# TRABAJO PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

**Guía para la integración de las soluciones UCI con GNU/Linux Nova.**

Autor: José Arturo Castelo Rojas

Tutor: Ing. Edilberto Blez Deroncelé

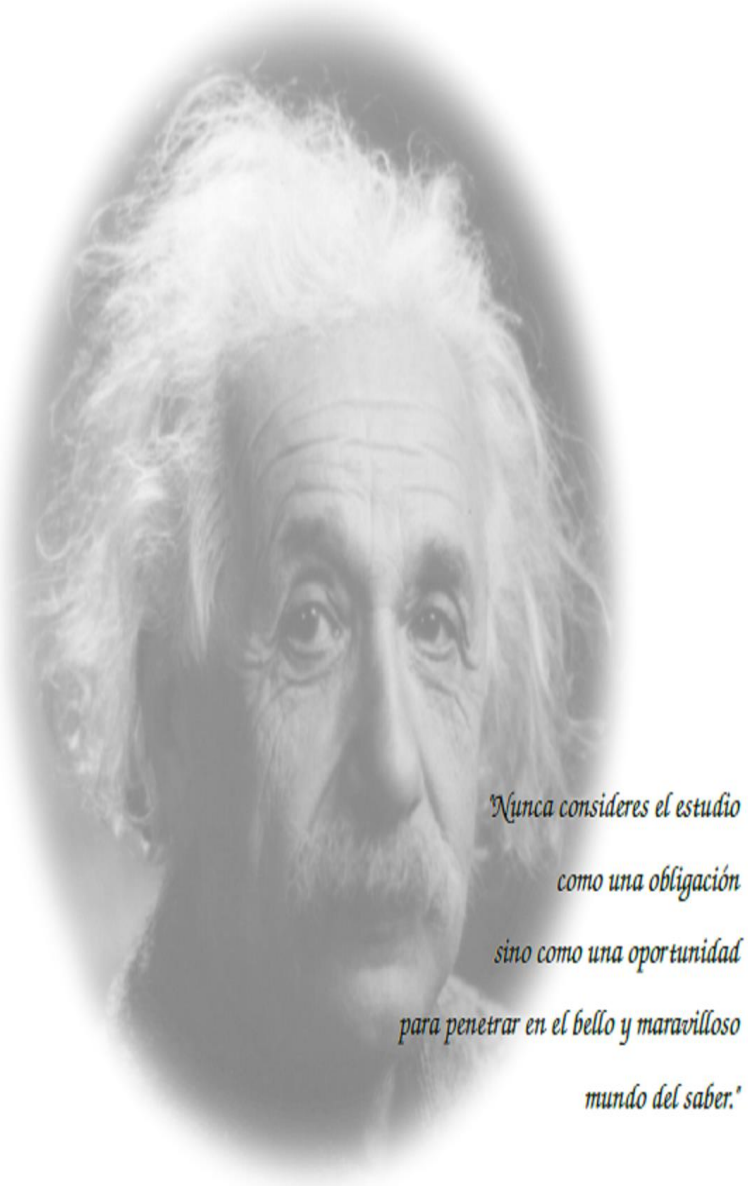
Ing. Héctor Pérez Baranda

8 DE JULIO DE 2014

UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

La Habana

# PENSAMIENTO



# DECLARACIÓN DE AUTORÍA

Declaro ser el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ___ días del mes de _____ del año _____.

Autor:

_____

José Arturo Castelo Rojas

Tutor:

_____

Ing. Edilberto Blez Deroncelé.

Tutor:

_____

Ing. Héctor Pérez Baranda.

# DATOS DE CONTACTO

Nombre y Apellidos del Tutor: Ing Edilberto Blez Deroncelé.
Institución: Universidad de las Ciencias Informáticas
E-mail: eblez@uci.cu
Ingeniero en Ciencias Informáticas, graduado en Universidad de las Ciencias Informáticas. Actualmente Especialista General del Centro de Soluciones Libres. Departamento de Sistema Operativo y Desarrollo de Tecnologías Libres

Nombre y Apellidos del Tutor: Ing Héctor Pérez Baranda.
Institución: Universidad de las Ciencias Informáticas
E-mail: hbaranda@uci.cu
Ingeniero en Ciencias Informáticas, graduado en Universidad de las Ciencias Informáticas. Actualmente Recién Graduado en Adiestramiento (NS) del Centro de Soluciones Libres. Departamento de Sistema Operativo y Desarrollo de Tecnologías Libres

# AGRADECIMIENTOS

*No es tarea fácil poner en papel a todos los que de una manera u otra estuvieron a mi lado durante este tiempo y mucho menos expresarles cuanto agradezco que así lo hicieran.*

*Me siento inmensamente dichoso y agradezco de la forma más grandiosa posible a mis padres que siempre creyeron en mí.*

*A mis tutores Edilberto y Héctor, que sin ellos nada hubiera salido, porque me apoyaron en todo momento.*

*En general a todos mis amigos, discúlpenme que no los mencione uno a uno, si lo hiciera sería infinita la lista. A todos siempre les estaré agradecido.*

# DEDICATORIA

*Este trabajo lo quiero dedicar a mi familia, en especial a mis padres por ser siempre mi apoyo y mi faro guía, a mi hermano, que siempre estuvieron pendientes de todo y a mis amigos que les toco la peor parte que fue soportarme todo este tiempo.*

*A todos muchas gracias y ojalá se sientan orgullosos de mi*

# RESUMEN

El presente trabajo de diploma titulado “Guía para la integración de las soluciones UCI con la distribución GNU/Linux Nova” pretende mejorar la integración de aplicaciones informáticas con la distribución GNU/Linux Nova a través de la descripción del proceso para lograr construir un paquete de código fuente correctamente estructurado de tal forma que permita obtener a partir del mismo, paquetes de código binario instalables en la distribución GNU/Linux Nova. El presente documento contiene los resultados obtenidos en la investigación de diferentes guías de empaquetado, herramientas para la construcción de paquetes de código fuente, así como los elementos fundamentales de dicho proceso de empaquetado. Se espera como resultado obtener una guía que permita construir paquetes de código fuente para la distribución GNU/Linux Nova de manera correcta y organizada.

**Palabras claves:** paquetes de código fuente, paquetes de código binario, empaquetado.

# ÍNDICE DE CONTENIDO

# ÍNDICE DE CONTENIDO

# ÍNDICE DE CONTENIDO

# ÍNDICE DE TABLAS

# ÍNDICE DE IMÁGENES

# INTRODUCCIÓN

## Introducción

El movimiento del *software* libre que comenzó como una pequeña comunidad de personas encaminadas a crear lo que se considera actualmente como una alternativa al uso del *software* privativo, hoy en día es un gran movimiento de desarrollo. Este crecimiento debido a las nuevas posibilidades y ventajas que traen los sistemas basados en *software* libre, han generado la existencia de varias comunidades de desarrollo, las cuales crean diversas soluciones informáticas. La creación de estas nuevas soluciones bajo los diversos sistemas operativos libres y herramientas existentes, generan problemas a la hora de estandarizar un producto y querer extenderlo hacia otras plataformas y sistemas libres. En la Universidad de las Ciencias Informáticas (UCI) existen 14 centros de desarrollo que abarcan disímiles áreas de investigación como: Visualización y Realidad Virtual, Bioinformática, Gestión de Proyectos, Investigación Desarrollo e Innovación en los Procesos Educativos, Matemática y Física Computacionales, Computación de Alto Rendimiento, Tecnologías de Bases de Datos, *Software* Libre, Redes de Telecomunicaciones y Seguridad Informática, Procesamiento Digital de Señales y Geo-información, Ingeniería y Gestión de Proyectos, Inteligencia Artificial, Web Semántica, en los cuales se utilizan diferentes lenguajes de programación, metodologías y tecnologías para el desarrollo de sus productos lo cual crea una amplia variedad de aplicaciones, pero muy diferentes entre sí. Esto provoca que al incluir aplicaciones en distribuciones sin una previa adaptación para estas, no funciones parcial o totalmente. Las aplicaciones informáticas constan de un conjunto de paquetes que durante el proceso de instalación su contenido es copiado hacia diferentes direcciones para lograr el funcionamiento final de la herramienta, sin embargo, estos paquetes muchas veces no son fáciles de instalar y es necesario modificarlos para que sea posible su uso.

Para incluir en un sistema operativo un determinado programa, se deben añadir los archivos binarios asociados a este en el sistema, o a partir de un código fuente poder realizar una instalación fácil del programa, sin embargo, esto es posible si los paquetes del mismo son compatibles con la distribución del sistema operativo que se usa, en caso de que no exista la compatibilidad se deben realizar una serie de transformaciones para lograr un portable del programa capaz de ser instalado en el sistema operativo el cual funcione correctamente.

Actualmente nuestro país está inmerso en un proceso de migración a tecnologías libres apostando por la utilización de la distribución GNU/Linux Nova, por lo que se hace necesario poder incluir en su sistema, aplicaciones que sean desarrolladas por otros centros de desarrollo de la Universidad de las Ciencias Informáticas, sin embargo,

muchas de estas entidades realizan las aplicaciones y herramientas que no cumplen con las pautas o requisitos necesarios para poder ser instalados en Nova, trayendo como consecuencia que sus miembros deben realizar un gran número de transformaciones a los paquetes para que puedan ser usados en este sistema. Estas transformaciones son costosas y necesarias para poder lograr la compatibilidad de las soluciones de las diferentes líneas de desarrollo de la UCI y la distribución Nova, provocando atrasos e insatisfacción con el cliente.

Ante el análisis de la problemática planteada y con el fin de resolverla, se plantea como **problema a resolver:** ¿Cómo contribuir a la integración de los productos realizados en los diferentes centros de la UCI, con la distribución cubana GNU/Linux Nova?

Como **objeto de estudio** de esta investigación se selecciona los procesos de construcción de paquetes de código fuente en los sistemas GNU/Linux, teniendo como **campo de acción** los procesos de construcción de paquetes de código fuente basados en Debian.

Para llevar a cabo la investigación y resolver la situación problémica existente se trazó como **objetivo general:** desarrollar una guía para la construcción de paquetes de código fuente a partir del código fuente de las aplicaciones UCI que permitan una correcta integración de las mismas con la distribución cubana GNU/Linux Nova.

Para poder tener una forma medible del avance del cumplimiento del objetivo general se definieron los siguientes **objetivos específicos:**

1. Caracterizar el proceso de empaquetado de las distribuciones basadas en el sistema de paquetes binarios de Debian (.deb).
2. Diseñar el proceso de construcción de un paquete de código fuente de una aplicación con la distribución GNU/Linux Nova.
3. Validar el aporte y efectividad del resultado de la aplicación de la guía desarrollada.

Se plantea como **idea a defender:** se puede mejorar y agilizar la integración de las aplicaciones desarrolladas en la UCI con la distribución GNU/Linux Nova si se documenta el proceso de construcción de paquetes fuentes en esta distribución.

El proceso de investigación estuvo guiado por las siguientes **tareas investigativas:**

1. Estudio de las guías de empaquetado de Debian y sus derivados, para determinar los rasgos fundamentales del proceso de empaquetado.

2. Establecimiento de una relación entre el proceso de construcción de paquetes de código fuente para aplicaciones de lenguajes compilados e interpretados.
3. Fundamentación de los pasos para la correcta construcción de un paquete de código fuente en la distribución GNU/Linux Nova.
4. Elaboración de los pasos para la correcta construcción de un paquete de código fuente en la distribución GNU/Linux Nova.
5. Utilización de la guía elaborada en un caso de estudio.
6. Evaluación del resultado de la aplicación de la guía elaborada.

Los **métodos teóricos** utilizados son:

1. **Analítico-sintético:** en el análisis y síntesis de la información consultada sobre las distintas guías de construcción de paquetes, y así elaborar ideas que se apliquen en la propuesta de solución.
2. **Modelación:** en el modelado y realización de la propuesta de solución resultante de la investigación.

El **método empírico** utilizado fue:

1. **Estudio de caso:** en el proceso de validación de la propuesta de solución dada por parte de la presente investigación.

**El presente trabajo de diploma está estructurado por 3 capítulos:**

**Capítulo 1: Fundamentación Teórica del Proceso de Empaquetado**. En el presente capítulo se describen los principales aspectos y conceptos de relevancia que serán objeto de análisis a lo largo de la investigación. Se analizarán los procesos de construcción de paquetes en Debian en sus derivados por su importancia para la investigación.

**Capítulo 2: Fundamentación de la Guía de Construcción de paquetes fuentes para la distribución GNU/Linux Nova**. En este capítulo se presenta la guía para la construcción de paquetes de código fuente que permitirá eliminar los problemas encontrados en la instalación de paquetes en la distribución GNU/Linux Nova. Se exponen además, criterios sobre cómo construir los principales componentes de los paquetes en otras distribuciones para lograr su compatibilidad con GNU/Linux Nova.

**Capítulo 3: Validación de la Guía de Construcción de paquetes fuentes para la distribución GNU/Linux Nova**. Una manera de comprobar si lo que se ha hecho está correcto lo constituye la validación. Este proceso se realiza con el objetivo de verificar y demostrar la confiabilidad de una propuesta. En el presente capítulo y como parte de la

validación y aceptación de la propuesta de solución se realizará una construcción de un paquete fuente utilizando la misma y se hará una descripción de los resultados obtenidos luego de la ejecución.

# 1. Capítulo 1

## 1.1. Introducción

En el presente capítulo se abordarán los principales conceptos relacionados con el objeto de estudio. Además, se realizará un estudio y análisis de las principales guías homólogas a la que se desea realizar, para extraer sus principales elementos así como los puntos que puedan representar parte de la solución a desarrollar. En esta sección de la investigación se analizarán diferentes tecnologías y técnicas utilizadas en el mundo para realizar los procedimientos descritos en las guías estudiadas.

## 1.2. Conceptos Fundamentales

### 1.2.1. Paquete binario

Es un paquete en el cual los componentes que contiene están listos para instalarse en el sistema operativo. En el caso de las aplicaciones desarrolladas en lenguajes que requieren ser compilados, el paquete contiene los archivos en algún formato binario compatible con el cargador de GNU/Linux, generalmente en formato ELF, además de otros archivos auxiliares como pudieran ser archivos de imágenes o de configuración. En un paquete binario los archivos a instalarse están organizados siguiendo la misma estructura con la que serán instalados en el sistema de archivos del sistema operativo. En el caso de las distribuciones basadas en Debian, como la que da origen al presente trabajo, utilizan archivos que tienen como extensión *.deb* o *.udeb*, y que cumplen con el estándar definido por los desarrolladores del mencionado proyecto (1). Otras distribuciones usan paquetes con extensión *.rpm*.

### 1.2.2. Paquete fuente:

Se conoce como paquete fuente o paquete de código fuente a un paquete que contiene los elementos necesarios para construir un paquete binario. En el caso de las distribuciones basadas en Debian GNU/Linux, los paquetes fuentes se conforman por un grupo de archivos: un comprimido que contiene el código fuente original desarrollado por el autor del programa o aplicación; otro contiene las modificaciones o parches del mantenedor. Este segundo paquete contiene los datos y *scripts* con las reglas requeridas por las herramientas de empaquetado, construcción e instalación de paquetes. Además, se provee un archivo de descripción que contiene información sobre el conjunto de paquetes fuentes, así como firmas de verificación de integridad de los mismos, datos del mantenedor, distribución de GNU/Linux para el cual está empaquetado, entre otros.

### 1.2.3. Dependencias de paquetes:

Las dependencias de un paquete, son aquellos paquetes binarios de los que este depende para poder funcionar en determinada plataforma de *hardware* y *software*.

Existen dos tipos de dependencias, las de tiempo de construcción (*build dependency*) y las de tiempo de ejecución (*runtime dependency*). Las dependencias de construcción son aquellos paquetes binarios que se requiere que estén instalados para poder llevar a cabo el proceso de construcción de un paquete binario a partir de un paquete fuente. Por ejemplo, el paquete que contiene al compilador un lenguaje con que esté desarrollada determinada aplicación sería una dependencia de construcción. Lo mismo pasaría con los paquetes que contienen, por ejemplo, a los archivos de encabezado (.h) del *kernel* Linux. Por otro lado, las dependencias de tiempo de ejecución son aquellos paquetes que se requieren que estén instalados en el sistema operativo para que determinada aplicación, una vez instalada, pueda ejecutarse correctamente. Por ejemplo, los paquetes que instalan bibliotecas dinámicas – objetos compartidos o *shared objects* (.so) – son un ejemplo de los mismos, pero también pudiera ser un paquete que instale un tema de iconos que necesite determinada aplicación para cargar en su interfaz de usuario (1). Un sistema de construcción de aplicaciones o componentes de *software* solo trabaja con dependencias de construcción o compilación.

### 1.2.4. Repositorio de paquetes:

Es una forma de organizar un conjunto de paquetes a través de una estructura de archivos que facilite su localización y recuperación. Por lo general un repositorio está organizado en una estructura de índice alfabético, donde los paquetes cuyos nombres comienzan por una misma letra se encuentran en un mismo directorio, a la vez que existen archivos de índices que especifican en qué directorio se encuentra cada paquete así como los datos de los mismos. Los archivos fundamentales que identifican a un repositorio por lo general se encuentran firmados usando algoritmos de cifrado como PGP. Un repositorio puede estar alojado de forma local o remota, en este último caso por lo general se puede acceder a través de los protocolos HTTP o FTP (1).

### 1.2.5. Construcción de un paquete:

Proceso mediante el cual un paquete fuente es procesado por un conjunto de herramientas que, siguiendo un conjunto de algoritmos definidos por el conjunto de reglas y datos establecidos por el desarrollador que empaquetó el código fuente, generan un paquete binario (1). La construcción de un paquete no siempre lleva implícito un proceso de compilación, pues a veces un paquete fuente solo contiene datos.

## 1.3. Estudio de las guías de empaquetado de Debian y sus derivados.

El estudio de la guía de empaquetado de Debian, será el documento principal a analizar en este epígrafe dado que la documentación ofrecida por Ubuntu sobre el tema, así como otras

guías que existen realizadas por diferentes autores, responden en sus mayoría a la de Debian: “Guía del nuevo desarrollador de Debian”.

### 1.3.1. Debianizando los paquetes o aplicaciones.

Lo primero que define la guía de empaquetado de Debian es debianizar los paquetes o la aplicación con la que se va a trabajar, el proceso de debianizar consiste en tomar el código fuente recibido de los desarrolladores y llevarlo a uno equivalente que contiene archivos de información que ayudan a la construcción de los paquetes binarios utilizando herramientas que se explicarán posteriormente. Este proceso consiste en crear un subdirectorio personal con el nombre que identificará a la aplicación, en el cual se almacenará todo el código fuente de la misma.

El directorio que contendrá el código fuente se debe nombrar convenientemente y siguiendo las políticas de Debian, las cuales indican que el nombre debe de ser nombre_paquete-versión, esta referencia se utiliza en el archivo debian/changelog. Debe contener al menos dos caracteres, empezar con un carácter alfanumérico y no coincidir con alguno de los paquetes ya existentes. Es buena idea limitar su longitud a un máximo de 30 caracteres (2).

Si el código fuente original utiliza palabras genéricas tales como prueba-privada por nombre, es buena idea cambiarlo para no «contaminar» el espacio de nombres y para identificar su contenido en forma explícita. Si el código fuente no utiliza el sistema habitual para codificar la versión, como 2.30.32 sino que utiliza algún tipo de fecha como 09Oct23, una cadena de nombre en clave al azar o un valor «*hash*» VCS como parte de la versión, se recomienda eliminarlo de la versión del código fuente. Esta información (eliminada) puede ser registrada en el archivo debian/changelog.

Si la versión a utilizar se quiere que sea una fecha, se debe utilizar el formato AAAAMMDD (por ejemplo 20110429). Esto asegura que dpkg considerará correctamente las versiones posteriores como actualizaciones. Si se quiere garantizar una transición fluida al formato de versión más habitual, como 0.1 en el futuro, se propone usar 0~AAMMDD como 0~110429 para la versión principal (2).

### 1.3.2. Configurar dh_make

Debian establece como buena práctica configurar el dh_make antes de utilizarlo para generar el paquete fuente, esto permite incluir el nombre del mantenedor y el correo electrónico del mismo en las plantillas que serán creadas luego. Primero se debe configurar las variables de entorno «*shell*» $DEBEMAIL y $DEBFULLNAME que son

utilizadas por varias herramientas de mantenimiento de Debian para obtener el nombre y correo electrónico como se indica a continuación (2).

```
$ cat >>~/.bashrc <<EOF

DEBEMAIL="usuario@dominio"

DEBFULLNAME="Nombre del mantenedor"

export DEBEMAIL DEBFULLNAME

EOF

$ . ~/.bashrc
```

### 1.3.3. Utilizando dh_make

Debian tiene definido para sus paquetes una clasificación teniendo en cuenta si fueron desarrollados específicamente para Debian o son paquetes creados por otras distribuciones derivadas o que inicialmente no fueron paquetes destinados para dicha distribución. Para esta situación Debian resuelve que los paquetes que fueron desarrollados originalmente para su funcionamiento en él, son nombrados paquetes nativos y los que no cumplen esta característica son paquetes no nativos (2).

#### 1.3.3.1. Paquete no nativo Debian inicial

Para paquetes que son considerados nativos de Debian, se define una forma específica para utilizar *dh_make*, con ello se garantiza que se generen los archivos correctos. Lo primero que plantea la guía es que se debe posicionar dentro de la carpeta que contiene el código y ejecutar el comando *dh_make* con cierto parámetro que indique cuál es el comprimido asociado a esta carpeta de código fuente, por lo que sería de la siguiente forma (2):

```
$ cd nombre_paquete-version

$ dh_make -f ../nombre_paquete-version.tar.gz
```

Luego de realizar estos pasos se posee fuera del directorio que se está trabajando lo siguiente:

```
nombre_paquete-version /

nombre_paquete-version.tar.gz

nombre_paquete_version.orig.tar.gz
```

Si se compara con el directorio que se tenía anteriormente se nota que hay dos cambios claves:

- ✓ Hay un .orig.tar.gz.
- ✓ El nombre del paquete y la versión de este .orig.tar.gz están separados por «_».

Luego de ejecutada la orden, se han generado varios archivos de plantilla en el directorio debian contenido dentro de la carpeta donde se encuentra el código fuente. Si accidentalmente se ha eliminado alguna de las plantillas mientras se trabajaba en ellas, se pueden volver a generar ejecutando *dh_make* con la opción *--addmissing* desde el directorio con las fuentes del paquete Debian (2).

#### 1.3.3.2. Paquete nativo Debian inicial

Los paquetes que Debian define como nativos poseen un proceder específico para utilizar *dh_make* en ellos, lo cual influirá en la obtención de los archivos necesarios para construir correctamente el paquete fuente. Si un paquete será mantenido solamente para Debian, posiblemente para uso local, es recomendable mantenerlo de forma nativa. Si se tienen fuentes en ~/mi_paquete-1.0, se puede generar un primer paquete nativo de Debian ejecutando la orden *dh_make* como sigue (2).

```
$ cd ~/mi_paquete-1.0

$ dh_make --native
```

Entonces el directorio debian y su contenido son generados. De esta manera no se genera un nuevo archivo .orig.tar.gz dado que este es un paquete Debian nativo. Pero esa es la única diferencia. El resto de las actividades de empaquetado son prácticamente las mismas (2).

### 1.3.4. Archivos que se modifican en el directorio creado.

Al término de los pasos anteriores ya se tiene creado en el directorio de trabajo los archivos necesarios para construir un paquete fuente pero estos archivos no están completos y necesitan algunas modificaciones y especificaciones. Para ellos Debian define todo un proceso explicando la función de cada archivo y los elementos que contienen cada uno, a continuación se muestran algunos de estos archivos.

#### 1.3.4.1. Archivo control

El archivo control contiene la información que el administrador de paquetes (como *dpkg*, *apt, aptitude, Synaptic y Adept*) usa: dependencias necesarias, información del mantenedor, y otros elementos (3).

#### 1.3.4.2. Archivo changelog

El archivo *changelog*, como su nombre indica, es una lista de cambios que se hacen en cada versión. Tiene un formato específico que proporciona una descripción del conjunto de modificaciones realizadas al paquete de código fuente. Cada entrada contiene el nombre del paquete, la versión, la distribución, descripción de los cambios, autor de las modificaciones y cuándo se hizo. Si se tiene una llave GPG, se debe asegurar de utilizar el mismo nombre y cuenta de correo que en el *changelog* respecto al de la llave (3). Este archivo es muy importante, porque en él radica el aspecto que hace posible identificar para qué distribución es nativo el paquete. Para ello se muestra un ejemplo sencillo de esta diferencia:

El nombre de un paquete está compuesto por 3 partes fundamentales:

Nombre del paquete-versión-revisión

Se ejemplifica usando un paquete con nombre hello:

**Forma de plantear estos datos en el archivo changelog de Debian** (2)**:**

Hello-1.0-1Ubuntu3, esto significa que:

El nombre del paquete es: Hello

Versión: 1.0

Revisión: este paquete ha sido modificado una vez por Debian y 3 veces por Ubuntu.

**Forma de plantear estos datos en el archivo changelog para paquetes nativos de Ubuntu** (4)**:**

Hello-1.0-0Ubuntu1, esto significa que:

El nombre del paquete es: Hello

Versión: 1.0

Revisión: este paquete no ha sido modificado por Debian y si, al menos una vez, por Ubuntu, lo que significa que es nativo de este último.

#### 1.3.4.3. Archivo copyright

El archivo *copyright* proporciona la información de derechos de autor. Generalmente, la información de *copyright* se encuentra en el archivo *COPYING* en el directorio principal del código fuente. En este archivo se debe incluir información como el nombre del empaquetador y del creador del paquete, la *URL* de donde estará disponible el código fuente, la licencia con el año del *Copyright* (3).

#### 1.3.4.4. Archivo rules

El último archivo que se necesita es el *rules*. Este, prácticamente, hace todo el trabajo para crear el paquete deb. Es un *Makefile* con instrucciones para compilar e instalar la aplicación, luego crear el archivo .deb a partir de los archivos instalados. También

tiene instrucciones para revertir la compilación e instalación eliminando todos los archivos originales y así poder volver a disponer solo del código fuente (2).

#### 1.3.4.4.1. Métodos de compilación simple

Luego del debianizado se procede a la compilación de estos archivos, los métodos de compilación simple permiten mediante archivos que contiene el directorio creado realizar instalaciones y configuraciones en los paquetes. Los programas sencillos incluyen un fichero *Makefile* y pueden (generalmente) compilarse con «*make*». Algunos de ellos soportan *make check*, esta orden ejecuta las comprobaciones automáticas que estén definidas. Generalmente se instalarán en sus directorios de destino ejecutando *make install*. En este paso el código está listo para ser compilado quedando disponibles opciones del comando *make* como *clean* y *distclean* para limpiar el árbol de construcción. En ocasiones se encuentra disponible el objetivo *uninstall* que se puede utilizar para borrar todos los archivos instalados anteriormente por el objetivo *install* (2).

#### 1.3.4.4.2. Métodos de compilación portables populares

Existen otros métodos de compilación conocidos como portables refiriéndose a los que utilizan una serie de herramientas para generar el archivo *makefile* que luego se utilizará para la compilación de estos paquetes. Buena parte de los programas libres están escritos en lenguaje C y C++. Muchos utilizan las «*Autotools*» y «*CMake*» para compilar en diferentes plataformas. Estas herramientas se utilizan para generar un archivo *Makefile* y otros archivos necesarios para la compilación. Así, muchos programas se compilan ejecutando «*make; make install*». Las *Autotools* son el sistema de compilación GNU e incluyen *Autoconf*, *Automake*, *Libtool* y *gettext*. Se puede comprobar generalmente si el programa utiliza las *autotools* por la presencia de los archivos configure.ac, *Makefile.am*, y *Makefile.in*. El primer paso en el uso de «*Autotools*» es la ejecución por parte del autor de la orden autoreconf -i -f la cual genera, a partir de los archivos fuentes los archivos que utilizará la orden «configure». Se pueden realizar cambios en el archivo *Makefile*, por ejemplo cambiando el directorio de instalación predeterminado usando las opciones de la orden ejecutando: /configure --prefix=/usr. Aunque no es necesario, la actualización del archivo configure y de otros archivos con la orden autoreconf -i -f es la mejor manera para lograr una correcta compatibilidad del código fuente. *CMake* es un sistema de compilación alternativo. La presencia del archivo CMakeLists.txt indica que se utiliza esta opción para compilar el programa (2)

#### 1.3.4.5. Archivo README.Debian

Cualquier detalle extra o discrepancias entre el programa original y su versión debianizada debería documentarse aquí (2).

#### 1.3.4.6. Archivo compat.

El archivo *compat* define el nivel de compatibilidad de *debhelper*. Actualmente establece compatibilidad a la versión 7 de *debhelper* (2).

#### 1.3.4.7. Archivo conffiles

Un comportamiento no deseado es que luego de gastar tiempo y esfuerzo adaptando un programa (como usuario), ocurre una actualización que destruye dichos cambios. Debian resuelve este problema marcando los ficheros de configuración. Así cuando se actualiza los usuarios pueden elegir mantener la configuración anterior o renovarla. Desde la versión 3 de *debhelper*, *dh_installdeb* considera automáticamente a todos los archivos ubicados en el directorio /etc como «*conffiles*» (archivos de configuración gestionados por el sistema de paquetes). Así, si todos los «*conffiles*» están en este directorio no es necesario incluirlos en este archivo. Para la mayoría de paquetes, la única ubicación de los «*conffiles*» es /etc por lo que no es necesario generar este archivo. En el caso de que el programa utilice ficheros de configuración pero también los reescriba él mismo, es mejor no marcarlos como «*conffiles*». *La herramienta dpkg* informará a los usuarios que verifiquen los cambios de estos ficheros cada vez que lo actualicen. Si el programa que se está empaquetando requiere que cada usuario modifique los archivos de configuración del directorio /etc, hay dos formas para no marcarlos como archivos «*conffiles*» y que no sean manipulados por *dpkg*, donde la más usada es construir un enlace simbólico de los archivos ubicados en /etc que apunten a archivos ubicados en el directorio /var generados por guiones del desarrollador («*maintainer scripts*») (2).

#### 1.3.4.8. Archivos tareas.cron.*

Si el paquete requiere tareas periódicas para funcionar adecuadamente, se puede usar este fichero como patrón. Se puede establecer la realización de tareas que se ejecuten cada hora, día, semana, mes, o en cualquier otro período de tiempo (2).

#### 1.3.4.9. Archivo dirs

Este fichero especifica los directorios que se necesitan pero que por alguna razón no se crean en un proceso de instalación (*make install* DESTDIR=... invocado por *dh_auto_install*). Generalmente es debido a un problema con el archivo *Makefile*. Es

recomendable ejecutar en primer lugar la instalación y solo hacer uso de este archivo si se produce algún problema. No debe ponerse la barra inicial en los nombres de los directorios listados en el archivo dirs (2).

#### 1.3.4.10. Archivo nombre_del_paquete.doc-base

Si el paquete tiene documentación además de las páginas de manual y de información, puedes utilizar el archivo doc-base para registrarla de modo que el usuario pueda encontrar esta documentación suplementaria con *dhelp*, *dwww* o *doccentral*. La documentación incluirá archivos HTML, PS y PDF ubicados en /usr/share/doc/nombre_del_paquete/ (2).

#### 1.3.4.11. Archivos nombre_del_paquete.init y nombre_del_paquete.default

El archivo nombre_del_paquete.init se instala como un guión en /etc/init.d/nombre_del_paquete. Se trata de una plantilla genérica construida por la orden *dh_make* como *init.d.ex*. Se debería renombrar y hacerle muchos cambios para que cumpla con las cabeceras del estándar base de Linux. La orden *dh_installinit* lo instalará en el directorio temporal. El archivo nombre_del_paquete.default se instalará en el directorio /etc/default/nombre_del_paquete. Este archivo establece los valores predeterminados que utiliza el guión *init*. En la mayoría de los casos, el archivo nombre_del_paquete.default, se utiliza para desactivar la ejecución del demonio, estableciendo algunos indicadores predeterminados o tiempos de espera. Las características configurables establecidas en el guión *init* deben incluirse en este archivo predeterminado, y no en el propio guión.

Si el programa original incluye un archivo guión *init*, se puede hacer uso de él o bien descartarlo. Si se opta por no hacer uso del guión *init* original, se debería construir uno nuevo en debian/nombre_del_paquete.init. En cualquier caso se debe construir los enlaces simbólicos rc* aunque el guión *init* original parezca correcto y se instale en el lugar adecuado. Para ello, deberás reescribir el objetivo dh_installinit en el archivo rules (2).

#### 1.3.4.12. Archivo install

Si hay archivos que deben ser instalados por el paquete pero no lo hace *make install,* se puede listar los archivos y sus destinos en el archivo *install*. Se encargará de la instalación la orden *dh_install*. Se debe asegurar que no hay un sistema más específico para hacer esta instalación. Por ejemplo, para la documentación debe utilizar el archivo docs, en lugar de este archivo. El archivo *install* tendrá una línea para cada uno de los archivos a instalar, con el nombre del archivo seguido de un

espacio y a continuación el directorio de instalación. La dirección de los nombres de los archivos a copiar es relativa a la carpeta donde se encuentra todo el código fuente, así como la dirección a donde se copiarán, que es relativo a la carpeta temporal que se crea para la generación del paquete binario. En caso de que se esté trabajando en paquetes grandes que separan como resultado de la compilación múltiples paquetes binarios, se debe especificar un fichero *.install* por cada uno de los binarios a generar, quedando nombre_del_paquete1.install, nombre_del_paquete2.install, etc. La orden dh_install retrocederá al directorio debian/tmp para buscar los archivos si no los encuentra en el directorio actual de construcción (2).

#### 1.3.4.13. Archivo source/format

El archivo debian/source/format, solo contendrá una línea indicando el formato deseado para el empaquetado (3).

El estudio de otras guía existentes para este procedimiento arrojó que los procesos descritos son exactamente los mismos pasos que plantea Debian, estas guías solo realizan un resumen de este proceder por lo que carecen de información importante para la investigación.

### 1.3.5. Generando el paquete de código fuente

Luego de haber realizado todo el procedimiento anterior ya se tienen todos los archivos necesarios para poder crear el paquete de código fuente por lo que ahora se tiene que realizar esta tarea. Ahora solo se tiene que construir dicho paquete para ello se utiliza el siguiente comando: dpkg-source –b /dirección_código (2).

### 1.3.6. Resultados del análisis de las guías de empaquetado.

Luego de estudiar y analizar los procedimientos brindados por diferentes guías de empaquetado en formato .deb y de plasmar los pasos generales en los epígrafes anteriores se posee una base de conocimientos sobre las generalidades del tema. El estudio de las guías mencionadas anteriormente permitió adquirir los elementos importantes y puntos en los que se debe realizar énfasis para el desarrollo de la investigación. Las presentes guías aportaron a la investigación los procesos fundamentales del empaquetado, luego de analizarlas se notó que no importa la guía o su nivel de claridad todos los procedimientos descritos en cada una de ellas convergen en 3 puntos fundamentales del proceso, a continuación se plantean estos puntos en los que coinciden las guías:

1. Debianizar el paquete o aplicación a empaquetar.
2. Configurar los ficheros que formarán parte del fuente: archivo *changelog*, archivo *control*, archivo *copyright*, el archivo *rules* y otros.
3. Compresión para generar el paquete de código fuente.

### 1.4. Conclusiones Parciales

La realización de este capítulo ha permitido obtener del estudio de las guías de empaquetado, los principales elementos y pasos definidos para realizar este procedimiento, detectándose puntos en común entre dichas guías. Se analizaron las principales diferencias existentes entre estas guías para definir cómo identificar los paquetes realizados por diferentes distribuciones. Se plantearon los principales conceptos asociados al objeto de estudio de la investigación facilitando la comprensión de términos técnicos que se utilizan en la presente investigación.

# 2. Capítulo 2

## 2.1. Introducción

El presente capítulo fundamentará las decisiones tomadas durante el proceso de construcción de un paquete de código fuente para la distribución cubana GNU/Linux Nova descrito en la guía propuesta por la presente investigación (Véase Anexo 1). Durante el desarrollo de este capítulo se plantearán las principales herramientas utilizadas y por qué su uso, además las principales configuraciones y variaciones que se le realizan a los paquetes de código fuente en la distribución cubana GNU/Linux Nova.

## 2.2. Debianizar

Esta es la primera actividad definida a hacer por la guía propuesta, primeramente se analiza que el interés fundamental de la investigación, es lograr la integración de las aplicaciones UCI con la distribución cubana GNU/Linux Nova. En este caso la integración de las aplicaciones con el sistema operativo significa que sea posible instalar estas aplicaciones en el mismo además de que dicha aplicación funcione correctamente. Todos los sistemas operativos poseen una manera de incorporar aplicaciones en él, mayormente es instalándolas. La distribución cubana GNU/Linux Nova es derivada de Ubuntu el cual se deriva de Debian, esto hace que al ser tanto GNU/Linux Nova como Ubuntu derivados de Debian utilicen el mismo formato para los paquetes instalables o paquetes binarios, para estos sistemas el formato de los paquetes instalables es .deb que no es más que la abreviatura de Debian, de ahí que el proceso se nombre debianizar (Véase Anexo 1).

### 2.2.1. Aplicaciones a instalar

En este paso se procede a instalar diversas aplicaciones necesarias para poder realizar el procedimiento descrito en la guía propuesta por esta investigación. La instalación de estas aplicaciones es de vital importancia para poder realizar este proceso muchas de ellas generan los ficheros que serán usados en determinados pasos y otras brindan funcionalidades que permiten automatizar casi en su totalidad un procedimiento lo cual evita una carga de trabajo para el mantenedor así como disminuir la probabilidad de ocurrencia de errores. En la guía propuesta se seleccionaron como aplicaciones y herramientas a instalar las siguientes: *build-essential*, *debhelper*, *dh_make*, *dpkg-dev* y *devscripts* estas resultan necesarias para poder realizar el procedimiento propuesto dado que brindan una mayor automatización al proceso y permiten el uso de funciones ya definidas.

- ✓ ***build-essential:*** Este es el paquete imprescindible para el desarrollo del proceso a realizar. Al instalarlo, también se instalarán otros paquetes requeridos, consiguiendo una instalación básica para la construcción de paquetes (5).

- ✓ ***debhelper, dh_make:*** Son necesarias para construir el esqueleto del paquete ejemplo, y se usarán algunas de las herramientas de *debhelper* para construir los paquetes. Aunque no son imprescindibles para la construcción de paquetes, se recomienda su uso a todos los desarrolladores. Hacen el proceso mucho más fácil al principio, y más fácil de controlar el resultado futuro (5).
- ✓ ***dpkg-dev:*** Este paquete contiene las herramientas necesarias para desempaquetar, construir y enviar paquetes fuente de Debian (5).
- ✓ ***devscripts:*** Este paquete contiene algunos guiones útiles para los desarrolladores, es importante mirar los paquetes recomendados y sugeridos por este paquete (5).

### 2.2.2. Ejecutar el script dh_make

En la guía propuesta se utiliza en el proceso de debianizado el script *dh_make* por sus ventajas y la amplia automatización que aporta al proceso. El script *dh_make* crea una carpeta debian y una serie de ficheros que son luego utilizados por el mantenedor para configurar y construir un paquete de código fuente (Véase Anexo 1).

**dh_make**

*dh_make* es una herramienta para convertir un paquete de código fuente regular en uno formateado de acuerdo con los requisitos de la normativa de Debian. *dh_make* debe ser invocado dentro de un directorio que contiene el código fuente, que debe ser nombrado <nombrepaquete> - <*versión*>. El <nombrepaquete> debe ser todo en minúsculas, dígitos y guiones. Si el nombre del directorio no cumple con este esquema, debe cambiar el nombre antes de usar dh_make. El uso de *dh_make* como se propone en la propuesta de solución permitirá además utilizando una de sus opciones crear una carpeta .orig.tar.gz la cual será una copia exacta de código fuente original de la aplicación teniendo así una copia fiel de dicho código pudiendo ser utilizada en caso necesario para realizar otras actividades (6).

**Clases de paquetes**

Luego de ejecutado el script *dh_make* este pide al usuario que elija el tipo de paquete que desea crear brindándole al mismo diferentes opciones como son: Binario simple ( s ), Arch- Independiente ( i ), Binaria múltiple ( m), Biblioteca ( l ), Módulo Kernel ( k ) y cdbs ( b ) de estas opciones para la investigación resultan de importancia las siguientes (6):

**Binario simple (s)**

El paquete generará un único binario .deb. Es el caso estándar, por lo que si no se tiene claro qué opción usar lo recomendado es seleccionar esta.

**Binaria múltiple (m)**

El paquete generará múltiples binarios .deb de un paquete fuente. Se elige esta opción para el caso de paquetes más grandes que necesitan ser divididos.

La selección de esta opción correctamente es de suma importancia para el proceso como se especifica en la descripción de cada una de las opciones de acuerdo a la que se elija será la cantidad de binarios que se podrán generar posteriormente por lo que una equivocación podría provocar que luego ocurran problemas para generar los paquetes binarios.

**Acciones realizadas**

Los archivos son personalizados con el nombre de paquete y la versión extraída del nombre del directorio. El nombre de usuario es consultado en el entorno variable capaz $ DEBFULLNAME si este existe. La variable $LOGNAME se utiliza para encontrar un nombre en el archivo / etc / passwd, y por medio de NIS, YP y LDAP. Si la variable de entorno $ EMAIL o $ DEBEMAIL se establece, o el correo (- email) entonces esa dirección de correo electrónico se introduce como correo electrónico del mantenedor en cada acción que sea necesario (6):

DEBEMAIL exportación = " jsmith@debian.org "

dh_make también generará archivos de ejemplo que también se han personalizado para el paquete generado. En caso de que no sean necesarios los archivos *.EX se puede optar por eliminar todos los archivos con dicha extensión. También se puede eliminar el archivo *README*. Debian al cual no se le realizará ningún cambio. La propuesta de solución planteada por esta investigación propone eliminar estos archivos *.EX o *.ex y en caso de ser necesarios en el futuro se crearán manualmente por el mantenedor.

La herramienta *dh_make* permite ser utilizada respondiendo a una serie de opciones que realizan independientemente diferentes funciones, ejemplos de las mismas son: -c, - <*license*> *copyright*, -e, - <dirección> *email*, -n, - nativa, -f, - file <*file*>, -l, - biblioteca, -k, - *kmod*, -b, - *cdbs* y – *kpatch*. Estas opciones le permiten al usuario generar diferentes resultados del uso de *dh_make*, para los fines de la investigación presente se recomienda el uso de la opción (6):

-e, - <dirección> email. Se sustituye <dirección> como la dirección de correo electrónico del Mantenedor: campo de archivo debian / control.

Además de especificar el correo electrónico como buena práctica esta investigación recomienda además usar a continuación del correo electrónico del mantenedor la opción *–createorig* la cual proporcionará una copia exacta del código original del paquete o aplicación que se está empaquetando en un comprimido .orig.tar.gz.

La capacidad que posee para lograr automatizar la mayor parte del proceso de empaquetado, además de ser bastante sencilla para usuarios avanzados e inexpertos y la facilidad que brinda para realizar este proceso debido a las variadas opciones que posee hace que el uso de *dh_make* es actualmente la opción más utilizada entre mantenedores y usuarios a nivel internacional y ser reconocido su uso como una buena práctica por parte de los desarrolladores de Debian.

#### 2.2.2.1. Formato del paquete de código fuente.

Durante la construcción de los paquetes de código fuente existen elementos a seleccionar que resultan relevante, por lo que su selección adecuada influye en el resultado final, de ahí que tener claro la selección que se realiza será importante durante todo el proceso. De los aspectos fundamentales a tener en cuenta en este proceso se encuentra la selección del formato del paquete de código fuente el cual definirá los archivos o carpetas que se obtendrán al finalizar el proceso. A continuación se muestran los diferentes formatos existentes y los archivos obtenidos finalizado el proceso (3):

**1.0 o 3.0 (native):** package_version.tar.gz

**1.0 (non-native):** pkg_ver.orig.tar.gz: Fuente original de *software*
pkg_debver.diff.gz: Parche para añadir cambios específicos de Debian

**3.0 (quilt):** pkg_ver.orig.tar.gz: Fuente original de *software*
pkg_debver.debian.tar.gz: Archivo tar con los cambios de Debian

**Format: 3.0 (quilt)**

Un paquete fuente con este formato contiene al menos un archivo tar original (.orig.tar.*ext*, siendo *ext* gz, bz2, lzma o xz) y un archivo tar de Debian (.debian.tar.*ext*). También contiene archivos tar originales adicionales (.orig−*componente*.tar.*ext*). Este último archivo el debian.tar.gz es una comprimido que tendrá en su contenido todos los parches que se posea el paquete lo cual permitirá posteriormente incluirlos o no en el paquete final. El uso de este formato trae ventajas lo cual hace que sea de todos los formatos anteriormente mencionados el más usado, algunas de las ventajas son (7):

- ✓ Parches separados y bien documentados.
- ✓ El archivo *rules* se torna más sencillo.
- ✓ Múltiples *tarballs*.
- ✓ Soporta bzip2, lzma y gzip.
- ✓ Soporta la inclusión de archivos binarios.

Estas ventajas que brinda el uso del formato 3.0 (quilt) hace que sea recomendado como buena práctica su uso por parte de los mantenedores de Debian y que sea el formato de paquete de código fuente recomendado para usarse en la presente investigación.

**Manualmente**

Ciertamente el usuario podría realizar los procedimientos que hace *dh_make* de manera manual, pero esto implicaría crear una carpeta o directorio de nombre debian y crear cada uno de los ficheros necesarios manualmente. Esta forma de realizar este procedimiento dista de ser eficiente, factible y lógica, la carga que recaería sobre el mantenedor sería inmensa, se aumentaría la probabilidad de cometer errores o se crearía un producto que no cumple con la calidad requerida. Por lo que realizar esta actividad manualmente no es considerada una solución factible.

## 2.3. Modificación de los archivos necesarios para la construcción del paquete de código fuente.

A estos paquetes de manera general se le realizan las mismas modificaciones que se explican en otras guías de empaquetado, por ello esta fundamentación estará centrada en las modificaciones que se realizan y son diferentes al resto en la distribución GNU/Linux Nova, garantizando el funcionamiento del paquete o aplicación en dicha distribución.

### 2.3.1. Archivo changelog

Este es un fichero requerido, con un formato especial, este es el formato utilizado por *dpkg* y otros programas para obtener el número de versión, revisión, distribución y urgencia del paquete. Es también importante, porque permite tener bien documentados todos los cambios que se hayan hecho. Esto ayudará a las personas que se descarguen el paquete, notar si hay temas pendientes en el mismo que deberían conocer de forma inmediata. Para el caso de la distribución GNU/Linux Nova en este fichero se recomienda hacer énfasis en la estrategia para realizar los cambios de la sección de revisión la cual contiene la información de que cantidad de modificaciones se le han realizado al paquete, en qué distribución se realizaron estas modificaciones y para qué distribución fue modificado, estos datos se muestran de la siguiente manera (Véase Anexo 1):

nombre _paquete-versión-revisión

Como ejemplo se propone el paquete ejemplo que es un paquete realizado por GNU/Linux Nova por lo que en su archivo *changelog* se muestra la siguiente información: ejemplo (1.0-0nova1)2013, es importante tener en cuenta esta información, dado que representa que el paquete ejemplo está en su versión 1.0, que Debian no le ha realizado ninguna modificación y que GNU/Linux Nova le realizó una modificación, lo cual significa que es un paquete nativo de GNU/Linux Nova. Por último y además muy importante está el valor de 2013 este valor significa para que versión de la distribución será empaquetado, este campo es importante dado que si se quiere incluir esta aplicación o paquete a un repositorio este valor influirá en cuál repositorio se incluya si en uno u otro.

### 2.3.2. Archivo control

El archivo control contiene la información que el administrador de paquetes (como *dpkg*, *apt*, *aptitude*, *Synaptic* y *Adept*) usa para el proceso de instalación y configuración del paquete en el sistema: dependencias necesarias, información del mantenedor, etc. Para este archivo primeramente verificar que el nombre del paquete esté correcto, luego modificar la *section* según las opciones que se muestran en la tabla (Véase Tabla 1). Los datos del mantenedor se toman de la configuración del *dh_make* realizada en el proceso. A continuación es importante el campo *Homepage,* este campo se modifica para introducir la dirección donde se encuentra disponible el código original realizado por el autor para el caso de GNU/Linux Nova esta url es: www.nova.uci.cu. Lo próximo que se debe tener en cuenta es la selección de la arquitectura, en este campo generalmente se le introducen dos valores *any* o *all* en dependencia de para qué arquitectura se desea construir el paquete (Véase Anexo 1).

### 2.3.3. Archivo copyright

El archivo *copyright* proporciona la información de derechos de autor. Generalmente, la información del *copyright* se encuentra en el archivo *COPYING* en el directorio principal del código fuente. A este archivo se le debe incluir información como el nombre del empaquetador y del creador del paquete, la URL de donde se puede obtener el código fuente, la licencia con el año del *Copyright*. Este archivo su modificación es relativamente sencilla, solamente se le tiene que añadir la licencia que utilizó para el desarrollo de la solución. (Véase Anexo 1).

### 2.3.4. Archivo *rules*

El último archivo que se propone a modificar en el archivo *rules*. Este, prácticamente, hace todo el trabajo para crear el paquete .deb. Es un *Makefile* con instrucciones para compilar e

instalar la aplicación, luego crear el archivo .deb a partir de los archivos instalados. También tiene instrucciones para revertir la compilación e instalación, eliminando todos los archivos originales y así poder volver a disponer solo del código fuente. Para la modificación de este archivo se hace una clasificación inicialmente de los paquetes o aplicaciones, atendiendo al tipo de lenguaje que utiliza para su implementación, para este caso se propone realizar una diferenciación en cuanto a las aplicaciones de lenguaje interpretado y aplicaciones de lenguajes compilados. Esta clasificación se realiza atendiendo al criterio de que la modificación que se le realiza al archivo rules es diferente y dependerá de qué tipo de aplicación sea (Véase Anexo 1).

#### 2.3.4.1. Aplicaciones de lenguajes de programación interpretados.

Un lenguaje interpretado es un lenguaje de programación que está diseñado para ser ejecutado por medio de un intérprete, en contraste con los lenguajes compilados. Teóricamente, cualquier lenguaje puede ser compilado o ser interpretado, así que esta designación es aplicada puramente debido a la práctica de implementación común y no a alguna característica subyacente de un lenguaje en particular. Sin embargo, hay lenguajes que son diseñados para ser intrínsecamente interpretativos, por lo tanto un compilador causará una carencia de la eficacia. Muchos autores rechazan la clasificación de lenguajes de programación entre interpretados y compilados, considerando que el modo de ejecución (por medio de intérprete o de compilador) del programa escrito en el lenguaje es independiente del propio lenguaje. A ciertos lenguajes interpretados también se les conoce como lenguajes de script. Estos tipos de aplicaciones se recomienda que no se modifique el rules para construir un paquete de código fuente de una aplicación en un lenguaje interpretado lo que se propone a realizar es mantener el archivo *rules* como fue generado originalmente y crear unos ficheros *.install*. Estos ficheros *install* tendrán una línea para cada uno de los archivos a instalar, con el nombre del archivo seguido de un espacio y a continuación el directorio de instalación. La dirección de los nombres de los archivos a copiar es relativa a la carpeta donde se encuentra todo el código fuente, así como la dirección a donde se copiarán, que es relativo a la carpeta temporal que se crea para la generación del paquete binario, en caso de que se esté trabajando en paquetes grandes que separan como resultado de la compilación múltiples paquetes binarios, se debe especifica un fichero .install por cada uno de los binarios a generar, quedando nombre_del_paquete1.install, nombre_del_paquete2.install, etc. (8).

#### 2.3.4.2. Aplicaciones de lenguajes de programación compilados.

En principio, cualquier lenguaje puede ser implementado con un compilador o un intérprete. Sin embargo, es cada vez más frecuente una combinación de ambas soluciones: un compilador puede traducir el código fuente en alguna forma intermedia (muchas veces llamado código de bytes), que luego se pasa a un intérprete que lo ejecuta. Entre los lenguaje compilados más comunes se tienen: C , C++, C# (a *bytecode*), Delphi y Java (a *bytecode*).

Para este caso el archivo rules debe cumplir los requisitos necesarios para que el sistema sea capaz de construir el paquete de código fuente a partir de este tipo de aplicación. Las modificaciones se realizan mayormente utilizando *debhelper* que es una herramienta que facilita este trabajo (8).

#### 2.3.4.3. Debhelper

*Debhelper* ayuda a construir un paquete de Debian. La filosofía que se esconde detrás de *debhelper* es ofrecer una colección de herramientas pequeñas, simples y fáciles de entender que son usadas en debian/rules para automatizar varios aspectos comunes a la hora de construir un paquete. Esto hace que el empaquetador, tenga que realizar menos acciones manualmente y aumenta el nivel de automización de este proceso. Además, si cambian las directrices de Debian, los paquetes que precisan cambios sólo necesitan ser reconstruidos para que se ajusten a las nuevas directrices. Un fichero debian/rules típico que use *debhelper* invocará órdenes de *debhelper* en cadena, o usará dh para automatizar el proceso. De las numerosas órdenes que brinda *debhelper* a continuación se propone las que resultarán más útiles de acuerdo a la propuesta de solución, además de plantear su función permitiendo conocer cuál es el propósito de su uso (9).

**Órdenes de *debhelper*** (9)**:**

A continuación se muestra una lista de las órdenes de *debhelper*.

**dh_testdir:** Comprueba el directorio antes de construir un paquete de Debian.

**dh_clean:** Leerá debian/clean y eliminará los ficheros ahí listados, eliminará ficheros *-stamp del nivel superior.

**dh_installdirs:** Crea subdirectorios en los directorios de construcción del paquete.

**dh_testroot:** Comprueba que el paquete se construye como usuario «root»

**dh_strip:** Específica el nombre del paquete en el que se colocan los símbolos de depuración, no los paquetes desde los que obtener los símbolos.

**dh_compress:** Comprime ficheros y arregla enlaces simbólicos en los directorios de construcción del paquete.

**dh_fixperms:** Arregla los permisos de los ficheros en los directorios de construcción.
**dh_installdeb:** Instala ficheros en el directorio DEBIAN
**dh_installdebconf:** Instala ficheros usados por debconf en los directorios de construcción.
**dh_shlibdeps:** Calcula dependencias sobre bibliotecas compartidas
**dh_gencontrol:** Genera e instala el fichero de control
**dh_md5sums:** Genera el fichero DEBIAN/md5sums
**dh_builddeb:** Construye paquetes binarios de Debian

Para la modificación del archivo rules la solución propuesta determina dos variantes, primeramente si la aplicación a empaquetar fue desarrollada con un lenguaje de programación compilado entonces se procede a modificar el archivo rules poniendo los target siguientes: *clean*, *install*, *build*, *binary-indep*, *binary-arch*, *binary*, luego utilizando herramientas *debhelper* modificar correctamente estos target. Sin embargo en caso de que sea una aplicación desarrollada con un lenguaje de programación interpretado se propone que no se modifique el archivo *rules* y se utilicen archivos .install evitándose así la dificultad que significa modificar el archivo *rules*, además de que para este caso el uso de ficheros .install resulta mucho más sencillo que la modificación de un *makefile* como es el archivo *rules*.

### 2.4. Utilizar dpkg-source

El paquete *dpkg-source* empaqueta y desempaqueta los archivos fuentes de Debian. Se considera una herramienta de manipulación y a través de diferentes opciones permite realizar su funcionalidad fundamental atendiendo a la opción seleccionada y a los parámetros introducidos. De los comandos que brinda dicha herramienta se pueden mencionar los siguientes: -x filename.dsc [salida de directorio], -b directorio [específicas del formato -parámetros], -h, - help y – version. Atendiendo a la solución que se desea brindar en la presente investigación se selecciona el uso de dpkg-source con el comando -b directorio [específicas del formato -parámetros] el cual se explica su funcionamiento a continuación (Véase Anexo 1) (7):

> -b directorio [específicas del formato -parámetros]
>
> Construir un paquete fuente. El primer argumento se toma como el nombre del directorio que contiene el árbol de fuentes debianizada (es decir, con un sub-directorio debian y tal vez los cambios en los archivos originales). Dependiendo del formato de paquete fuente utilizado para construir el paquete, los parámetros adicionales podrían ser aceptados.
> La herramienta *dpkg-source* con el comando –b permite automatizar el último de los procesos necesarios para la obtención de un paquete de código fuente, que es precisamente generar el mismo, así con el uso de dicha herramienta se obtiene un paquete de código fuente o lo

que es lo mismo se tendrá un archivo .dsc y además se tendría debido a las pautas seguidas en procesos anteriores: una carpeta con el código fuente de la aplicación, un .orig.tar.gz y un .debian.tar.gz que serían el resultado final del proceso de construcción del paquete de código fuente (7).

### 2.5. Conclusiones parciales

Se elaboró una propuesta de guía para el empaquetado de paquetes de código fuente para la distribución GNU/Linux Nova usando los conocimientos obtenidos en la investigación. Quedaron plasmadas las actividades necesarias para realizar la construcción de un paquete fuente funcional para la distribución GNU/Linux Nova así como las principales modificaciones realizadas en el proceso de obtención del mismo, se plasmó y recomendó el uso de determinadas herramientas en cada paso del proceso. Se realizó una fundamentación de la guía, planteando el objetivo de cada actividad así como la justificación de por qué el uso de determinadas herramientas.

# 3. Capítulo 3

## 3.1. Introducción

En el presente capítulo se procede a realizar una validación de la propuesta resultante de la investigación, para ello se utilizará el método de estudio de casos, el cual a través de diferentes casos de uso reales de la solución propuesta permitirá comprobar su validez.

## 3.2. Definición de Estudio de Casos

Un estudio de caso es un método de aprendizaje acerca de una situación compleja se basa en el entendimiento comprehensivo de dicha situación, el cual se obtiene a través de la descripción y análisis de la situación, situación tomada como un conjunto y dentro de su contexto (10).

## 3.3. Cuándo utilizar esta herramienta de evaluación.

Realizar uno o diversos estudios de caso consiste en utilizar uno o varios ejemplos reales con el objetivo de profundizar en el conocimiento del tema analizado y, si es posible, extraer una serie de lecciones aplicables al conjunto de la evaluación. En situaciones complejas, el objetivo del estudio de caso es responder a las preguntas « ¿Cómo?» y « ¿Por qué?» a partir de ejemplos concretos adecuadamente seleccionados en función de los objetivos de la evaluación (10).

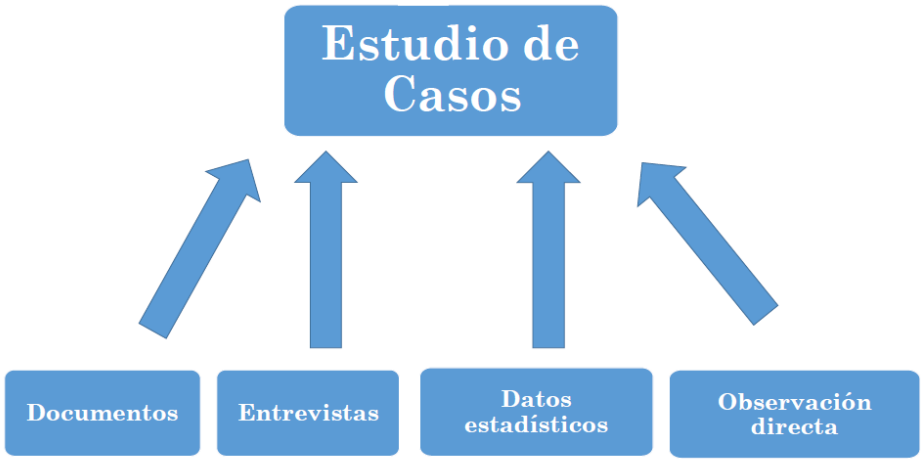



**Imagen 1: Elementos constitutivos del estudio de caso** *(10)*

## 3.4. Ventajas y desventajas (10)

**Ventajas:**

- La abundancia de una información cualitativa detallada que describe de forma clara los contextos de aplicación.
- Relativa simplicidad de uso.
- Flexibilidad para adaptarse a las situaciones en tiempo real.

- Plazos de puesta en práctica compatibles con la evaluación geográfica.
- Delimitar las lógicas de acción de los diferentes actores.

**Desventajas**

- Dificultad para identificar a los interlocutores adecuados.
- Dificultad para identificar los casos y sus límites y para asociarlos a las amplias problemáticas de las evaluaciones geográficas.
- Dificultades para la generalización (por ejemplo a escala de país) de temáticas estudiadas a escala local.
- Coste de la herramienta.
- Esta herramienta sólo permite la interpretación estadística de los datos en raras ocasiones.
- Esta herramienta se basa en el juicio de uno o diversos evaluadores, por lo que, a pesar de adoptar las debidas precauciones al emplearla, puede comportar cierta parcialidad.

### 3.5. Realización del método

Para la realización del método de estudio de casos se seleccionaron 2 aplicaciones sencillas de las cuales se posee su código fuente, una de ellas está realizada con un lenguaje de programación interpretado mientras que por el otro lado se tiene una herramienta realizada con un lenguaje de programación compilado, lo cual permitirá probar la validez de la solución propuesta en ambos casos.

#### 3.5.1. Utilización de la guía para empaquetar el paquete hello

El paquete hello es una aplicación sencilla desarrollada en C que es un lenguaje compilado, la cual permitirá poner en práctica los aspectos, pasos y herramientas establecidas para este tipo de aplicación en la solución propuesta por esta investigación. Siguiendo los pasos planteados en la solución se comenzará a realizar el proceso:

##### 3.5.1.1. Caso de estudio 1

**Paso 1: Debianizar**

Debianizar incluye varias actividades, por lo que no es solo un proceso el que se realizará, este paso incluye realizar varias acciones las cuales se muestran a continuación:

**Obtener el código de la aplicación:**

Esta aplicación se encuentra generalmente en los repositorios de Debian, Ubuntu y GNU/Linux Nova por lo que se puede descargar de dicha localización, en este caso se tomará como si fuera una aplicación donde su código fuente aún no ha sido debianizado ni incluido

en ninguna de estas distribuciones, es decir, se le realizará el proceso de debianizado como si fuera un código fuente intacto.

Teniendo el código fuente de la aplicación a empaquetar se procede a crear el área de trabajo, para este caso significa crear un nuevo directorio con el nombre y la versión que se desea poner a la aplicación una vez empaquetada.

**Creación del directorio de trabajo**.

En este caso se utilizó la terminal de comandos para crear el directorio sin embargo no influye en los resultados del proceso si se realiza de manera manual dicho procedimiento. Luego de realizado este paso se comprueba que en la dirección especificada se ha creado una nueva carpeta que cumpla con las pautas establecidas en la solución propuesta, donde el nombre de la carpeta debe cumplir el formato de nombre-versión (Véase Anexos 1).

**Incluir el código fuente dentro del nuevo directorio**

Ahora se necesita tener el código de la aplicación dentro del directorio de trabajo, por lo que se procede a incluir el mismo en dicha dirección. Luego de esta acción ya se tienen los elementos necesarios para poder comenzar el proceso de empaquetado.

**Utilización del script dh_make.**

Este *script* se debe ejecutar dentro del directorio creado donde se encuentra el código fuente de la aplicación, por lo que se deberá de acceder a dicha carpeta a través de la terminal de comandos. Luego de ejecutado el *script* y una vez especificado el tipo de paquete que se generará, culmina la primera parte del proceso de debianizado como se plantea en la solución propuesta.

Se comprueba que se hayan creado los elementos correctamente, correspondientes a la realización de los pasos anteriores verificando primeramente que en la dirección donde se creó el nuevo directorio aparece un nuevo archivo con formato .orig.tar.gz y dentro del directorio creado inicialmente aparece una nueva carpeta con nombre debian que contiene una serie de ficheros, como se muestra en las imágenes siguientes.

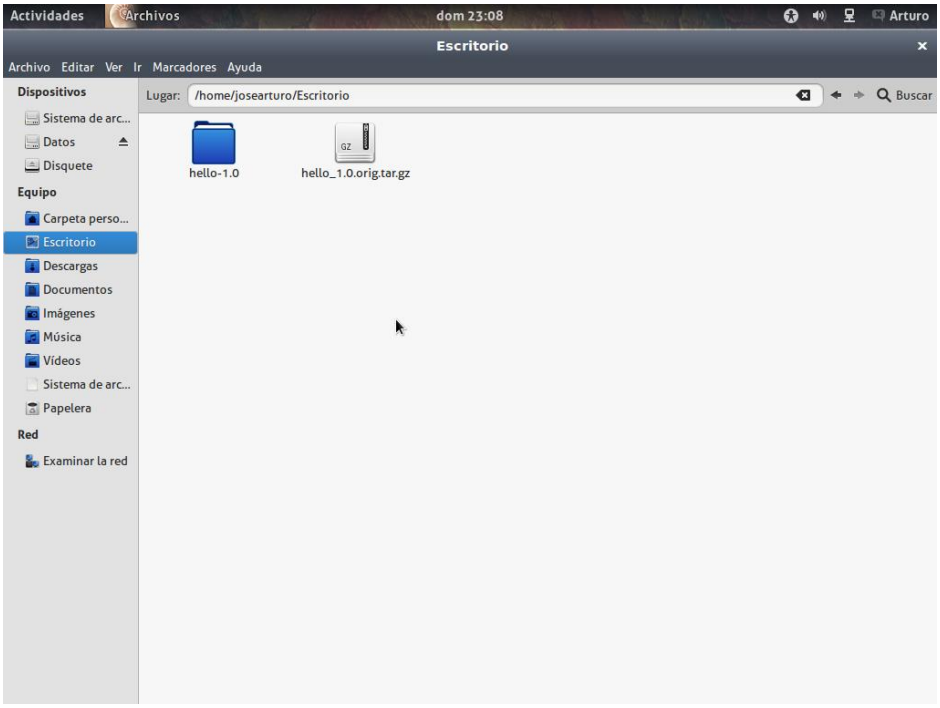



**Imagen 2: Creación del comprimido .orig.tar.gz**

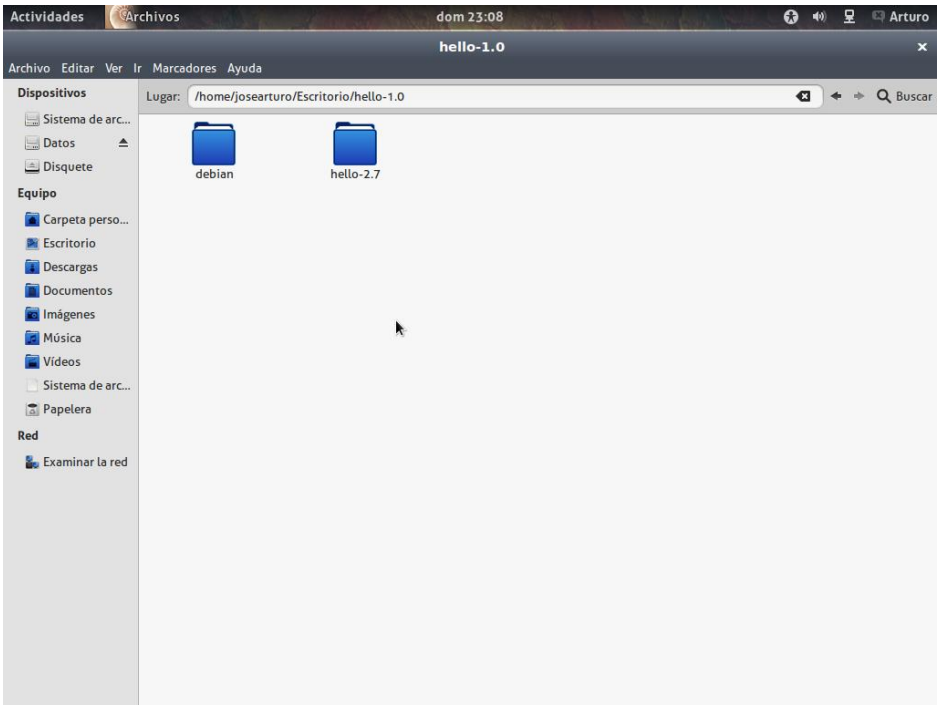



**Imagen 3: Creación de la carpeta debian**

Siguiendo los pasos establecidos en la solución propuesta se procede a eliminar todas las plantillas que no se utilizarán, esto se materializa en la eliminación de todos los ficheros que tengan el formato .EX o .ex, y los ficheros README.Debian y README.source.

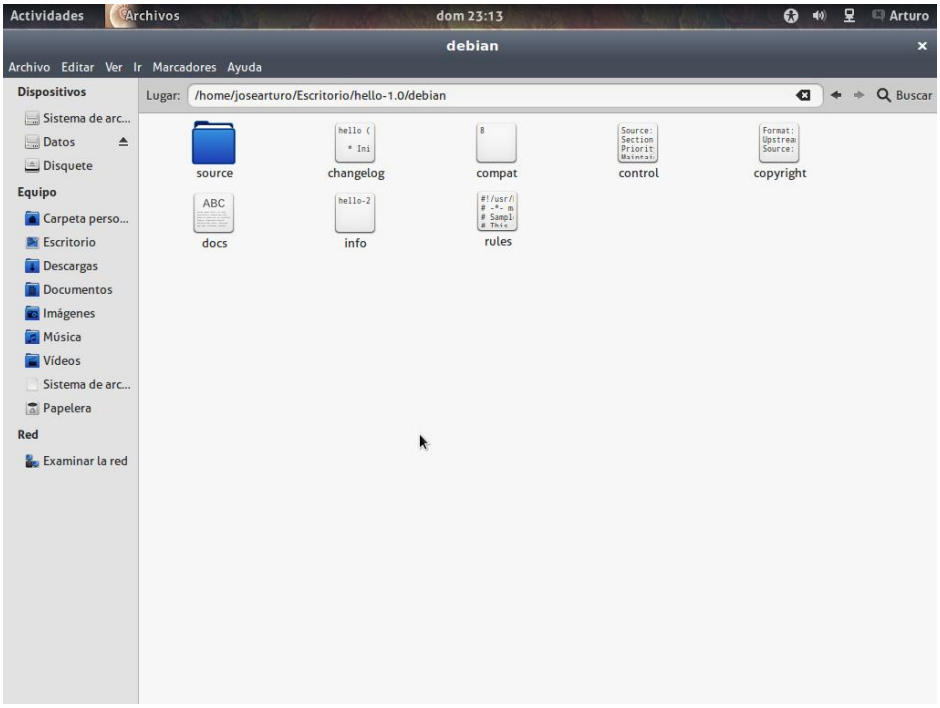



**Imagen 4: Carpeta debian sin archivos *examples***

Con este último paso se han terminado las actividades incluidas dentro del proceso de debianizado por lo que se continúa con el siguiente proceso.

### Paso 2: Modificación de los archivos del directorio debian.

Luego de eliminados los ficheros que no se utilizarán en el proceso, se procede a modificar los archivos restantes según lo establecido por la guía propuesta. Se comenzará por modificar el archivo changelog.

El *script* dh_make creó estos ficheros inicialmente con un formato y valores por defecto. Luego de ajustarlo a las características de la distribución GNU/Linux Nova, se muestra de la siguiente forma:

```
hello (2.0-0nova1) 2013; urgency=low

 * Initial release (Closes: #nnnn) <nnnn is the bug number of your ITP>

 -- Arturo <jacastelo@estudiantes.uci.cu> Mon, 02 Jun 2014 08:45:21 -0400
```

Destacar que la principal modificación realizada es en el campo donde se muestra la versión del paquete, según lo establecido por la solución propuesta, si un paquete es nativo de GNU/Linux Nova este campo debería tener el siguiente formato versión-0nova1, lo cual indica que es un paquete nativo para GNU/Linux Nova.

La siguiente acción que se realiza es la modificación del archivo control donde se describe que tipo de paquete es, cuáles son sus dependencias a la hora de compilarse y a la hora de ejecutarse entre otras características. La principal modificación planteada por la solución propuesta y que es importante en este fichero, es tener bien claro que como se está debianizado una aplicación de código compilado en el campo *Architecture* debe tener el valor *any*, lo cual indica que esta aplicación tiene que ser compilada para cada una de las arquitecturas en la cuales se va a utilizar. Luego de modificados estos campos el archivo control queda de la siguiente manera:

```
Source: hello
Section: utils
Priority: optional
Maintainer: Arturo jacastelo@estudiantes.uci.cu
Build-Depends: debhelper (>= 8.0.0)
Standards-Version: 3.9.2
Homepage: www.nova.uci.cu
#Vcs-Git: git://git.debian.org/collab-maint/hello.git
#Vcs-Browser: http://git.debian.org/?p=collab-maint/hello.git;a=summary

Package: hello
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Description: Esta es una aplicación de prueba
 Esta es una aplicación de prueba para demostrar el empaquetado en las aplicaciones de
código compilado.
```

La modificación del archivo *copyright* depende de la licencia bajo la cual se desarrolló dicha aplicación por lo que su modificación será incluir el texto de dicha licencia en el archivo.

En este caso de estudio como su principal función es mostrar la validez de la solución propuesta para todo el proceso de empaquetado, nos limitaremos a utilizar la licencia GNU-GPL 2 que viene por defecto establecida en esta plantilla, por lo tanto este fichero no recibirá modificación alguna.

A continuación se realizará el paso fundamental y donde se muestra la diferencia principal descrita por la guía propuesta, que es la modificación del archivo rules. Para este caso como se especifica en la propuesta de solución se procede a realizar los cambios necesarios en dicho archivo quedando de la siguiente manera:

```
#!/usr/bin/make -f

package="hello"

clean:

        dh_testdir

        dh_clean

        rm –f build

install: build

        dh_clean

        dh_installdirs

        $(MAKE) destdir=$(CURDIR)/debian/$package install

build:

        $(MAKE) prefix=$(CURDIR)/debian/tmp

        touch build

binary-indep:



binary-arch:

        dh_testdir -a

        dh_testroot -a

        dh_strip -a

        dh_compress -a

        dh_fixperms -a

        dh_installdeb -a

        dh_shlibdeps -a
```

```
        dh_gencontrol -a

        dh_md5sums -a

        dh_builddeb -a

binary: binary-indep binary-arch
```

Es necesario aclarar, como se plantea en la propuesta de solución que esta aplicación debe estar implementado con la ayuda de las *autotools* o al menos presentar los ficheros que se generan de utilizar esta herramienta.
Para este caso de estudio ya se ha finalizado el proceso de modificación de los ficheros del directorio debian por lo que se concluye esta parte del proceso.

**Paso 3: Generando el paquete de código fuente**

Luego de concluido el paso anterior ya se poseen los archivos necesarios para crear el paquete de código fuente, además estos ya han sido modificados de acuerdo a los aspectos y pautas planteados en la solución propuesta, necesarios para que el paquete de código fuente obtenido funcione correctamente en GNU/Linux Nova, por lo que solamente quedaría generar dicho paquete de código fuente.
Como se propone en la solución se usa el comando dpkg-source para generar este paquete de código fuente para ello se realizan los siguientes pasos:

1. Se accede a través de la terminal a la carpeta que contiene el código fuente de la aplicación.
2. Se utiliza dpkg-source de la siguiente manera:

```
dpkg-source –b .
```

Luego se pasa a comprobar si se generó correctamente el paquete de código fuente, para ello se comprueba si se creó en el área de trabajo los archivos que se plantean en la solución propuesta, en este caso debido al uso del formato 3.0 (quilt), deben existir los comprimidos hello_2.0.orig.tar.gz, hello_2.0-0nova1.debian.tar.gz y el fichero hello_2.0-0nova1.dsc, comprobándose que se realizó satisfactoriamente el proceso de empaquetado de una aplicación con lenguaje compilado, siendo compatible con la distribución GNU/Linux Nova, como se muestra en la siguiente imagen:

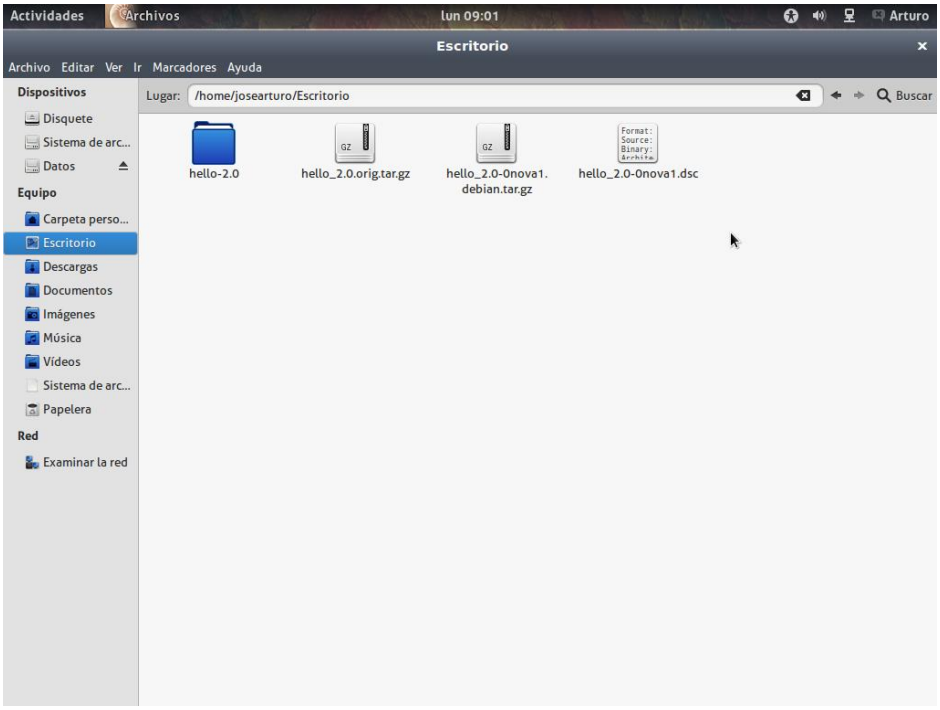



**Imagen 5: Resultados obtenidos del uso de dpkg-source**

#### 3.5.1.2. Caso de estudio 2

Para el segundo caso de estudio se seleccionó una aplicación sencilla desarrollada por miembros del proyecto Nova utilizando un lenguaje de programación interpretado (Python), es una aplicación que permite grabar los canales de audio de la web, esta aplicación servirá para realizar las diferentes actividades y comprobar si se obtienen los resultados esperados.

##### Paso 1: Debianizar

En este paso se realizarán las mismas actividades que en el caso de estudio anterior (véase caso de estudio 1), dado que no existe ningún cambio en esta actividad planteada por la solución propuesta por lo que no se plasmarán estas nuevamente.

##### Paso 2: Modificación de los archivos del directorio debian

Durante la realización de este paso se procede de igual manera que el caso de estudio anterior, hasta llegar a la modificación del fichero control donde, según la solución propuesta, el campo *Architecture* debe tener como valor *all*, especificando así que esta aplicación no necesita compilarse para cada una de las arquitecturas para las cuales pueda usarse, pues dicha aplicación tiene como características que es de lenguaje interpretado por lo que no es necesario compilarlo para todas las arquitecturas, quedando de la siguiente manera:

```
Source: grabar

Section: unknown

Priority: extra

Maintainer: Arturo jacastelo@estudiantes.uci.cu

Build-Depends: debhelper (>= 8.0.0)

Standards-Version: 3.9.2

Homepage:www.nova.uci.cu

#Vcs-Git: git://git.debian.org/collab-maint/grabar.git

#Vcs-Browser: http://git.debian.org/?p=collab-maint/grabar.git;a=summary



Package: grabar

Architecture: all

Depends: ${shlibs:Depends}, ${misc:Depends}

Description: Esta es una aplicación de prueba.

 Esta es una aplicación de prueba para el probar el proceso de empaquetado en GNU/Linux

Nova.
```

A diferencia del caso de uso anterior donde se modificó el archivo *rules*, en este caso se crea y modifica un nuevo fichero llamado *install*, el sigue la estructura planteada para el mismo en la solución propuesta. En este archivo se especifica en qué direcciones se copiarán cada uno de los ficheros necesarios para que la aplicación funcione correctamente, quedando la siguiente estructura:

```
 codigo/*        usr/lib/grabar

images/grabar.png     usr/shared/pixmaps/grabar.png

images/*        usr/shared/grabar/images

applications/grabar.desktop   usr/shared/applications/grabar.desktop
```

**Paso 3: Generando el paquete de código fuente**

En este paso se procede de manera similar al caso de estudio 1 dado que durante la realización de estas actividades no influye que la aplicación sea de un lenguaje compilado o interpretado, se realizarán las mismas acciones que en el caso de estudio 1 (véase caso de estudio 1).

### 3.6. Conclusiones parciales

El uso de la propuesta de solución para generar el paquete de código fuente de dos aplicaciones que responden a los dos caminos diferentes que se especifican en la solución propuesta y la obtención de un paquete de código fuente correctamente generado y funcional para la distribución GNU/Linux Nova garantiza la validez de la solución. La obtención de los resultados esperados durante el uso de la propuesta de solución brindó la certeza de la efectividad de las actividades propuestas para obtener el paquete de código fuente para la distribución cubana GNU/Linux Nova. Los pasos definidos garantizan una correcta generación de un paquete de código fuente funcional para la distribución anteriormente mencionada.

**Conclusiones Generales**

Con la culminación de la presente investigación se cumplieron los objetivos y metas planteados al inicio de la misma:

- Con análisis de las principales guías de empaquetado existentes se evidenció que no existe un procedimiento claramente definido para lograr el empaquetado de todo tipo de aplicaciones en los sistemas GNU/Linux. Además, se concluyó con este estudio, que existen varias vías de construcción de paquetes de código fuente, cada una de ellas con características específicas, pero no existía una descripción formal que explicara cuál de estas vías o el conjunto de ellas era la más idónea a utilizar para un entorno de desarrollo similar la de la UCI.
- Con el uso de los distintos procedimientos de construcción de paquetes de código fuente estudiados se llega a la obtención de una solución que se adecua al proceso de desarrollo de la UCI y que permite la integración de estas aplicaciones con la distribución GNU/Linux Nova.
- A través del uso de casos de estudio a los cuales se les aplicó la propuesta de solución se logró obtener paquetes de código fuente para la distribución GNU/Linux Nova demostrándose la validez de la solución propuesta.

**Recomendaciones**

Se recomienda aplicar la solución propuesta como documentación para el proceso de integración de futuras soluciones UCI con la distribución GNU/Linux Nova. Además se sugiere profundizar dicho proceso para extender la solución propuesta hasta la obtención de paquete de código binario instalable en la distribución anteriormente mencionada. Se recomienda utilizar la investigación como documentación para la capacitación del personal, en el proceso de empaquetado para la distribución cubana GNU/Linux Nova.

# BIBLIOGRAFÍA REFERENCIADA

**Bibliografía Referenciada**


1. **Dariem Pérez Herrera, , Sonia Guerrero Lambert , Yusleydi Fernández del Monte, Miguel Albalat Águila, Jorge Luis Machin, Héctor Pérez Baranda, Ricardo Quevedo Mejías.** *Estado actual de los sistemas construcción paquetes en Estado actual de los sistemas construcción paquetes en diferentes distribuciones GNU/Linux.* La Habana : Ediciones Futuro, 2012. pág. 17. Vol. 6. ISSN: 1994-1536 | e-ISSN: 2227-1899.

2. **Josip Rodin, Osamu Aoki, Javier Fernández-Sanguino Peña, David Martínez, Ana Beatriz Guerrero López, Francisco Javier Cuadrado, y Innocent De Marchi.** *Guía del nuevo desarrollador de Debian.* 2011. pág. 56.

3. **Nussbaum, Lucas.** Guía de creación de paquetes Debian. 31 de Octubre de 2013. pág. 82.

4. **Jordan Mantha, Alexandre Vassalotti, Jonathan Patrick Davies, Ankur Kotwal,.** *The Ubuntu Packaging Guide.* 2006. pág. 72.

5. **López, Jorge Rodríguez.** UbuntuWiki. *Guia de empaquetamiento Completa.* [En línea] 20 de mayo de 2012. [Citado el: 15 de febrero de 2014.] https://wiki.ubuntu.com/PackagingGuide/Complete.

6. **Christoph Lameter, Craig Small, Bruce Sass.** UbuntuManpages. *man dh_make.* [En línea] 10 de Abril de 2008. [Citado el: 23 de marzo de 2014.] http://manpages.ubuntu.com/manpages/karmic/en/man8/dh_make.8.html.

7. **Akkerman, Wichert.** UbuntuManpages. *man dpkg-source.* [En línea] 16 de Marzo de 2008. [Citado el: 16 de enero de 2014.] http://manpages.ubuntu.com/manpages/gutsy/man1/dpkg-source.1.html.

8. **G, Leopoldo Sanclemente.** leosanblog. [En línea] 5 de septiembre de 2012. [Citado el: 16 de Marzo de 2014.] http://leosanblog.wordpress.com/2012/09/05/diferencias-entre-lenguajes-de-programacion-compilados-e-interpretados/.

9. **Hess, Joey.** UbuntuManpages. *man debhelper.* [En línea] 2010. [Citado el: 23 de Mayo de 2014.] http://manpages.ubuntu.com/manpages/intrepid/es/man7/debhelper.7.html.

10. **Lund, María Inés, Aballay, Laura N., Torres, Estela Herrera, Myriam Ormeño, Emilio G.** *Validación de Usabilidad de una plantilla para documentar casos de uso.* Red de Universidades con Carreras en Informática (RedUNCI). Argentina : s.n., 2011.

# BIBLIOGRAFÍA CONSULTADA

**Bibliografía Consultada**


1. **Dariem Pérez Herrera, , Sonia Guerrero Lambert , Yusleydi Fernández del Monte, Miguel Albalat Águila, Jorge Luis Machin, Héctor Pérez Baranda, Ricardo Quevedo Mejías.** *Estado actual de los sistemas construcción paquetes en Estado actual de los sistemas construcción paquetes en diferentes distribuciones GNU/Linux.* La Habana : Ediciones Futuro, 2012. pág. 17. Vol. 6. ISSN: 1994-1536 | e-ISSN: 2227-1899.

2. **Josip Rodin, Osamu Aoki, Javier Fernández-Sanguino Peña, David Martínez, Ana Beatriz Guerrero López, Francisco Javier Cuadrado, y Innocent De Marchi.** *Guía del nuevo desarrollador de Debian.* 2011. pág. 56.

3. **Nussbaum, Lucas.** Guía de creación de paquetes Debian. 31 de Octubre de 2013. pág. 82.

4. **Jordan Mantha, Alexandre Vassalotti, Jonathan Patrick Davies, Ankur Kotwal,.** *The Ubuntu Packaging Guide.* 2006. pág. 72.

5. **López, Jorge Rodríguez.** UbuntuWiki. *Guia de empaquetamiento Completa.* [En línea] 20 de mayo de 2012. [Citado el: 15 de febrero de 2014.] https://wiki.ubuntu.com/PackagingGuide/Complete.

6. **Christoph Lameter, Craig Small, Bruce Sass.** UbuntuManpages. *man dh_make.* [En línea] 10 de Abril de 2008. [Citado el: 23 de marzo de 2014.] http://manpages.ubuntu.com/manpages/karmic/en/man8/dh_make.8.html.

7. **Akkerman, Wichert.** UbuntuManpages. *man dpkg-source.* [En línea] 16 de Marzo de 2008. [Citado el: 16 de enero de 2014.] http://manpages.ubuntu.com/manpages/gutsy/man1/dpkg-source.1.html.

8. **G, Leopoldo Sanclemente.** leosanblog. [En línea] 5 de septiembre de 2012. [Citado el: 16 de Marzo de 2014.] http://leosanblog.wordpress.com/2012/09/05/diferencias-entre-lenguajes-de-programacion-compilados-e-interpretados/.

9. **Hess, Joey.** UbuntuManpages. *man debhelper.* [En línea] 2010. [Citado el: 23 de Mayo de 2014.] http://manpages.ubuntu.com/manpages/intrepid/es/man7/debhelper.7.html.

10. **Lund, María Inés, Aballay, Laura N., Torres, Estela Herrera, Myriam Ormeño, Emilio G.** *Validación de Usabilidad de una plantilla para documentar casos de uso.* Red de Universidades con Carreras en Informática (RedUNCI). Argentina : s.n., 2011.

11. **Wirzenus, Lars.** Debian Wiki. [En línea] 11 de Enero de 2011. [Citado el: 5 de Marzo de 2014.] https://wiki.debian.org/es/IntroDebianPackaging?action=fullsearch&context=180&value=linkto%3A%22es%2FIntroDebianPackaging%22.

# BIBLIOGRAFÍA CONSULTADA

12. **Samayoa, Josué Miguel Abarca.** 3.0 quilt. Guatemala : s.n., 21 de marzo de 2010. pág. 40.

13. **Ramírez, Juan Alejandro Cortés.** *Método de estudio de casos como estrategia de investigación aplicada en organizaciones.* MEDELLIN, COLOMBIA : s.n., 2008. pág. 14. ISSN 1554-7752.

14. **Mora, Carlos Alejandro Guerrero.** *Guía_de_Referencia_para_el_Desarrollador.* 2010.

15. **Juan Diego Lopera Echavarría, Carlos Arturo Ramírez Gómez, Jeniffer Ortiz Venegas.** *El método analítico como método natural.* Universidad de Antioquia. Colombia : s.n., 2010. ISSN 1889-7231.

16. **Iniesta, Alberto González.** Creación de paquetes Debian. *Presentación.* España : s.n., febrero de 2009. pág. 25.

17. —. Creación de paquetes Debian. febrero de 2009.

18. **Fernández, Orlando Gabriel Cárdenas.** *Material de apoyo para la preparación didáctica de las clases.* Universidad de las Ciencias informáticas. La Habana : s.n., 2013.

19. **Duret-Lutz, Alexandre.** Using GNU Autotools. *Tutorial.* 16 de mayo de 2010. pág. 162.

20. **David MacKenzie, Tom Tromey, Alexandre Duret-Lutz, Ralf Wildenhues, Stefano Lattarini.** *GNU Automake.* Software Libre. 2013. pág. 182, Manual de usuario.

# ANEXOS

## 4. Anexos

Anexo 1: Guía para la integración de las aplicaciones UCI con GNU/Linux Nova.

### 4.1. Introducción

La presente guía refleja un compendio de pasos específicos para lograr la construcción de un paquete de código fuente capaz de funcionar correctamente en la distribución cubana GNU/Linux Nova. Dichos procedimientos son la selección de las mejores prácticas en la actualidad y describen el proceso actual de empaquetado del proyecto Nova y sus mantenedores.

### 4.2. Debianizar la aplicación.

#### 4.2.1. Preparando el sistema.

Lo primero a realizar será configurar y preparar el entorno de trabajo, de manera que se posean todas las herramientas y condiciones necesarias para realizar la actividad propuesta. Para ello se propone por parte de la guía que el sistema operativo a utilizar sea alguno de los siguientes:

- ✓ Debian
- ✓ Ubuntu
- ✓ Nova

En la guía se recomienda utilizar como sistema operativo la distribución cubana GNU/Linux Nova ya que existen configuraciones y modificaciones a ficheros que se utilizarán en este proceso que teniendo este sistema operativo instalado se realizarán automáticamente, lo que podría ayudar a reducir la cantidad de trabajo realizado por los usuarios, así como las modificaciones a realizar

#### 4.2.2. Aplicaciones a instalar previo al comienzo del proceso de debianizado.

El proceso de empaquetado necesita algunas herramientas para poder ser ejecutado, por ello se recomienda que inicialmente, antes de comenzar a realizar cualquier actividad se instalen las siguientes herramientas:

- ✓ ***build-essential:*** es un metapaquete que depende de *libc6-dev*, *gcc*, g++, *make* y *dpkg-dev*.

- ✓ ***debhelper:*** es un conjunto de *scripts* que realizan tareas comunes de empaquetado.
- ✓ ***dh_make:*** es una herramienta para convertir un paquete de código fuente regular en uno formateado de acuerdo con los requisitos de la normativa de Debian.
- ✓ ***dpkg-dev****:* Uno de los paquetes con los que se puede no estar familiarizado es *dpkg-dev*. Este contiene herramientas como *dpkg-buildpackage* y *dpkg-source* que se usan para crear, desempaquetar y construir paquetes binarios o fuentes.
- ✓ ***devscripts:*** contiene muchos *scripts* que facilitan considerablemente las tareas de un mantenedor de paquetes. Algunos de los más usados son *debdiff*, *dch*, *debuild* y *debsing*.

**4.3. Configurar dh_make.**

Se considera como buena práctica del mantenedor configurar *dh_make* previo a su utilización, dado que muchos de los archivos que se generan luego con el script *dh_make* requieren tener la información de mantenedor por lo que es favorable configurar estos valores previamente en aras de no tener que repetirlos, tan solo tenerlos almacenados en una variable que facilitará esta tarea. Para ello se debe configurar las variables de entorno «*shell*» *$DEBEMAIL* y *$DEBFULLNAME*, que son utilizadas luego por varias herramientas de mantenimiento para obtener el nombre y correo electrónico del mantenedor como se indica en el ejemplo a continuación:

```
$ cat >>~/.bashrc <<EOF

DEBEMAIL="jacastelo@estudiantes.uci.cu"

DEBFULLNAME="José  Arturo Castelo Rojas"

export DEBEMAIL DEBFULLNAME

EOF

$ . ~/.bashrc
```

**4.4. Crear un directorio con el nombre de la aplicación y su versión.**

Con las configuraciones previas realizadas entonces se necesita crear un directorio para realizar el trabajo, este directorio deberá de estar completamente vacío para iniciar las actividades. Para crear este nuevo

directorio se puede realizar de manera visual o a través de la terminal de comandos. Para este paso se recomienda realizar esta actividad utilizando la opción que mejor le sea al usuario, la selección de la manera en que se crea este directorio no presenta ningún conflicto para el procedimiento. Estos nombres para los directorios que almacenan el código fuente deberán cumplir con ciertas pautas que se definen para nombrar estas carpetas que serán utilizadas para el proceso de construcción del paquete de código fuente, en caso contrario podrán ocurrir errores a la hora de acceder a ellos o de utilizarlos en el procedimiento dado que el sistema operativo no reconocerá como válidos sus nombres. Por lo que se recomienda para esta actividad nombrar el directorio de forma que contenga solo letras minúsculas (a-z), además puede contener dígitos (0-9), el símbolo de suma (+), resta (-) y puntos (.). Debe contener al menos dos caracteres, empezar con un carácter alfanumérico y no coincidir con alguno de los paquetes ya existentes. Es una buena práctica limitar su longitud a un máximo de 30 caracteres.

**4.5. Incluir el código de la aplicación o paquete ya esté estructurado o no.**

Este paso se incluye para hacer una salvedad sobre este momento en el procedimiento, luego de creado el directorio donde se va a trabajar se puede entonces copiar el código fuente de la aplicación dentro del mismo. Este código fuente puede encontrarse de manera estructurada o no, es decir, puede estar bajo alguna estructura organizativa definida por el IDE de desarrollo utilizado, puede ser una estructura decida por el usuario o puede ser un conjunto no organizado de carpetas que poseen el código dentro de ellas.

**4.6. Ejecutar el script dh_make**

Se utiliza el comando: dh_make –e ¨correo electrónico del mantenedor¨ –createorig un ejemplo sería de la siguiente manera:

```
dh_make –e $DEBMAIL –createorig
```

Luego de dar <<Enter>> dh_make solicita que se seleccione que tipo de paquete desea construir, como se muestra a continuación:

Type of packager *single binary*, *multiple binary*, *libreary*, *kernel module* or *cdbs*? [s/m/l/b],

- La opción «s» para programas que generen un solo paquete deb. Normalmente, es la opción más utilizada.

- La opción «m» genera más de un paquete deb (por ejemplo los juegos que tienen 2 paquetes juego.deb y juego-data.deb).
- La opción «l» para bibliotecas.
- La opción «k» se utiliza para módulos del kernel Linux.
- La opción «b» utiliza el programa cdbs para generar el archivo rules del paquete deb.

En este paso generalmente para GNU/Linux Nova se construyen paquetes simples o múltiples lo cual genera un paquete .deb o varios paquetes .deb respectivamente a partir de un paquete de código fuente, por lo que se selecciona la opción s o m en dependencia del resultado que se desee obtener.

Luego de realizada esta operación dentro de nuestro directorio estará una nueva carpeta nombrada debian/ y dentro de ella un conjunto de ficheros de configuración y de información de los cuales se profundizará en el epígrafe a continuación.

## 4.7. Modificación de los archivos necesarios para la construcción del paquete fuente.

### 4.7.1. Borrar los ficheros .ex y los .EX.

El último paso descrito antes de empezar este epígrafe fue utilizar el comando dh_make este comando crea una plantilla de ficheros que se utilizarán luego para la generación del paquete fuente. Estas plantillas no todas son utilizadas en la construcción del paquete de código fuente por lo que en esta guía se propone la eliminación de estos ficheros que no utilizaremos y en caso de que se necesite utilizarlos en un futuro se crearán manualmente por el mantenedor del paquete.

Para eliminar estos archivos se utiliza el siguiente comando:

```
rm /debian/*.ex /debian/*.EX
```

### 4.7.2. Archivo changelog

Este es un fichero requerido, con un formato especial, este es el formato utilizado por *dpkg* y otros programas para obtener el número de versión, revisión, distribución y urgencia del paquete. Es también importante, tener documentados todos los cambios que se hayan hecho. Esto ayudará a las personas que se descarguen el paquete, notar si hay temas pendientes en el mismo que deberían conocer de forma inmediata. Se

guardará como usr/share/doc/gentoo/changelog.Debian.gz en el paquete binario. Al ejecutar *dh_make* este lo creará de la siguiente manera:

```
nombre_del_paquete (versión) distribución; urgency=prioridad/urgencia

 Initial release

--Nombre del mantenedor <correo.electrónico>

fecha_de_creación_del_paquete
```

Esto sería un archivo *changelog* vacío, por lo que se debe llenar con los datos correctos cada campo de este archivo lo cual se vería de la siguiente manera:

```
ejemplo (2.1.1-0nova1)2013; urgency=low

* Información de los cambios realizados

 - Detalles del cambio realizado

 -- José Arturo Castelo Rojas <jacastelo@estudiantes.uci.cu> Wed, 8 Apr

2013
```

Este archivo es importante que se modifique correctamente o que se cree correctamente porque además de brindar la información sobre los cambios realizados al paquete, es la forma fundamental de reconocer para qué distribución fue empaquetado dicho paquete.

Primero hay que aclarar qué significan cada uno de sus campos:

La línea 1 es el nombre del paquete, versión, revisión, distribución y urgencia. El nombre debe coincidir con el nombre del paquete fuente, la distribución debería ser para la cual el paquete está siendo empaquetado, como se puede notar en ejemplo anterior esta versión sería 2013. Este archivo permite identificar para qué distribución fue empaquetado originalmente el paquete. En la primera línea del archivo se ponen el nombre del paquete, versión y revisión en este último campo es donde se muestra la diferencia entre las distribuciones, a continuación un ejemplo: Hello (1.0-1ubuntu3), estos valores que significan

- ✓ Hello es el nombre del paquete
- ✓ 1.0 es la versión del paquete

- ✓ -1ubuntu3 este es el campo importante -1 significa que Debian le hizo una modificación y ubuntu3 significa que Ubuntu le ha realizado 3 modificaciones.

En caso de que el paquete sea nativo es decir propio de una distribución este campo (revisión) se muestra de la siguiente manera hello (1.0-0ubuntu1), esto representaría que el paquete es nativo de Ubuntu. Es de interés para la investigación definir estos procedimientos para la distribución cubana GNU/Linux Nova por lo que se propone que para dicha distribución esta línea del archivo *changelog* quedaría así:
hello (1.0-0nova1) 2013|2015|….
Todas estas modificaciones al archivo *changelog* se pueden hacer utilizando el comando *dch* o *debchange* con los parámetros que estos soportan, por ejemplo:

*dch* –i creará una nueva entrada en el *changelog* donde solamente tendrá que especificar los cambios realizados al código todos los parámetros anteriormente vistos se autogenerarán con ayuda de la automatización que provee estas aplicaciones.
*dch* –e Modificará la última entrada del *changelog*.
Entre otras que existen para mayor información puede dirigirse al man del paquete ya sea man *dch* o man *debchange*.

### 4.7.3. Archivo control

El archivo control contiene la información que el administrador de paquetes (como *dpkg, apt, aptitude, Synaptic y Adept*) usa para el proceso de instalación y configuración del paquete en el sistema: dependencias necesarias, información del mantenedor, etc.
A continuación se utilizará un ejemplo para mostrar los campos que contiene este fichero.

```
1 Source: hello

2 Section: python

3 Priority: extra

4 Maintainer: Jose Arturo Castelo Rojas <jacastelo@estudiantes.uci.cu>

5 Build-Depends: debhelper (>= 7.0.50~)
```

```
6 Standards-Version: 3.8.4

7 Homepage: <introduce aquí la URL del autor original >

8

9 Package: hello

10 Architecture: any

11 Depends: ${shlibs:Depends}, ${misc:Depends}

12 Description: <insertar hasta 60 caracteres de descripción>

13 <inserta una descripción larga, indentada con espacios.>
```

- ***Source*:** indica el nombre del paquete fuente que debe coincidir con el nombre de la carpeta que contiene el código fuente que se está trabajando.
- ***Section*:** indica la categoría del paquete dentro de un repositorio *apt* que es reconocida por gestores de paquetes como *Synaptic* o *Adept*.

Las categorías disponibles son:

***Tabla 1: Categorías del campo section***

| Nombre | Descripción | Nombre | Descripción | Nombre | Descripción | Nombre | Descripción |
|---|---|---|---|---|---|---|---|
| **admin** | Administración del sistema | **kde** | Escritorio KDE | **shells** | Shells | **graphics** | Gráficos |
| **base** | Sistema base | **libs** | Bibliotecas | **sound** | Sonido y vídeo | **comm** | Comunicación |
| **libdevel** | Bibliotecas de desarrollo | **tex** | TeX, autoría | **contrib** | Contrib | **mail** | Correo electrónico |
| **utils** | Utilidades/Herramientas | **devel** | Desarrollo | **math** | Matemáticas | **web** | Web (World Wide Web) |
| **doc** | Documentación | **Misc** | Misceláneos - Basados en texto | **x11** | Misceláneos - Gráficos | **editors** | Editores |
| **net** | Redes. | **interpreters** | Lenguajes de ordenador interpretados | **electronics** | Electrónica | **news** | Grupos de noticias |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| **science** | Ciencia | **embedded** | Dispositivos integrados | **non-free** | No libre | **hamradio** | Radio |
| **games** | Juegos | **oldlibs** | Bibliotecas antiguas | **python** | Lenguaje de programación Python | **gnome** | Escritorio Gnome |
| **otherosfs** | Plataformas cruzadas | **perl** | Lenguaje de programación Perl | | | | |

- ***Priority*:** indica la prioridad del paquete en el sistema. Se dispone de las siguientes opciones:
  - ✓ ***Required***: paquetes esenciales para que el sistema trabaje adecuadamente. Si son eliminados hay un alto riesgo de que el sistema quede inutilizable.
  - ✓ ***Important***: un conjunto mínimo de paquetes para que un sistema sea usable. Eliminar estos paquetes no produce un quiebro irrecuperable del sistema, pero generalmente se consideran herramientas importantes sin las cuales la instalación de Linux sería incompleta. Nota: Esto no incluye programas como *Emacs* o incluso el servidor X.
  - ✓ ***Standard***: poca explicación hay sobre ella.
  - ✓ ***Optional***: en esencia esta categoría es para los paquetes no-requeridos, o la mayor parte de los paquetes. Sin embargo, estos paquetes no deberían presentar ningún conflicto con otros.
  - ✓ **Extra**: paquetes que pueden presentar conflictos con otros paquetes en alguna de las categorías anteriores.
- ***Maintainer*:** muestra el nombre del empaquetador y su correo electrónico de contacto en caso de algún problema con el paquete u otras cosas.
- **Standards-Version:** Especifica la versión de la Normativa de Debian (Debian *policy*) que sigue el paquete. Esta es propia de cada versión de Ubuntu y por ello los paquetes deb deben seguirla.
- **Build-Depends:** esta es una parte esencial del empaquetado del deb y quizás una de las que pueden llegar a ser una complicación si no se conocen las dependencias del paquete.

- **Homepage:** una URL donde se puede encontrar más información acerca del *software*. Para la distribución GNU/Linux Nova que es en la que se centra esta investigación se colocará en este campo la siguiente URL: www.nova.uci.cu que es la que se utiliza en los paquetes de dicha distribución.

La siguiente descripción es para el paquete binario que será construido a partir del código fuente. Si hay más de un paquete binario que se genere del código fuente se debe redactar un párrafo que se describa cada uno.

- **Package:** Esto especifica el nombre del paquete deb pero no la firma ni la etiqueta de la arquitectura. Los nombres que aparezcan en Package y Source deben ser el mismo.
- **Architecture:** La arquitectura para la cual será construido el paquete. Algunos ejemplos son:
  - ✓ **all** - El programa a empaquetar no depende de la arquitectura del sistema. Se refiere a programas hechos en lenguajes interpretados o independientes de la plataforma, como Python o Java. El resultado será un paquete binario cuyo nombre acabará en _all.deb.
  - ✓ **any** - El programa a empaquetar depende de la arquitectura del sistema y debe ser compilado en todas las arquitecturas compatibles. Habrá un archivo .deb para cada arquitectura (_i386.deb por ejemplo).

Se puede optar por una o más de un subconjunto de arquitecturas (i386, amd64, ppc, etc.) para indicar que el programa a empaquetar depende de una arquitectura concreta y no funciona para todas las arquitecturas soportadas por el sistema operativo. Para el caso de Nova esta distribución soporta como arquitectura: i386 y amd64.

- **Depends:** esto indica al sistema las dependencias del paquete deb resultante de todo el proceso, normalmente se resuelven basándose en las dependencias de construcción (*Build-Depends*) de modo que si por ejemplo se indica allí programa-dev automáticamente el sistema sustituirá las variables ${*shlibs:Depends*} y/o ${*misc:Depends*} por programa (sin el -dev).

En algunas ocasiones (empaquetado de programas hechos en Python, Java) es necesario indicar manualmente las dependencias del programa.

Si se diera este caso lo recomendado es añadirlas después de ${*misc:Depends*} respetando la estructura de comas.

- **Recommends:** se usa para paquetes altamente recomendados y que normalmente se instalan junto al paquete. Algunos administradores de paquetes, por ejemplo *aptitude*, instalan automáticamente los paquetes que se introduzcan en este campo.
- **Suggests:** se usa para paquetes que son similares o útiles cuando el paquete esté instalado**.**
- **Conflicts:** se usa para paquetes que presentarán problemas con este paquete. Ambos no pueden estar instalados en un mismo sistema. Si uno se instala el otro se desinstalará y viceversa.
- **Description:** aquí se describe el programa, esto es lo que se verá cuando se abra el archivo .deb con un programa como *GDebi*. La descripción de un paquete deb se basa en dos partes:
  - ✓ Un título de 60 letras que va directamente después de *Description*:
  - ✓ Un texto de unas 47 letras de largo con un espacio antes de cada línea. Los espacios entre párrafos se hacen mediante un punto.

### 4.7.4. Archivo copyright

El archivo *copyright* proporciona la información de derechos de autor. Generalmente, la información de *copyright* se encuentra en el archivo *COPYING* en el directorio principal del código fuente. En este archivo se incluye información como el nombre del empaquetador y del creador del paquete, la URL de donde se obtuvo el código fuente, la licencia con el año del *Copyright*. Un plantilla del archivo podría ser:

```
This package was debianized by (Tu nombre) <tu_correo_electrónico>

(Fecha)

It was downloaded from: (URL de la web de descarga del programa)

Upstream Author(s): (Nombre(s) y e-mail(s) del (los) autor(es) del programa)

Copyright: Copyright (C) {año(s)} por {autor(es)} {correo(s) electrónico(s)}
```

```
License: (Añade el texto de la licencia aquí. Para las licencias GNU basta
con añadir la cabecera y enlazar al texto completo normalmente alojado en
/usr/share/common-licenses.)
the Ubuntu packaging:
 Copyright (C) (año(s)) por (Tu nombre) <tu_correo_electrónico> (Fecha)
 released under {la licencia que quieras para tu paquete}
```

El script *dh_make* proporciona una plantilla para el archivo *copyright*, con la opción --*copyright* gpl2 se consigue la plantilla para el paquete hello con la licencia GPL-2.

Debes completar la información sobre el lugar donde se puede obtener el código fuente, las condiciones de derechos de autor y la licencia. Las licencias de código libre más comunes son GNU GPL-1, GNU GPL-2, GNU GPL-3, LGPL-2, LGPL-2.1, LGPL-3, GNU FDL-1.2, GNU FDL-1.3, Apache-2.0 o la «*Artistic license*». Debe incluirse el texto completo de la licencia, se introduce el texto de la licencia bajo la cual se desarrolló la aplicación.

### 4.7.5. Archivo rules

El último archivo que vamos a necesitar es el rules. Este, prácticamente, hace todo el trabajo para crear el paquete deb. Es un *Makefile* con instrucciones para compilar e instalar la aplicación, luego crear el archivo .deb a partir de los archivos instalados. También tiene instrucciones para revertir la compilación e instalación eliminando todos los archivos originales y así poder volver a disponer solo del código fuente. A continuación un ejemplo de cómo estará el archivo rules originalmente:

```
#!/usr/bin/make -f
2 # -*- makefile -*-
3 # Sample debian/rules that uses debhelper.
4 # This file was originally written by Joey Hess and Craig Small.
5 # As a special exception, when this file is copied by dh-make into a
6 # dh-make output file, you may use that output file without restriction.
```

```
7 # This special exception was added by Craig Small in version 0.37 of dh-
make.
8
9 # Uncomment this to turn on verbose mode.
10 #export DH_VERBOSE=1
11
12 %:
        dh $@
```

La última línea del archivo rules especifica en caso de que no se modifique que se ejecutarán todos script que provee *debhelper* para lograr una mayor automatización del proceso de construcción de paquete de código fuente.

De interés para la investigación resulta la modificación del archivo rules para la distribución GNU/Linux Nova, esta modificación se dividió en dos partes una para las aplicaciones de lenguajes de programación compilados y otra parte para los interpretados.

#### 4.7.5.1. Aplicaciones de lenguajes interpretados:

Para este tipo de aplicaciones las de lenguajes interpretados el trabajo con el rules es bastante fácil en este caso este archivo no necesita modificación, se deja tal cual se genera inicialmente. Pero en sustitución se creará uno nuevo fichero nombrado *install*, en caso de que se esté trabajando en paquetes grandes que separan como resultado de la compilación múltiples paquetes binarios, se debe especificar un fichero.install por cada uno de los binarios a generar, quedando nombre_del_paquete1.install, nombre_del_paquete2.install, etc. A estos archivos *install* se le incluye la información de los ficheros que se necesitan copiar para el sistema así como la dirección para donde lo harán. El archivo *install* tendrá una línea para cada uno de los archivos a instalar, con el nombre del archivo seguido de un espacio y a continuación el directorio de instalación. La dirección de los nombres de los archivos a copiar es relativa a la carpeta donde se encuentra todo el código fuente, así como la dirección a donde se copiarán, que es relativo a la carpeta temporal que se crea para la generación del paquete binario.

#### 4.7.5.2. Aplicaciones de lenguajes compilados

Este tipo de aplicación genera más trabajo y modificaciones en el archivo rules, como es de interés de nuestra investigación analizar en la distribución GNU/Linux Nova se mostrarán las principales modificaciones que se le realizan al archivo rules en dicha distribución, siempre y cuando la aplicación haya sido construida con la ayuda de las a*utotools* o presenta de una manera u otra los ficheros necesarios que se generan utilizando esta herramientas. A continuación un ejemplo de los principales aspectos que se plasman en el archivo rules de Nova.

```
#!/usr/bin/make -f

package="nombre_paquete"

clean:

        dh_testdir

        dh_clean

        rm –f build

install: build

        dh_clean

        dh_installdirs

        $(MAKE) destdir=$(CURDIR)/debian/package install

build:

        $(MAKE) prefix=$(CURDIR)/debian/tmp

        touch build

binary-indep:



binary-arch:

        dh_testdir -a

        dh_testroot -a
```

```
        dh_strip -a

        dh_compress -a

        dh_fixperms -a

        dh_installdeb -a

        dh_shlibdeps -a

        dh_gencontrol -a

        dh_md5sums -a

        dh_builddeb -a

binary: binary-indep binary-arch
```

A continuación se procede a realizar el análisis del archivo con más detalle. La primera parte que se puede ver es la declaración de algunas variables que facilitarán el trabajo futuro evitando repetir información.

```
package="grabar"
```

En esta sección se pueden exponer las banderas para el compilador y también se pueden indicar las opciones *noopt* para la información de depuración (*debugging*).

Ahora procedemos a ver la sección clean:

```
clean:

        dh_testdir

        dh_clean

        rm –f build
```

Aquí se especifica la limpieza del código lo primero que se hace es:

**dh_testdir:** Este comprueba el directorio antes de construir un paquete de Debian.

Luego se realiza:

**dh_clean:** Leerá debian/clean y eliminará los ficheros ahí listados y eliminará ficheros *-stamp del nivel superior.

Por último lo que se hace es

**rm –f build:** Para borrar todo el contenido del build.

La siguiente sección a analizar será *build*:

```
build:

 ./configure --prefix=/usr

 $(MAKE) prefix =$(pwd)/debian/tmp

 touch build
```

Estos comandos ejecutan el. /configure con las variables adecuadas, luego ejecutan *make* al cual se le pueden especificar las banderas de compilación utilizando las variables o no, que se declararon al inicio y se le puede especificar cuál será la dirección de construcción en el ejemplo se muestra como; prefix =$(pwd)/debian/tmp y por último se hace un registro de la construcción para prevenir compilaciones múltiples erróneas.

La próxima sección que se analizará será *install*:

```
install: build

        dh_clean

        dh_installdirs

        $(MAKE) –prefix=/usr install
```

Dónde se especifica que este *target* depende primeramente de la ejecución del *build*, es por eso que aparece luego de los dos puntos, en esta sección se ejecutan:

**dh_clean:** Leerá debian/clean y eliminará los ficheros ahí listados y eliminará ficheros *-*stamp* del nivel superior.

Luego se realizará lo siguiente:

**dh_installdirs:** Crea subdirectorios en los directorios de construcción del paquete

Por último se realiza $(MAKE) –prefix=/usr install que especificándole la dirección relativa a donde se va instalar los binarios una vez compilado el código fuente, ejecuta el comando *make install*.

Lo próximo que se verá serán unas instrucciones *binary-indep* vacías. Algunos paquetes se crean independientes de una arquitectura específica (como 32 o 64 bits) si no contienen archivos que necesiten de un compilador. La mayoría de los programas Python o de temas de escritorio son ejemplos de esto, sus paquetes terminan siempre en *_all.deb*.

En esta categoría se analizan las aplicaciones de lenguaje por lo que necesitan de un compilador que las procese y el resultado será un código binario diferente para cada plataforma, sus paquetes pueden terminar en *_i386.deb*, *_amd64.deb*, etc… dependiendo de la arquitectura para la cual se compilará. Para estos paquetes dependientes de una arquitectura se usa el campo *binary-arch*:

```
binary-arch: install

 dh_testdir -a

        dh_testroot -a

        dh_strip -a

        dh_compress -a

        dh_fixperms -a

        dh_installdeb -a

        dh_shlibdeps -a

        dh_gencontrol -a

        dh_md5sums -a

        dh_builddeb -a
```

Aquí se ponen en funcionamiento una serie de scripts *debhelper* que crean los paquetes deb. *dh_testdir* y *dh_testroot* hacen algunas comprobaciones de la estructura. *dh_strip* tomará la información de depuración (*debugging symbols*) y la apartará de la aplicación reduciendo drásticamente su tamaño. *dh_compress* ejecuta *gzip* (un compresor de consola) para comprimir alguna documentación. *dh_fixperms* hará ejecutables todos los ficheros en los directorios bin/ y etc/init.d. dh_installdeb marca automáticamente todos los ficheros en etc/ como conffiles. *dh_shlibdeps* añade las bibliotecas dependientes al campo "*Depends: ${shlibs:Depends}*" en el archivo debian/control. *dh_gencontrol* es el que se encarga de generar e instalar el fichero de

control. dh_md5sums genera el fichero DEBIAN/md5sums. Finalmente *dh_builddeb* construye nuestro archivo .deb.

### 4.8. Construcción del paquete fuente.

#### Uso del comando dpkg

Para finalizar y obtener el paquete de código fuente se utiliza el comando *dpkg*-source –b /dirección_código. Este permite construir un paquete fuente, el primer argumento no opción se toma como el nombre del directorio que contiene el árbol de fuentes debianizada (es decir, con un sub-directorio debian y tal vez los cambios en los archivos originales). Dependiendo del formato de paquete fuente utilizado para construir el paquete, los parámetros adicionales podrían ser aceptados.

Esto generará el paquete de código fuente listo para luego compilarse y que se generen los paquetes binarios para la distribución cubana GNU/Linux Nova.