:

Estos tipos de tarball es muy empleado en **Slackware y derivados**, aunque se ha extendido para empaquetar código para el resto de distribuciones. Instalar tar.gz es así (recuerda ejecutar `./configure`, `make` y `make install` con privilegios, ya sabes, como root o anteponiendo `sudo` al comando...):

```
1 cd directorio_donde_se_encuentra_el_tarball
2 tar -zxvf nombre_paquete.tar.gz (o nombre_paquete.tgz, en caso de ser
3 un .tgz)
4 cd nombre_paquete_desempaquetado
5 ./configure
6 make
7 make install
```

Si esto no funcionara para instalar tar.gz, puedes acceder al directorio desempaquetado para comprobar si existe algún fichero de texto con las instrucciones para instalarlo. A veces, cuando no siguen este procedimiento estándar, los desarrolladores incluyen este tipo de ficheros para explicarte las particularidades, dependencias, etc.

Tar.bz2 o .tbz2:

Se trata de un paquete muy empleado **en BSD** y que también se ha extendido a Linux y otros *nix. Es un empaquetado con tar y comprimido utilizando BSD Zip 2. El procedimiento para instalar este tipo de programas es:

```
1 cd directorio_donde_se_encuentra_el_paquete
2 tar -jxvf nombre_paquete.tar.bz2 (o nombre_paquete.tbz2, e incluso
3 nombre_paquete.tbz)
4 cd nombre_directorio_desempaquetdo
5 ./configure
6 make
```

make install

Así debería bastar para instalar programas en Linux. Asegúrate que usas **privilegios** para los últimos comandos.

Otros Tape Archive:

En ocasiones se emplea un tape archive o **fichero tar sin**

compresión. Este tipo de paquetes mantiene la información necesaria para restaurar totalmente los ficheros que contiene y para desempaquetarlo tan solo hay que hacer esto:

```
1tar xvf nombre_paquete.tar
```

Luego busca un fichero con nombre **README.txt** (o similar) dentro del directorio desempaquetado y busca las instrucciones de instalación. Normalmente se trata de hacer un procedimiento similar a los anteriores...

Tar.xz o .xz o .txz:

Ultimamente estoy viendo más de este tipo. Para operar con este tipo de paquetes hay que tener la herramienta **xz-utils** instalada. Para desempaquetarlos e instalarlos se emplea:

```
1tar Jxvf nombre_paquete.tar.xz
```

o

```
1Xz -d nombre_paquete.tar.xz
```

```
2Tar -xf nombre_paquete.tar
```

o

```
1Unxz nombre_paquete.xz
```

Y una vez descomprimido se busca un fichero **README.txt** o **INSTALL.txt** para ver los detalles de la instalación, que por lo general es la típica ./configure, make y make install. Aunque a veces puede emplearse cmake.

.gz o .gzip o .bzip2:

Con **GNU Zip** se pueden comprimir paquetes de tipo .gz o .gzip. Estos son tratados de forma similar a los paquetes comprimidos con BSD Zip 2 con extensión .bzip2. Para tratar este tipo de paquetes debemos tener disponibles las herramientas unzip y bunzip2 en nuestro sistema:

```
1gunzip -c nombre_paquete.gz
2bunzip2 nombre_papuede.bz2
```

El resto es **similar a los pasos vistos** con los tarballs anteriores...

Asegurate de ver los ficheros README o INSTALL presentes.

.tar.lzma, .tlz:

Tanto si aparece por su nombre largo, .tar.lzma, como si aparece por su nombre corto .tlz, estos paquetes utilizan el algoritmo de compresión Lempel-Ziv-Markov y para extraerlos e instalarlos, debes teclear en la consola (previamente se necesita tener instalado el paquete lzma):

```
1unlzma nombre_fichero.lzma
```

o

```
1lzma -d file.lzma
```

o

```
1tar --lzma -xvf file.tlz
```

o

```
1tar --lzma -xvf file.tar.lzma
```

Dependiendo del formato en el que se nos presente el paquete. Luego puedes mirar algún fichero de texto en su interior con instrucciones o seguir los pasos que hemos descrito para instalar los otros tarballs (./config, make, make install). Otra buena práctica es mirar en la **web**

del desarrollador, donde aparecen tutoriales de como se instalan los paquetes o existen sitios Wiki con multitud de información.

Nota: también puedes instalar ciertos paquetes empaquetados con una herramienta denominada **installpkg.*

Cómo instalar paquetes binarios:

.jar:

Para instalar **paquetes java** es bastante sencillo. Los requisitos son evidentes, tener instalada la máquina virtual Java de Oracle (ya sea la JRE o JDK). Para instalarlo debemos hacer clic con el botón derecho del ratón sobre él y seleccionar “Abrir con otra aplicación” en el menú desplegable. Aparecerá una ventana con una lista de aplicaciones de nuestro sistema y una línea de formulario abajo para escribir una. Pues en dicho espacio escribes “*java -jar* ” sin comillas, incluido el espacio tras jar que he dejado yo. Luego presionas sobre el botón “Abrir” y se debería ejecutar sin problema. Como puedes comprobar no es necesario instalarlo.

.bin:

Los podemos ejecutar haciendo doble clic sobre ellos para abrirlos, si previamente le hemos dado **permisos de ejecución**. Para ello haz clic con el botón derecho del ratón sobre el fichero y luego ve a “Propiedades” para asignarle permisos de ejecución en la pestaña «Permisos». También se puede instalar desde la consola haciendo lo siguiente:

```
1cd directorio_donde_está_el_binario
2./nombre_binario.bin
```

.run:

Para los **.run** procederemos de forma similar a los **.bin**. Este formato es muy utilizado para drivers, como por ejemplo los AMD Catalyst Center. Para instalarlo se puede usar la consola:

```
1cd directorio_donde_está_el_paquete  
2sh ./nombre_paquete.run
```

Recuerda asignarle previamente permisos de ejecución. Además, algunos necesitan ejecutarse con privilegios, en tal caso hazlo como root o con sudo.

Si quieres instalar el **.run en modo gráfico**, puedes hacer clic con el botón derecho del ratón sobre él y seleccionas “**Propiedades**”, luego en la pestaña “**Permisos**” marcas “**Permitir ejecutar el archivo como un programa**” y aceptas para cerrar. Ahora al hacer doble clic sobre el **.run** verás que se abre un instalador muy similar a los de Windows (tipo Siguiente, Siguiente, Aceptar...).



Cómo instalar scripts:

.sh:

En Linux también podemos encontrar **scripts con extensiones .sh o los .py**. Para instalar este tipo de scripts nos dirigiremos al directorio donde se encuentra el script con el comando “cd” como hemos visto anteriormente. ¡Ojo! Si el script estuviese empaquetado, primero desempaquetalo o descomprímelo. Luego, puedes darle permisos de ejecución como ya sabes (puedes hacerlo en modo gráfico o desde el terminal con el comando “*chmod +x nombre_script*» sin comillas).

Una vez tengan permisos de ejecución, desde el terminal:

```
1 sh nombre_script.sh
```

o

```
1 ./nombre_script.sh
```

.py:

Para los ficheros con **extensión .py** se debe llamar al interprete del lenguaje de programación Python. Para ello, teclea en la consola esto:

```
1 python nombre_script.py install
```

Otros:

Existen otros tipos de ficheros y paquetes para instalar programas en Linux. Ciertos paquetes de BSD, Solaris, Mac OS X, y otros *nix se pueden instalar en Linux. Un ejemplo de ello son los **.pkg de Solaris**. Para instalar los .pkg se puede hacer clic sobre ellos con el botón derecho del ratón, ir a “*Propiedades*” y “*Permisos*” y asignarle permisos de ejecución. Luego haces doble clic sobre ellos para instalarlos.

También existen herramientas como **Alien** para convertir de un formato a otro, por ejemplo de rpm a deb, etc. Esto no es muy

recomendable y en ocasiones puede generar problemas. Así que no te lo recomiendo.

Continuando con el galimatías de los paquetes en Linux, decir que existen más de los vistos aquí, pero son más raros e inusuales. Un ejemplo de rareza es el **.slp** que usan desde el proyecto Stampede Linux. Para transformar .slp en otros formatos más cotidianos puedes emplear Alien (previamente instalado Alien) así:

```
1sudo alien nombre_paquete.slp nombre_paquete.extensión_nueva generated
```

Por ejemplo, para transformar de .slp a rpm:

```
1sudo alien miprograma.slp miprograma.rpm generated
```

Pueden dejar sus comentarios con peticiones, **dudas o comentarios**. Si tienen algún problema siguiendo los pasos de este tutorial, Estaré encantado de ayudarlos.

OTRAS FORMAS DE INSTALAR SOFTWARE

PPA: Es un Archivo de Empaquetado Personal. Se utilizan a menudo para instalar versiones más avanzadas de los programas que cuenta en tu sistema y además para instalar nuevos programas que no están en tu paquetería. Los PPA son compatibles con Linux Mint y Ubuntu. La mayoría de los PPA están en [Launchpad](#), que es una web que plataforma de colaboración de software.

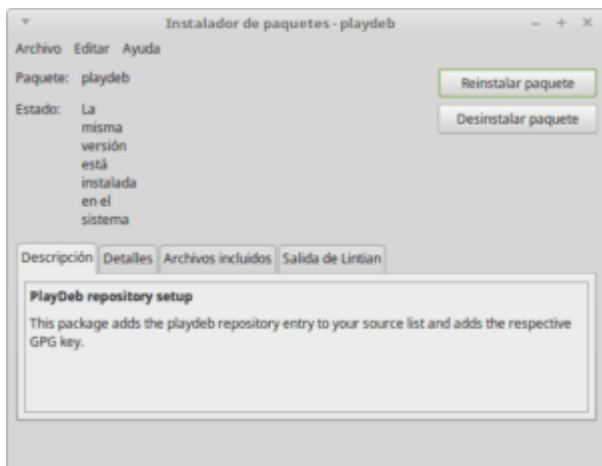
Los PPA se instalan desde la terminal o desde interfaz gráfica. En Linux Mint para añadir un PPA abría que abrir Orígenes de Software desde el Menú u clic sobre el botón PPA.



También, y lo más habitual, es instalar los PPA desde la terminal. Por ejemplo, si quisiéramos instalar Shutter desde un PPA, abríramos la terminal y añadiríamos la siguiente secuencia de comandos.

```
SUDO ADD-APT-REPOSITORY  
PPA:SHUTTER/PPA #AÑADIMOS  
EL PPA  
SUDO APT-GET UPDATE  
#ACTUALIZAMOS EL SISTEMA  
PARA QUE RECONOZCA LOS  
CAMBIOS  
SUDO APT-GET INSTALL SHUTTER -  
Y #INSTALAMOS, LA OPCIÓN -Y  
EVITA PEDIR CONFIRMACIÓN AL  
USUARIO
```

Paquetes .deb: Los paquetes deb son lo más parecido a la forma de instalar programas en Windows. La idea es la misma: buscar, descargar y ejecutar para abrir. En Ubuntu se abrirá el Centro de Software o si tienes instalado otro gestor, ese concreto gestor. En Linux Mint viene por defecto Gdebi, que es un programa que instala los paquetes .deb muy fácilmente. Está disponible también en Ubuntu, de hecho es una de los primeros programas que instalo en Ubuntu. Nota: Para instalarlo en Ubuntu sudo apt-get install gdebi -y desde una terminal.



A veces queremos instalar un programa pero viene en muchos archivos .deb y no sabemos el orden correcto. Como en OpenOffice. Para estos casos es mejor recurrir a la terminal. Nos situamos en el directorio donde están los archivos .deb y ejecutamos el comando `sudo dpkg -i *.deb`

Código fuente

Muchos desarrolladores disponen sus programas en este formato nativo. Lo que obtenemos, en la mayoría de los casos en un archivo comprimido con extensión tar.gz o similar, al descomprimir tendremos unos archivos con un README, que hay que leer muchas veces ya que contienen las instrucciones para el compilado y otro llamado INSTALL.

Por orden había que hacer:

Descomprimir el código fuente. Desde la terminal o de manera gráfica.
Resolver las dependencias. Usando Synaptic por ejemplo o la terminal.
A continuación deberemos configurar los archivos que nos permitirán compilar el programa.

./CONFIGURE

Luego escribimos:

MAKE

Y finalmente, instalamos con:

MAKE INSTALL

Este proceso tardará dependiendo de la cantidad de elementos a instalar, velocidad del procesador, memoria, uso del sistema, etc.

Archivos sh y .bin

Lo más fácil es abrir una terminal (Ctrl+ Alt+ T) y ubicarnos en el archivo. Una vez que estés en la ruta por ejemplo el archivo está en tu Escritorio.

CD ESCRITORIO

Habrás que otorgarle permisos de ejecución:

Si tenemos un archivo .sh

*SUDO CHMOD A+X
NOMBRE_ARCHIVO.SH*

Si tenemos un archivo .bin

*SUDO CHMOD A+X
NOMBRE_ARCHIVO.BIN*

WINE (Wine Is Not an Emulator)

Si tienes las librerías de Wine en tu sistema podrás ejecutar programas que son nativos de Windows que usan la extensión .exe o .msi. No es objeto de este pequeño tutorial informar de cómo utilizar Wine.



PlayOnLinux

PlayOnLinux es un programa que os permite instalar y usar fácilmente numerosos juegos y programas previstos para correr exclusivamente en Windows® de Microsoft®. No es objeto de este pequeño tutorial informar de cómo utilizarlo.

Applmage



Applmage no es, técnicamente hablando, un administrador de paquetes; tampoco una tienda de aplicaciones. Quizá entendamos mejor lo que ofrece Applmage si recordamos que, **entre 2011 y 2013 se le conoció con el**

nombre de "PortableLinuxApps" (antes de eso, desde su lanzamiento en 2004, se le denominó "Klik").

En resumen, **se trata de una imagen de la aplicación (similar a un archivo .iso)** que no requiere de instalación ni -lo que es más importante- de permisos de administrador para funcionar: ofrece aplicaciones portables, empaquetadas en un único archivo con todas sus dependencias, al margen de la distribución utilizada.

Cuando usemos un archivo .appimage, **sólo deberemos perder tiempo otorgándole permisos de ejecución**, automáticamente se montará en el sistema de archivos del espacio de usuario (lo que lo convierte en compatible con sistemas de archivo inmutables, como los basados en OSTree).

Algunas aplicaciones **ofrecerán la opción al iniciarlas de 'instalar un archivo de escritorio'**, lo que lo integrará en los menús de aplicaciones de nuestra distribución. Sólo en estos casos su "desinstalación" requerirá de algo más que mandar a la Papelera el archivo ejecutable.

Puedes descargar imágenes AppImage desde [AppImageHub](#), pero no existe ningún repositorio centralizado.

Snap

Snap es [una iniciativa impulsada por Canonical](#) para su uso en su propia distribución Linux para móviles, [la desaparecida Ubuntu Touch](#), en 2014. Si bien desde entonces se ha exportado a múltiples



distribuciones relevantes (aunque **su excesiva dependencia de Canonical** aún [lastra su adopción](#)).

Su objetivo era lograr un formato único de distribución de software que pueda usarse por igual en PCs de escritorio y en dispositivos IoT. Como en el caso de Appliance, **cada Snap empaqueta dentro del mismo todas las dependencias del software en cuestión**, facilitando que funcione en cualquier equipo.

Al contrario que Appliance, los snaps están diseñados para admitir actualizaciones, sin obligarnos a borrar y sustituir el paquete por uno más nuevo. Más aún, esta actualización se lleva a cabo en segundo plano incluso si estamos usando la aplicación.

Es un sistema seguro, porque **los paquetes vienen firmados** y, una vez se empieza a usar la aplicación, ésta lo hace en un entorno aislado, con acceso limitado a los recursos del equipo.

El programa que permite gestionar los snaps es Snapd, y su tienda de aplicaciones, [Snapcraft](#).

Flatpak



FLATPAK

flatpak.org

El creador de [FlatPak](#), Alexander Larsson, se inspiró en el primer antecesor de Appliance, Klik, para crear un sistema de paquetes centrado en la ejecución del

software dentro de entornos aislados, donde pudieran funcionar sin privilegios de root. Larsson lanzó en 2015 un sistema llamado *xdg-app* que **se terminó convirtiendo en Flatpak un año más tarde, con el apoyo de Red Hat.**

Su principal diferencia con respecto a los otros dos sistemas de gestión de paquetes es que **no empaqueta el software que nos interesa junto a todas sus dependencias**: eso puede obligarnos a instalar numerosos paquetes, pero también **reduce sensiblemente el tamaño de los mismos.**

Otra diferencia es **su alto nivel de integración con los entornos de escritorio KDE y Gnome y con el estándar FreeDesktop**, lo que facilita la integración de las aplicaciones instaladas mediante este sistema con las herramientas gráficas de los principales escritorios de Linux. El apoyo de FreeDesktop, que incluso aloja la web del proyecto Flatpak, le ha granjeado muchos apoyos.

El principal repositorio de paquetes Flatpak es [Flathub.org](https://flathub.org) (aunque no depende de ninguna 'tienda de aplicaciones' en concreto).