UD7. Distribución de las aplicacións

RA7. Preparar aplicaciones para su distribución, para lo que utiliza y evalúan herramientas específicas.

– CA2.1. Se empaquetan los componentes que requiere la aplicación.
– CA2.2. Se personaliza el asistente de la aplicación.
– CA2.3. Se empaqueta la aplicación para ser instalada de manera típica, completa o personalizada.
– CA2.4. Se generan los paquetes de la instalación usando un entorno de desarrollo.
– CA2.5. Se generan paquetes de instalación usando herramientas externas.
– CA2.6. Se generan paquetes de instalación de forma desatendida.
– CA2.7. Se prepara el paquete de instalación para que la aplicación se pueda instalar de forma correcta.
– CA2.8. Se prepara la aplicación para descargarla desde un servidor web y ejecutarla.

BC7. Distribución de aplicaciones

- Componentes de una aplicación. Empaquetado.
- Instaladores. Paquetes autoinstalables. Herramientas para crear paquetes de instalación.
- Parámetros de la instalación. Personalización de la instalación: logotipos, fondos, diálogos, botóns, idioma, etc.
- Asistentes de instalación e desinstalación.
- Tecnologías para la automatización de la descarga y ejecución de aplicaciones desde servidores web de aplicaciones.

1. Componetes de una aplicación. Empaquetado

El **empaquetado de aplicaciones** consiste en proporcionar a los futuros usuarios las aplicaciones en forma de paquetes, a los que se suele llamar en inglés ***software bundle o application bundle.*** Estos paquetes están formados por:

- los **programas ejecutables** de la aplicación,
- las **bibliotecas necesarias** de las que depende
- otros tipo de **ficheros** (como imágenes, bases de datos, ficheros de audio, traducciones y localizaciones, etc.),

Todos en ellos dentro del mismo fichero forman un todo o un conjunto.

Las biblotecas de las que depende el programa pueden haber sido enlazadas, como ya hemos visto en unidades anteriores, tanto de **forma dinámica como también estática**. Independientemente de ello, el usuario percibe que el paquete como un conjunto que representa al programa en sí, cuando en realidad incluye varios ficheros.

Una de las mayores ventajas de un correcto empaquetado de aplicaciones es que permite **evitar los problemas de las dependencias** tanto a la hora de instalar la aplicación como a la hora de usarla, ya que cada paquete lleva consigo sus dependencias, y la instalación o desinstalación de otro software no va a afectar a las dependencias de dicho paquete. De ahí radica su principal **objetivo evitar la problemática de las dependencias**, y que la aplicación se puede trasladar de un computador a otro sin necesidad de reinstalarla, ya que el paquete de la aplicación contiene todos los ficheros necesarios para ejecutarla.

Sin embargo, como **desventaja** se presenta que estos paquetes **ocupan mucho más espacio** en el disco, especialmente si el paquete incluye bibliotecas.

Por lo general cada distribución tiene su propia forma de empaquetar sus aplicaciones:

- **Linux:** tenemos dos tipos de paquetes que sobresalen del resto:
  - **rpm**: (Redhat Package Manager) de la familia de RedHat (RHEL, Fedora,CentOS), Mandriva, Suse
  - **deb: d**e la familia de Debian (Debian,Ubuntu, y derivados)
- **Windows:** tomando como base su IDE Visual Studio el formato de empaquetado es:
  - **msi:** se definen como **instaladores de Microsoft,** a saber, aquellos paquetes de software que contienen la información necesaria para automatizar su instalación, minimizando la intervención manual del usuario, ya que toda la información iría contenida en el propio fichero **msi**. La información de instalación, y a menudo los archivos mismos, son empaquetados en **paquetes de instalación**, bases de datos

estructuradas como **OLE Structuree Storage** (almacenamiento estructurado de ficheros) y comúnmente conocido como "MSI files" por su extensión de archivo. El sistema de archivos **msi** es muy útil cuando queremos distribuir aplicaciones en equipos pertenecientes a un dominio **Windows**.

- **Java:** el principal es jar.
  - **jar:** es un tipo de archivo que permite ejecutar aplicaciones escritas en el lenguaje Java. Las siglas están deliberadamente escogidas para que coincidan con la palabra inglesa "jar" (tarro). Los archivos JAR están comprimidos con el formato **zip** y cambiada su extensión a .jar. Es el más popular entre los dispositivos móviles y en los SO más importantes.

2. Instaladores. Paquetes autoinstalables. Herramientas para crear paquetes de instalación.

WINDOWS

En equipos **WINDOS** existen los ficheros que comúnmente son llamados ejecutables. Los primeros fuenos los llamados **bat** (extensión de un fichero formado por un lote de órdenes Dos**, que ya no se usan**), los **exe** y los **msi,** éstos dos últimos son utilizados con mayor frecuencia en el entorno gráfico de Windows.

La herramienta encargada de llevar a cabo la instalación es **Windows Installer** que es un motor para la instalación, mantenimiento y eliminación de programas en plataformas **Windows.**

Los paquetes **MSI (Microsoft Installer) se definen como instaladores de Microsoft o** paquetes de software que contienen la información necesaria para automatizar su instalación, minimizando la intervención manual del usuario, ya que toda la información iría contenida en el propio fichero "msi".

Una de las ventajas de los paquetes msi es su facilidad para **la distribución de software** desde servidores **Windows Server.** Estos paquetes se pueden crear incluso con *software de terceros* para luego instalarlos en equipos en red pertenecientes a un dominio.

Un paquete describe la instalación completa de un producto ya que Windows Installer no maneja dependencias entre productos y está identificado por un **GUID**. El paquete stá compuesto de **componentes** agrupados dentro de sus **características**.

Un **componente** es la mínima parte de un producto. Cada componente es tratado por Windows Installer como una unidad. Los componentes pueden contener *archivos, grupos de archivos, directorios, componentes COM, claves del registro de Windows, accesos directos y otro tipo de datos*. El usuario final no interviene directamente con los componentes.

También los componentes al estar identificados globalmente por GUID's, ello permite que un mismo componente sea compartido entre varios del mismo paquete o de múltiples paquetes, idealmente a través del uso de la unión de módulos (aunque para trabajar correctamente, diferentes componentes no deberían compartir ningún sub-componente).

Una **ruta maestra** es un fichero específico, clave de registro, o fuente de datos ODBC que el autor del paquete especifica como crítico para un componente dado. Como las rutas maestras más utilizadas son en forma de fichero, se suele utilizar el término *fichero maestro (key file)*. Un componente puede contener, a lo sumo, una ruta maestra; si un componente no tiene establecida de manera explícita una ruta maestra, el directorio destino del componente es tomado como la ruta maestra. Cuando se ejecuta una aplicación basada en MSI, Windows Installer comprueba la existencia de estos **fichero críticos o claves de registro** (es decir, las rutas maestras). Si existe un desajuste entre el estado actual del sistema y el valor especificado en el paquete MSI (e.g., un fichero maestro desaparecido), entonces la característica asociada **es reinstalada.** Este proceso es también conocido como *auto-reparación*. Dos componentes no pueden utilizar la misma ruta maestra u otra.

InstallShield es una herramienta de software para crear instaladores. InstallShield se utiliza sobre todo para instalar software del escritorio y las plataformas de servidor de Windows, pero también se puede usar para administrar aplicaciones y paquetes de software en una amplia gama de móviles y portátiles. Fue desplazado por **Windows Installer.**

Por otro lado para la creación del instalador **msi de Windows,** Visual Studio tiene su propia herramientas **VSI (Visual Studio Installer).** Con el Visual Studio Installer, se puede desarrollar un proyecto de instalación creando un archivo **.wip** como una parte integral de su solución. El archivo de paquete de instalación de Windows (.msi), se construye a partir del proyecto de Visual Studio Installer (.wip) que contiene todos los datos e instrucciones necesarias para instalar la aplicación.

En la red hay diferentes manuales que explican su uso, cosa no complicada ya que se trata de un asistente.

LINUX

Hay 3 formas de instalar paquetes en GNU/Linux:

- **Compilar el paquete:** Esta es la forma clásica, y antigua, de instalar paquetes. Consiste en bajar el código fuente, comprimido en un archivo **.tar.gz o .tar.bz2.**

  Una vez bajado, entramos en la consola (shell) y nos movemos hasta el directorio donde tengamos el paquete.Si el paquete está en formato .tar.gz escribimos:

```
# tar -xzvf archivo.tar.gz          //para descomprimir el paqute
```

Si está en .tar.bz2 escribimos:

```
# bzip2 -dc archivo.tar.bz2 | tar -xv
```

Una vez hecho esto, hay que entrar en el directorio creado y compilar el código para obtener la aplicación funcional y que consiste en escribir en la siguiente línea de comandos lo siguiente:

```
# ./configure

# make

# make install
```

Lo que estamos haciendo es compilando el programa a partir de código fuente.

Uno de los principales problemas de este método es si el paquete tiene dependencias, es decir, si depende de algún otro paquete para que funcione correctamente. En ese caso, habrá que instalarlos manualmente.

- **Paquetes .deb y .rpm**:Los paquetes .deb y .rpm son un método de instalación muy efectivos para sus respectivas distribuciones.

  Los paquetes **.deb** son paquetes que se pueden instalar en la distribución Debian y derivados (Ubuntu, Kubuntu...).

  Los .rpm (RedHat Package Mangaer) son los de la distribución Red Hat y dervidados (OpenSuse, Mandriva, Fedora...).

  Un paquete .rpm no lo podremos instalar en la distro Debian o derivados, y un .deb tampoco en RedHat y derivados. No obstante, existen programas, por ejemplo, uno llamado **'Alien'** que permite convertir un paquete .rpm a .deb y viceversa.

  Para instalar un paquete .deb entramos la siguiente línea de comandos en la consola (Situándonos en el directorio donde está el paquete:

  ```
  # dpkg -i nombredelpaquete.deb
  ```

  Para **crear un paquete deb** ya hay asistentes que facilitan bastante su creación. Uno de ellos es **checkinstall.** Los pasos son los siguientes:

  Para empezar, tenemos que **instalar checkinstall**. Así que hacemos (*como root*):

  ```
  # apt-get install checkinstall
  ```

Lo siguiente es ir a la **carpeta** en la que tenemos el **código de la aplicación**, y abrir una **terminal**. Ejecutamos los siguientes comandos, uno a uno:

```
# ./configure
```

```
# make
```

Con "*./configure*" se configuran los paquetes para nuestra distribución y se crea un "*Makefile*" (*un archivo que contiene instrucciones de compilación*), y con "*make*" se compila el código y deja los binarios, librerías, etc en la carpeta "*src*". Ahora, antes de continuar, es recomendable **no tener instalada** la aplicación de la que se hará el paquete. Si lo está:

```
# make uninstall
```

Y entonces es momento de comenzar con la parte importante, el uso de **checkinstall**. En esa misma terminal, escribimos:

```
# checkinstall
```

Y se abrirá el "*asistente*" de *checkinstall*. En él podemos modificar **la información** que tendrá el paquete que crearemos. Las opciones que podemos modificar son:

- **Maintainer**: el desarrollador principal del paquete.
- **Summary**: una descripción del paquete.
- **Name**: nombre que quieres darle al paquete.
- **Versión**: versión del paquete.
- **Release**: viene siendo la versión principal del paquete, podemos dejarlo como venga.
- **License**: licencia de la aplicación, es preferible no tocarlo.
- **Group**: grupo por el cuál fué creado, podemos dejarlo como está.
- **Architecture**: arquitectura de procesador del paquete.

- **Source location**: nombre de la carpeta (solo la carpeta, no la ruta entera) en la que está el código del paquete.
- **Alternate source location**: no es necesario modificarlo.
- **Requires**: dependencias que deben ser instaladas para su correcto funcionamiento.
- **Provides**: nombre del paquete que provee, no es necesario modificarlo.
- **Conflicts**: paquetes con los que entra en conflicto.
- **Replaces**: paquetes a los que reemplaza.

Cada una tiene **un número** a su izquierda, así que para editarla solo **escibimos su número**. No es necesario modificar todas, pero si las más importantes como Maintainer, Summary, Name y Version.
Una vez hayamos modificado lo que queremos, presionamos [**Enter**] (*sin ningún número previo*) y comenzará a **compilar e instalar** el paquete. Cuando haya terminado, en el directorio donde compilamos habrá aparecido un **paquete .deb** de la aplicación, listo para instalar. Asegurarse en "*Version*", **no** hayan letras. Eso suele evitar que se cree el paquete. Es posible que "*Requires*" dé un fallo, con dejar el espacio **en blanco**.

En RedHat paraara instalar un paquete -**rpm** introducimos:
rpm --install nombredelpaquete.rpm
Para crear un paquete rpm:
yum groupinstall "Development Tools"

```
# yum install rpmdevtools
# yum install rpmlint
# rpmdev-setuptree
```

Ésto nos creará un entorno de trabajo para crear los RPM's o "Árbol del Proyecto". Necesitaremos dos cosas: un Archivo **".spec"** y el **código fuente del programa** para instalar. El archivo ".spec" lo puedes conseguir de cualquier "src.rpm" creado para tu versión de Fedora (con una versión anterior del código fuente por ejemplo) o con un "src.rpm" para una versión antigua de fedora, una fuente para dicho tipo de paquetes está en: **http://www.rpmfind.net/**

Debemos abrir el paquete "src.rpm" que hayas descargado con un gestor de archivadores, ahí dentro encontrarás el **".spec"**. Debido a que lo que haremos será construir un Nuevo RPM desde código fuente más reciente, generalmente sólo debemos cambiar el campo *"Version"* y agregar un *"Changelog"* y fijarse en el campo *"Release".* Por ejemplo usar el paquete de **Kmess 2.0.5-2** de **Fedora 14** para crear el **2.0.6.1-0** usable en el mismo Fedora, en ése caso, sólo modificarísmos el apartado *Version* por el nuevo, quitando el **2** de *Release* y poniendo un **0** ("Release" es el número que aparecerá después del guión) y agregando un *changelog* copiando uno de los ejemplos anteriores y modificándolo según aplicó con el *changelog* oficial.

Otra cosa que es chequear el archivo ".spec" en su apartado de *#BuildRequires* ya que todo lo que esté en éste apartado son dependencias sin las cuáles el paquete no podrá ser creado. Instalamos todos los paquetes requeridos con un:

```
# yum -y install paquete1 paquete2 paquete3
```

Vamos a la carpeta "rpmbuild" y luego en la carpeta de SOURCES debemos colocar el código fuente de nuestra aplicación, el "tarball" para explicarme mejor. Por otro lado, debemos copiar nuestro ".spec" modificado/recientemente creado en SPECS y continuamos con el siguiente paso.

Usamos una pequeña herramienta llamada *rpmlint* para nuestro cometido. Ésta herramienta nos avisará de errores en la estructura del archivo que acabamos de modificar/escribir y su uso es bastante sencillo:

```
# rpmlint program.spec
```

Se nos mostrará un Output donde nos dará información acerca de nuestro archivo. Si hay errores o advertencias, simplemente tenemos que hacer caso y modificar según se necesite.

Ya tenemos todo listo para la construcción:

```
# rpmbuild -ba program.spec
```

Tras un tiempo de compilado y trabajo, tendremos un paquete RPM listo para usar, su "src.rpm" y también los paquetes adicionales que se hayan creado.

Terminada la creación del paquete, si todo salió bien, podremos irnos al directorio *rpmbuild* y en el subdirectorio RPMS encontraremos los binarios de nuestra aplicación y los extras que la acompañen, mientras que en el directorio SRPMS encontraremos los paquetes "src.rpm" para la misma aplicación. Tener a la mano éstos dos tipos de paquetes es lo que debería importarte... Ahora simplemente se chequea el RPM final generado (el que usaremos para la instalación de la aplicación) con rpmlint:

```
# rpmlint *.rpm
```

El último paso tanto en **deb** como **rpm** es subirlos a una web de repositorios para que puedan ser descargados con **aptitude o apt-get** en el primer caso o con **yum** en el segundo caso.

3. Parámetros da instalación. Personalización da instalación: logotipos, fondos, diálogos, botóns, idioma, etc.

Se verá en las prácticas.

4. Asistentes de instalación y desinstalación

WINDOWS

Una de las herramientas “ocultas” de Windows es **iexpress.**

IExpress es una utilidad de Microsoft incluido con varias ediciones de los sistemas operativos Windows (32-para arriba y 64 hacia arriba-): Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7 y Windows 8. También fue incluido como parte de todos los de Internet Explorer Administration Kit Lanzamientos 4, 5 y 6, y era parte de toda la instalación de Internet Explorer 6.

Iexpress.exe se utiliza para crear auto-paquetes de un conjunto de archivos. Estos paquetes pueden utilizarse para instalar aplicaciones, el ejecutable, los paths, otros componentes del sistema, o bootstrappers de configuración.

De esta forma se puede utilizar para la distribución de paquetes de instalación a múltiples ordenadores con Windows locales o remotos. Crea **ejecutable autoextraíble (.exe) o un archivo contenedor comprimido (. CAB)** utilizando la interfaz proporcionada que facilita la automatización.

Los archivos pueden ser modificarse con cualquier texto plano / editor ASCII, como el Bloc de notas. Todos los archivos autoextraíbles creados por el uso de Iexpress se comprimen con el MakeCab (MAKECAB.EXE) herramienta, y se extraen mediante el Wextract (wextract.exe) de Microsoft.

Iexpress.exe se encuentra en la carpeta System32 de Windows. La interfaz frontal (IExpress Wizard) se puede iniciar manualmente escribiendo IExpress en la ventana Ejecutar del menú Inicio. También se puede utilizar desde la línea de comandos (o archivo por lotes en la consola) para crear paquetes de instalación personalizada, con el tiempo desatendida (funcionamiento automático):

c:> IEXPRESS / N letraDeUnidad ₩ directory_name ₩ file_name.SED

A través de la creación de interfaz del Asistente de IExpress le permite al usuario especificar un **título para los paquetes**, **añadir la solicitud de la licencia de usuario final** que debe ser aceptado como una orden para permitir la extracción, seleccione los archivos que se archivan, la visualización avanzada ventana de opciones, y finalmente, especificar un mensaje que se mostrará al finalizar.

La **desinstalación de software** es el proceso de revertir los cambios producidos en un sistema por la instalación del mismo. Por ello no solo deben ser borrados los archivos, sino también cambios en otras aspectos del software, como por ejemplo, eliminar usuarios que hayan sido creados, retirar derechos concedidos, borrar directorios creados hasta llevar la contabilidad en un sistema de gestión del sistema, en el caso de Windows el Registro.

Debido a la creciente complejidad de sistemas operativos y sus interfaces (API), la desinstalación

de software puede ser no solo contraproductiva sino también poner en peligro la estabilidad del sistema. Por esta razón **la calidad de un software** no solo depende de sus efectos productivos o creativos sino también de su **capacidad de integración en el sistema operativo y compatibilidad con otros programas**. El desarrollador del software debe ofrecer una función para deinstalar su software sin dañar o desestabilizar el sistema.

Cada vez es más difícil la desinstalación, dado que muchas bibliotecas se comparten entre aplicaciones de diferentes productores de software que utilizan enlaces duros o simbólicos a través del directorio.

En sistemas de alta complejidad, el esfuerzo para desinstalar un programa puede ser mayor que el de la instalación.

En Windows la desinstalación mejora si utilizamos software que además de eliminar los archivos del software eliminen las entradas en los registros. Esto es debido a que el desinstalador que lleva Windows (*Agregar y Quitar Programas*) no es muy fiable en ese aspecto. Existen varias herramienta como por ejemplo es **Ccleaner, RegCleaner, Revo Uninstaller o Iobit Uninstaller** entre otras, que además de desinstalar hacen búsquedas en la base de datos del Registro **para eliminar aquellas entradas** que no están asociadas a ningún software instalado.

En cuanto a Linux los comandos para desinstalar aplicaciones son los siguientes:

 # apt-get remove "nombre-del-paquete" //desinstalamos el paquete

 # apt-get purge "nombre-del-paquete" //borra los archivos de configuración

 # apt-get clean "nombre-del-paquete" // borra los archivos descargados con la aplicación

6. Tecnologías para la automatización de la descarga y ejecución de aplicaciones desde servidores web.

Un **sistema de gestión de paquetes,** también conocido como gestor de paquetes, es una colección de herramientas que sirven para automatizar el proceso de **descarga, instalación, actualización, configuración y eliminación de paquetes de software**. El término se usa comúnmente para referirse a los gestores de paquetes en sistemas UNIX, especialmente GNU/Linux, ya que se apoyan considerablemente en estos sistemas de gestión de paquetes.

En estos sistemas, el software se distribuye en forma de paquetes, frecuentemente encapsulado en un solo fichero. Estos paquetes incluyen otra información importante, además del software mismo, como pueden ser el nombre completo, una descripción de su funcionalidad, el número de versión, el distribuidor del software, la suma de verificación y una lista de otros paquetes requeridos para el correcto funcionamiento del software. Esta metainformación se introduce normalmente en una base de datos de paquetes local.

Sistema de Gestión de Paquetes	Instalador
Forma parte del sistema operativo.	Cada producto viene unido a su propio instalador.
Usa una única base de datos de instalación.	Rastrea su propia instalación
Puede verificar y administrar todos los paquetes sobre el sistema.	Sólo trabaja con su propio producto.
Un único vendedor de sistema de administración de paquetes.	Múltiples vendedores de instalador.
Un único formato de paquetes.	Múltiples formatos de instalación

Los sistemas de gestión de paquetes tienen la tarea de organizar todos los paquetes instalados en el sistema y se encargan de mantener su usabilidad. Esto se consigue combinando las siguientes técnicas:

- Comprobación de la suma de verificación para evitar que haya diferencias entre la versión local de un paquete y la versión oficial.
- Comprobación de la firma digital.
- Instalación, actualización y eliminación simple de paquetes.
- Resolución de dependencias para garantizar que el software funcione correctamente.
- Búsqueda de actualizaciones para proveer la última versión de un paquete, ya que normalmente solucionan bugs y proporcionan actualizaciones de seguridad.
- Agrupamiento de paquetes según su función para evitar la confusión al instalarlos o mantenerlos.

Muchos de los sistemas de gestión de paquetes ampliamente utilizados utilizan backends simples para instalar los paquetes. Por ejemplo, YUM utiliza RPM como backend y APT utiliza dpkg.

En los sistemas donde las aplicaciones comparten módulos, como en la mayor parte de las distribuciones de GNU/Linux, **la resolución de dependencias al instalar y desinstalar software se convierte en una necesidad.** Algunos de los sistemas de gestión de paquetes más avanzados tienen la capacidad de desinstalar los paquetes recursivamente o en cascada, de forma que se eliminan todos los paquetes que dependen del paquete a desinstalar y todos los paquetes de los que el paquete a desinstalar depende, respectivamente.

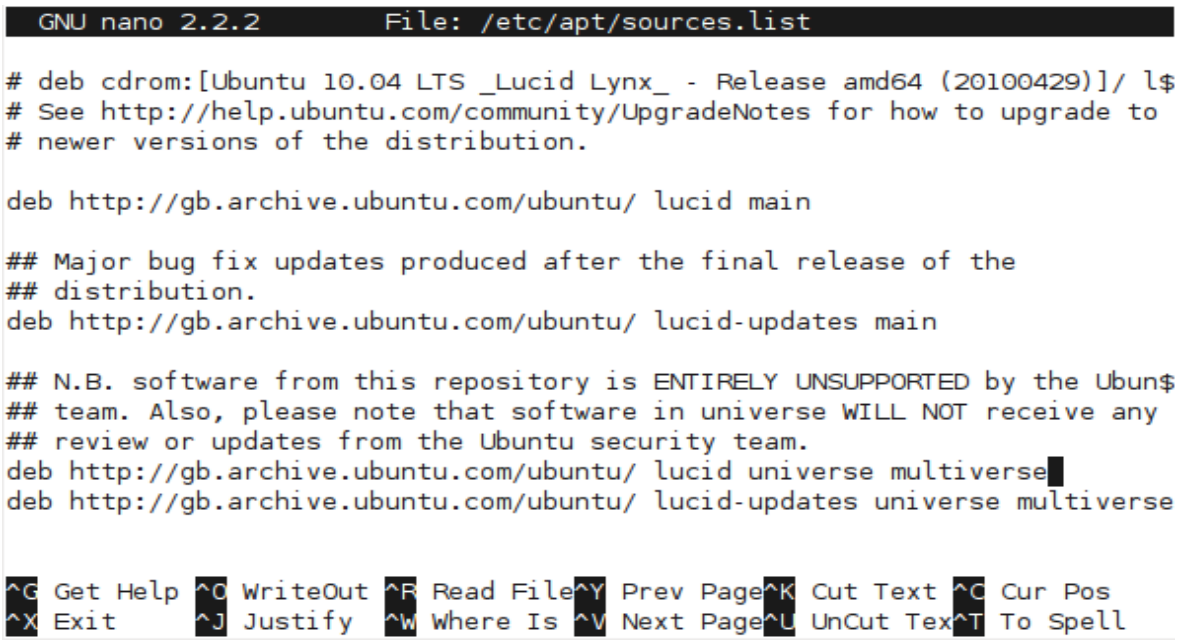
Es común que un administrador **instale software que no está disponible en los repositorios provistos**. Algunos ejemplos pueden ser una nueva versión de una aplicación que todavía no está disponible en la distribución o una alternativa distinta de la elegida por la distribución. Si este software adicional sólo se distribuye en forma de código fuente, la instalación requerirá **la compilación del código**. Sin embargo, la instalación de este software adicional en el sistema **ocasionará que el estado del sistema y la base de datos del gestor de paquetes no estén**

sincronizados, por lo que el administrador deberá tomar medidas adicionales para asegurar que el sistema de gestión de paquetes se mantenga al día, puesto que éste no es capaz de hacerlo automáticamente.

Alien es un programa que convierte entre los diferentes formatos de paquetes de GNU/Linux. Soporta la conversión entre Linux Standard Base, RPM, deb, Stampede (.slp) y paquetes de Slackware (.tgz).

Otra problemática aparte de la **actualización de software es la actualización de ficheros de configuración**. Ya que los sistemas de gestión de paquetes sólo son capaces de sobrescribir o retener los ficheros de configuración, en lugar de poder aplicarles reglas de modificación. Sin embargo, hay excepciones, que normalmente se aplica al proceso de configuración del núcleo, ya que si estos son incorrectos pueden ocasionar fallos al reiniciar el sistema, pudiendo incluso hacer que el sistema no arranque. Estos problemas pueden ocasionarse cuando el formato de los **ficheros de configuración cambia.** Por ejemplo, cuando el antiguo fichero de configuración no deshabilita nuevas opciones que deberían ser deshabilitadas. Algunos sistemas de gestión de paquetes, como el dpkg de Debian, permiten configurar el software durante la instalación. En cualquier otra situación es preferible instalar los paquetes con la configuración por defecto y sobrescribirla posteriormente.

El software normalmente **se pone a disposición de los usuarios en los repositorios,** con el fin de proporcionar a los usuarios de un sencillo control sobre los diferentes tipos de software que van a instalar en su sistema y, en ocasiones, debido a razones legales o conveniencias por parte de los distribuidores.



Cuando un usuario interactúa con el gestor de paquetes para realizar una actualización, éste suele mostrar una lista de las tareas a realizar (normalmente la lista de paquetes a actualizar y, posiblemente, también los números de versión) y también es probable que permita realizar una actualización completa o bien seleccionar los paquetes que se desea actualizar. Algunos gestores de paquetes permiten indicar los paquetes que no se desea actualizar nunca o solamente cuando **estos corrigen errores importantes en la versión anterior.** A este proceso se lo suele denominar **version pinning**. Por ejemplo, yum permite esto mediante la sintaxis exclude=openoffice*, pacman con IgnorePkg=openoffice (en ambos casos para evitar la actualización de OpenOffice), mientras que las herramientas de Debian poseen una sintaxis más compleja y potente.

Sistemas basados en paquetes binarios

- **dpkg,** usado originalmente por Debian y ahora también por otros sistemas, usa el **formato .deb** y fue el primero en poseer una herramienta de resolución de dependencias ampliamente conocida, **APT.**

Instalar un paquete:

```
# apt-get install <paquete>
```

Desinstalar un paquete:

```
# apt-get remove <paquete>
```

Eliminar un paquete incluidos sus ficheros de configuración:

```
# apt-get purge <paquete>
```

Eliminar de forma automática aquellos paquetes que no se estén utilizando:

```
# apt-get autoremove
```

Actualizar un paquete a la última versión disponible en el repositorio:

```
# apt-get update <paquete>
```

Actualizar el sistema. Actualizará todos los paquetes que dispongan de una versión superior dentro de la rama instalada de la distribución:

```
# apt-get upgrade
```

Actualizar la distribución completa. Actualizará nuestro sistema a la siguiente versión disponible de la distribución:

```
# apt-get dist-upgrade
```

Descargar únicamente las fuentes de un paquete para manipularlas de forma manual:

```
# apt-get source <paquete>
```

Limpiar cachés, paquetes descargados, etc:

```
# apt-get clean

# apt-get autoclean
```

Verificar que no tenemos ninguna dependencia incumplida:

```
# apt-get check
```

Buscar un paquete en los repositorios, se puede especificar un patrón, expresión regular, el nombre exacto del paquete, etc:

```
# apt-cache search <paquete>
```

Mostrar información sobre un paquete específico (nombre del paquete, versión, dependencias…):

```
# apt-cache showpkg <paquete>
```

Mostrar información del paquete incluyendo la descripción, información del paquete como su sitio web, página de bugs…

```
# apt-cache show <paquete>
```

Mostrar dependencias de un paquete:

```
# apt-cache depends <paquete>
```

Mostrar los nombres de todos los paquetes instalados en el sistema:

```
# apt-cache pkgnames
```

- **fink**, para Mac OS X, deriva parcialmente de dpkg/apt y de ports. Esta herramienta pretende hacer más sencilla la instalación de programas libres en Mac OS X
- **el sistema RPM**, creado por Red Hat y usado por un gran número de distribuciones de GNU/Linux, es el formato de paquetes del Linux Standard Base. Para trabajar con este sistema de paquetes existen muy diversas herramientas como apt4rpm, up2date (de Red Hat), urpmi (de Mandriva), YaST (de SuSE) y YUM (usado por Fedora Yellow Dog Linux).

yum -y install paquete	Instala la última versión del paquete indicado. Instala sin pedir confirmación.
yum -y install paquete1 paquete2	Instala la última versión de los paquetes indicados, no hay límite de cuantos paquetes se pueden indicar. Instala sin pedir confirmación.
yum -y install paquete.arch	Instala la última versión del paquete indicado con la arquitectura indicada, por ejemplo: **yum install mysql.i386**.
yum -y update	Actualiza todos los paquetes en el sistema.
yum -y update --exclude=sendmail	Actualiza todos los paquetes del sistema, excepto sendmail.
yum -y update httpd	Actualiza solo el paquete indicado, en este caso el servidor Web Apache.
yum -y update opera firefox	Actualiza los paquetes indicados.
yum -y update --enablerepo=centosplus	Además de los repositorios que se tengan se habilta otro, en este caso 'centosplus', esta opción también aplica para 'install'.
yum -y upgrade	Actualiza los paquetes indicados, pero tomando en cuenta paquetes obsoletos en el cálculo de la actualización. Esta opción es idéntica a **yum -y --obsoletes update** y solo es realmente útil cuando se actualizan paquetes a través de distintas versiones de la distrubición, por ejemplo de centos4 a centos5.

yum check-update	Muestra una lista de paquetes que necesitan ser actualizados sin instalarlos.
yum info paquete	Descripción completa del paquete indicado. Ejemplo:**yum info samba**
yum info recent	Muestra información resumida de los últimos paquetes instalados o actualizados.
yum info available	Muestra información resumida de los paquetes disponibles a actualizarse.
yum list	Lista de todos los paquetes disponibles para instalación, actualización o ya instalados.
yum list | grep mysql	Muestra solo los paquetes disponibles o ya instalados de mysql.
yum list installed	Lista de todos los paquetes instalados en el sistema.
yum list available	Lista de todos los paquetes disponibles para ser instalados.
yum list updates	Lista de todos los paquetes disponibles para ser actualizados.
yum remove telnet	Remueve el paquete indicado.
yum -y remove telnet vncserver	Remueve los paquetes indicados sin pedir confirmación.
yum search paquete	Busca el 'paquete' en la base de datos de paquetes instalados o para instalar. 'paquete' puede ser una palabra parcial del paquete a buscar.
yum clean headers	Elimina todos los achivos de encabezados que yum utiliza para resolver dependencias.
yum clean packages	Cuando utilizas la opción 'update' o 'install' el paquete que se desacrga e instala o actualiza no se elimina del sistema, ocupando espacio, con esta opción eliminas esos paquetes.
yum clean all	Limpia tanto archivos de encabezados como paquetes, como utilizar las dos opciones previas, pero al mismo tiempo.
yum repolist	Lista los repositorios que se tengan de yum.

- El sistema tgz, usado por Slackware, empaqueta el software usando tar y gzip. Pero, además, hay algunas herramientas de más alto nivel para tratar con este formato: slapt-get, slackpkg and swaret.
- Pacman, para Arch Linux usa binarios precompilados distribuidos en un fichero pkg.tar.xz.

Sistemas de metapaquetes

Los siguientes sistemas unifican la gestión de paquetes para muchas o todas las distribuciones de GNU/Linux y otras variantes de Unix basándose también en el concepto de ficheros-receta:

- **klik** proporciona una forma sencilla de instalar paquetes de software para la mayor parte

de distribuciones sin los problemas de dependencias tan comunes en otros formatos de paquetes.

- **Autopackage** usa fichero .package.
- **epm**, desarrollado por Easy Software Products (creadores de CUPS), es un metaempaquetador que permite crear paquetes nativos para todas las distribuciones de GNU/Linux y otros sistemas operativos basados en Unix (.deb, .rpm, .tgz para GNU/Linux; .pkg para Solaris y *BSD, .dmg para Mac OS X, ...) a partir de un único fichero .list.

Uno de los sistemas para **la distribución de aplicaciones en Linux,** aparte de los repositorios **SVN** de las aplicaciones es la web *sourceforge.ne*t. es una central de desarrollos de software que controla y gestiona varios proyectos de software libre y actúa como un repositorio de códigos fuente. Fue creado en 1999. En el año 2013 ha girado hacia posiciones más mercantilistas, mediante inserciones **adware,** aunque sigue manteniendo su esencia.

Su funcionamiento se basa en la sincronización de tu proyecto con sourceForge además de gestionar las versiones de tu código, pudiendo crear tickets, tareas pendientes, dar de alta varios desarrolladores para ese proyecto, etc.. El proceso es el siguiente:

1. Se crea una cuenta en la web de sourceforge
2. Registras un nuevo proyecto (nombre, la ruta de la web del proyecto en source...)
3. Subir el código de la versión que tengas del proyecto

Según el tipo de lenguaje e IDE que utiilces la sincronización de tu versión del proyecto con el que has distribuido o es manual, o como en Java hay *addons* en Eclipse y en Netbeans que automatizan la sincronización de las sucesivas versiones que vas obteniendo.

Sistemas propietarios

En la actualidad, una gran variedad de sistemas de gestión de paquetes es usada por algunos sistemas operativos propietarios para tratar con la instalación tanto de paquetes propietarios como libres. De los que nos interesan tenemos el framework .NET de Microsoft, un ensamblado es una biblioteca de código parcialmente compilado destinado al uso en deployment, versioning y seguridad.

Gestión de paquetes incrustada en aplicaciones

Algunos sistemas de gestión de paquetes no forman parte nativa del sistema operativo, como pueden ser fink en Mac OS X o el entorno Unix-like de Cygwin (para Windows).

Algunos lenguajes de programación interpretados tienen su propio sistema de gestión de paquetes para manejar módulos del lenguaje, como pasa con los lenguajes de programación Perl (CPAN), Python (pip), PHP (PEAR) o Ruby (RubyGems). Otros programas pueden venir con su propio sistema para gestionar módulos.

Bibliografía