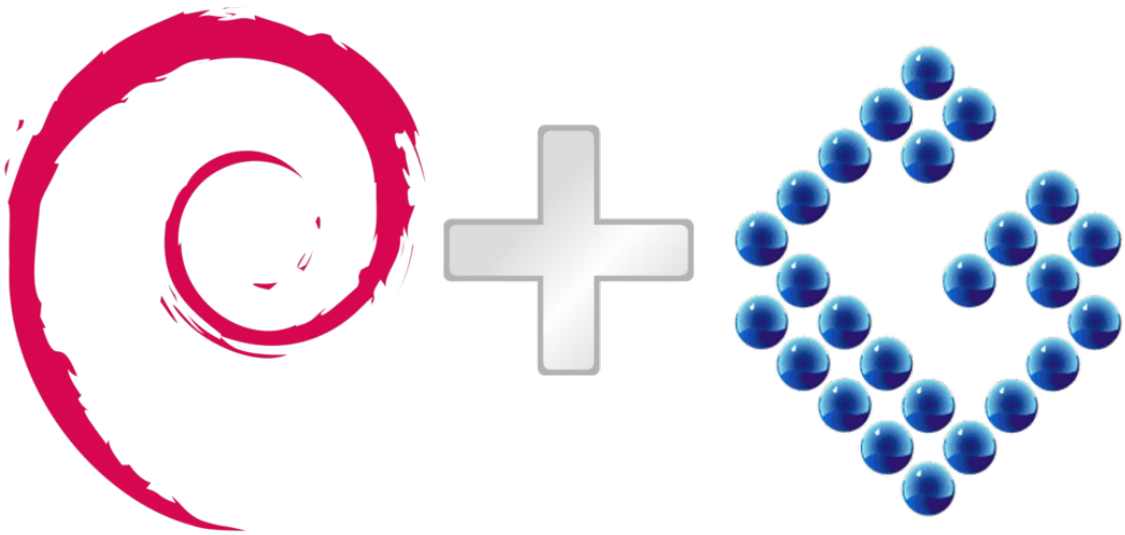


Trabajo Práctico Final:

Debianización de Gobstones

Laboratorio de Redes y Sistemas Operativos

Profesor: Jose Luis Di Biase

Integrantes:

Sergio Ivanic,
Nazareno Castro,
Rodrigo Gonzalez.

Introducción:

Gobstones es una herramienta de programación orientada al aprendisaje con elementos básicos y concretos. Fue desarrollada por la Universidad Nacional de Quilmes.
El principal objetivo de Gobstones es acercar a los estudiantes novatos a los conceptos mas importannes de programación. Esta herramienta esta compuesta por un programa desarrollado en Python y un lenguaje el cual tambien se llama Gobstones.

Por otro lado, las distribuciones de linux tienen varias formas de agregar programas, hay dos que son las mas comunes, instalar un paquete via `sudo apt-get install <<program-name>>` ; obtener de algun lugar el archivo <<program-name>>.deb y ejecutarlo con algun gestor de software.

Objetivos:

Gobstones es una herramienta utilizada por todos los estudiantes que comienzan sus estudios en las carreras de programación de la UNQ. De esta manera podemos ver que todos los estudiantes van a tener que "*Instalarlo*".

Existe una idea general de que el sofware libre es mejor. Para aportar a esta idea de manera concreta, creemos que si los estudiantes que recien ingresan a las carreras de programación, pueden instalar Gobstones de una manera mas "Windows", o mas sensilla, seria mas facil invitarlos a usar software libre.
Para poder lograr esto necesitamos:

1. Entender el mecanismo de generar paquetes binarios de debian, y de publicar nuestro sofware en un repositorio desde donde se pueda instalar facilmente.
2. Crear un paquete debian para poder instalar Gobstones con un solo archivo.
3. Subir Gobstones a un Personal Package Archive (PPA) para poder agregar el repositorio a cualquier linux y poder instalar Gobstones con un simple comando.

Source Package vs. Binary Package

Antes que nada, deberíamos saber que son o que contienen los paquetes sobre los que estamos tratando.

Los source packages, o paquetes fuente, contienen todos los archivos necesarios para compilar o construir el software deseado.

Los binary packages, o paquetes binarios, son paquetes de aplicación que contienen especificamente archivos ejecutables preconstruidos. Para correr estos ejecutables, es necesario descomprimir el paquete; generalmente hay sistemas de gestión de paquetes que se encargan de realizar este trabajo (por ejemplo apt, dpkg).

Hay varias maneras muy sencillas de generar paquetes binarios, como paquetes .deb para Debian, .rpm para RedHat, etc. Pero, aunque paresca una alternativa agradable para trabajar, no es solamente incompatible para subir a un repositorio, donde se requieren paquetes fuente, si no que tambien es inferior. Esto es debido a que las distribuciones binarias:

- no proveen una manera unificada de obtener un software (como lo son los repositios APT para los paquetes fuente – apt-get).
- dificultan (si no imposibilitan) reconstruir un paquete binario debido a la falta de estandarización en el proceso de construcción del paquete y descripciones de las dependencias. A menudo los autores “upstream” (los usuarios que publican su software) suben varias instrucciones en archivos README, las cuales pueden variar de paquete a paquete y no hay garantias de que esten totalmente completas; haciendo que el proceso de reconstruccion del paquete sea bastante frágil.
- están compiladas para una distribución particular de nuesto sistema UNIX.

Gobstones en un Debian Package.

Al momento de querer crear nuestro paquete binario es necesario entender que es lo que estamos queriendo hacer.

Si vamos a las principales paginas de consulta como por ejemplo:

- https://wiki.debian.org/es/IntroDebianPackaging
- https://wiki.debian.org/HowToPackageForDebian

podemos encontrar mucha información sobre como crear un un paquete debian. El problema con estos links es que nos dan mucha información que en primera medida no es necesaria.

Estas paginas nos invitan a crear paquetes de manera que puedan ser aceptados por la comunidad de debian y puedan ser agregados en un futuro a los repositorios de Debian o Ubuntu por ejemplo.

Tomando como referencia:

- http://askubuntu.com/questions/90764/how-do-i-create-a-deb-package-for-a-single-python-script/91616#91616
- http://stackoverflow.com/questions/17401381/debianzing-a-python-program-to-get-a-deb

tenemos que, para que nuestra aplicación sea debianizable necesitamos respetar el siguiente formato de carpetas:

— gobstones
|— DEBIAN/control
|— DEBIAN/copyright
|— opt/gobstones
|— usr/bin/gonstones
|— usr/share/applications/gobstones.desktop
|— usr/share/icons/hicolor/16x16/app/gobstones.png
|— usr/share/icons/hicolor/32x32/app/gobstones.png
|— usr/share/icons/hicolor/48x48/app/gobstones.png
|— usr/share/icons/hicolor/128x128/app/gobstones.png
|— usr/share/icons/hicolor/256x256/app/gobstones.png

En la carpeta DEBIAN (con mayuscula) se agregan los archivos necesarios para el empaquetamiento de nuestro programa.
DEBIAN/control

Package: gobstones – nombre del paquete que estamos construyendo.
Version: 1.5.3 – version.
Architecture: all – arquitectura, se puede especificar x86 x64.
Maintainer: CPI UNQ<unq.cpi.debian@gmail.com> – mail de los mantenedor del paquete.
Installed-Size: 6000 – tamaño en Kbs.
Depends: python, python-qt4 – programas que necesita nuestro paquete para funcionar.
Priority: optional – La importancia del paquete.
Homepage: http://www.gobstones.org/ – pagina de los desarolladores.
Description: IDE basado en python para el uso del lenguaje de programacion Gobstones.

Este archivo es que el encargado de determinar como se va a instalar nuestro programa. Cuando se ejecute el .deb

DEBIAN/copyright. Licencias del programa.

opt/gobstones es la carpeta del programa, dentro se encuentran todos los archivos del programa. Una pequeña aclaración, en nuestro caso estamos queriendo empaquetar un programa desarrollado en un lenguaje no compilado como es python. Si nuestro programa necesitaria ser compilado, en esta carpeta deberia estar nuestro programa compilado. En este caso tenemos un archivo que al ser ejecutado con python abre el programa.

usr/bin/gobstones

```
#!/bin/sh
exec python /opt/gobstones/pygobstones.py "$@"
```

Es el archivo que refiere al comando por consola con el que se ejecuta nuestro programa. Gracias a que este archivo esta en la carpeta usr/bin/, en la consola podemos escribir gobstones (o el nombre de cualquier archivo que tengamos en esa carpeta) y se va a ejecutar.
En este caso especial, al ejecutar nuestro archivo, ejecuta con python la entrada para nuestro programa.

usr/share/applications/gobstones.desktop

```
[Desktop Entry]
Version= 1.5.3
Type=Application
Name=Gobstones
GenericName=Text Editor
Comment=IDE para el lenguaje de Programación Gobstones
Exec=/usr/bin/gobstones %F
Terminal=false
MimeType=text/plain;
Icon=gobstones
```
— nombre que va a buscar a la carpeta icons que definimos mas adelante
```
Categories=Development;
```
— categoria en el arbol de aplicaciones
```
StartupNotify=true
Actions=Window;Document;
```
— De que manera se puede ejecutar este programa.

```
[Desktop Action Window]
Name=New Window
Exec=/usr/bin/gobstones -n
OnlyShowIn=Unity;
```

Este archivo es la entrada de escritorio de nuestro programa, sirve para que aparesca en nuestra lista de aplicaciones.

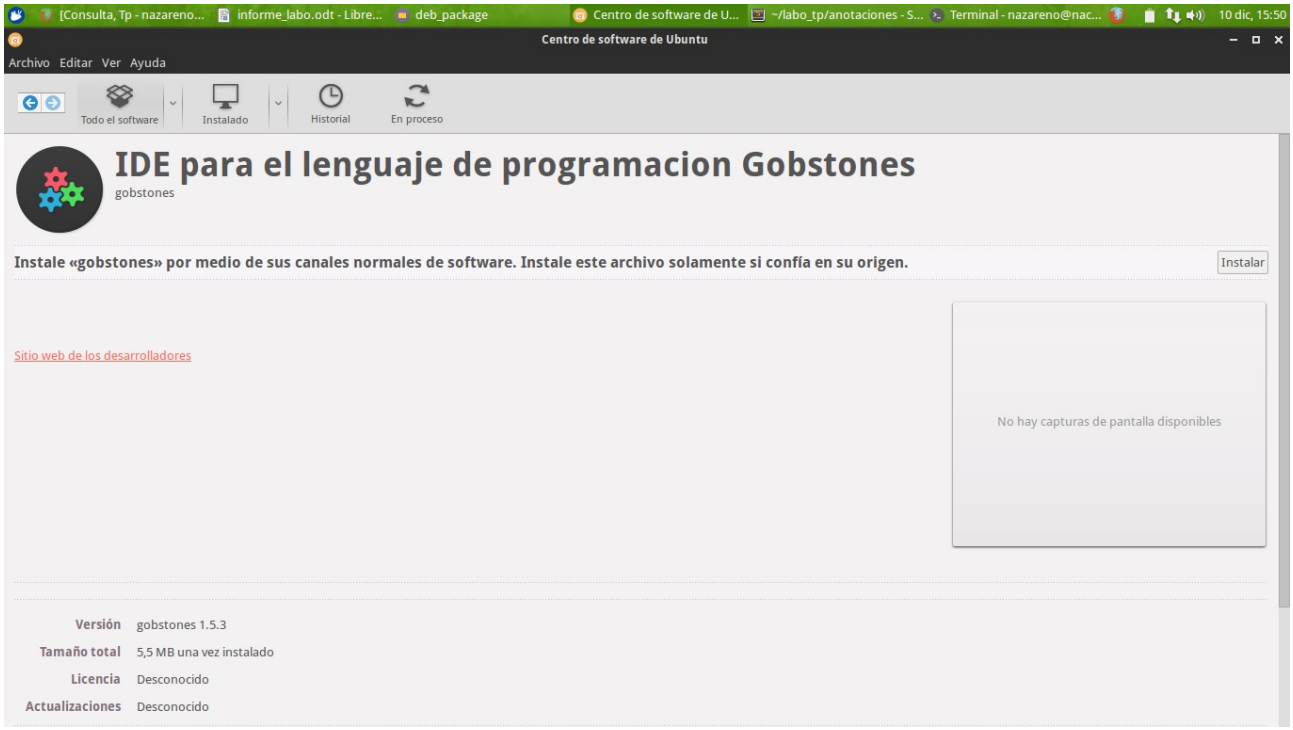
usr/share/icons/hicolor son todas las carpetas con los archivos png del mismo tamaño de la carpeta que sirve para darle imagen a los iconos.

Una vez que tenemos todo esto, respetando la estructura. Necesitamos crear nuestro paquete debian.

Con el comando `fakeroot dpkg-deb --build gobstones ,` donde gobstones es la carpeta donde estan todos los archivos antes mencionados.
¿Que estamos haciendo en este comando?

- `Fakeroot` es un programa que ejecuta una orden en un entorno que falsea privilegios de superusuario para la manipulacion de ficheros. Se necesita este comando porque sino el software de instalacion del paquete nos va alertar sobre la seguridad del paquete.
- `dpkg-deb –build` Crea un archivo de Debian a partir del árbol del sistema de ficheros ubicado en el directorio. El directorio debe tener un subdirectorio DEBIAN, el cual contiene los ficheros de información tales como el fichero de control
- `gobstones` carpeta donde esta nuestro arbol de archivos antes explicado.

Con este comando deberia haberse creado un archivo gobstones.deb en el directorio donde esta la carpeta gobstones. Si lo ejecutamos y tenemos por ejemplo el Centro de Software de Ubuntu aparecera algo como esto



Obviamente para instalar presionamos **instalar,** nos pide nuestra contraseña y tenemos instalado gobstones.

Distintas formas que encaramos para poder entender como crear el paquete.

Primero intentamos usando programas que te crean una carpeta emulando el root del sistema para el que se quiere hacer el paquete. De esta manera podiamos crear el paquete y “probarlo”. El problema fue que no era lo que necesitabamos, no teniamos el paquete, por lo tanto no era el camino a seguir.

Despues intentamos usar un debianizador, un programa que crea el arbol de archivos y carpetas necesarios para luego crear el paquete. Fue un acercamiento a la solucion final pero tenia archivos que no eran necesarios, por otro lado no nos daba una explicacion de como configurar cada cosa.

Por ultimo decidimos tomar un programa tambien desarrollado en python y que este debianizado. Elegimos el Sublime, de ahi pudimos entender como se podia acceder al programa desde la lista de aplicaciones, como setearle una imagen al icono, como hacer para ejecutar nuestro programa desde la consola.Nos faltanba poder crear el archivo .deb. Encontramos que lo que era necesario para que el archivo .deb se instale era el archivo DEBIAN/control (anteriormente copiamos unos links que llevan a las paginas desde donde recolectamos información para poder armar este archivo y que comando nos crea el paquete con formato debian).

Gobstones en un Personal Package Archive (PPA)

¿Qué es PPA?

PPA, llamado así por sus siglas en inglés Personal Package Archive, es un repositorio de software para sistemas de tipo UNIX dedicado a la subida exclusiva de source packages. Este repositorio no pertenece a los repositorios oficiales. En otras palabras, para poder descargar e instalar un software ubicado en el repositorio primero se debe agregar el mismo a la lista de repositorios en nuestro sistema, y luego finalmente gestionar el mismo como cualquier software ubicado en los repositorios oficiales.
Estos paquetes, luego de ser subidos al repositorio, serán generados en forma de paquetes binarios y publicados como un repositorio APT por Launchpad, para que puedan ser accedidos libremente por cualquier usuario.
APT, o Advanced Package Tool, es un software libre dedicado a la gestión de software en distribuciones Debian y sus variantes. APT simplifica el proceso de manipulación de software automatizando la bajada, configuración e instalación del mismo, ya sea mediante archivos precompilados o compilando código fuente.

¿Por qué un repositorio PPA?

El repositorio PPA no solo ofrece la posibilidad de almacenar nuestro software en la nube y encargarse de generar los paquete binarios, si no que también permite llevar un versionado de nuestro software en el mismo repositorio.
Esto quiere decir que al modificar nuestro codigo fuente, y lógicamente su versión, solo se debe crear el source package y subirlo al repositorio para que esté disponible para los usuarios de nuestro

software. El usuario, de la misma manera que maneja software ubicado en repositorios oficiales de Ubuntu, mediante un comando en consola tiene la posibilidad de actualizar el software descargado.

1. Herramientas necesarias

1.1 DPKG (Debian Package Gestor)

sudo apt-get install dpkg

Esta herramienta sirve para gestionar paquetes Debian en el sistema.

1.2 Fakeroot

sudo apt-get install fakeroot

Explicado anteriormente.

1.3 OpenSSH

sudo apt-get install openssh-client

Esta herramienta sirve para generar claves SSH.

1.4 DPUT

sudo apt-get install dput

Esta herramienta nos permite subir nuestros paquetes al repositorio PPA.

1.5 dh_make

sudo apt-get install dh-make

Esta herramienta sirve para convertir archivos de código fuente en paquetes de código fuente de Debian.

2. Organización de los archivos

Los archivos deben estar organizados dentro de una carpeta nombrada <paquete>-<versión>, por ejemplo 'gobstones-0.8'. Dentro de la misma deben estar los archivos que se instalaran, o que finalmente quedaran en el sistema del usuario.

La estructura de los archivos debería ser muy parecida, si no igual, al utilizado para crear el paquete binario (.deb). Un ejemplo de esta estructura sería:

- gobstones-0.8

    |--- opt / gobstones / *

    |--- usr / share / *

    |--- usr / bin / *

3. Generando los archivos necesarios para el paquete source

Dentro de la carpeta <package>-<version> ejecutamos 'dh_make –createorig'. Esto generará los

archivos base y necesarios para poder empaquetar nuestro software.

Cuando ejecutemos este comando, nos preguntara el tipo de paquete que queremos crear

'Type of package: single binary, indep binary, multiple binary, library, kernel module, kernel patch? [s/i/m/l/k/n] '

Elegimos single binary con la letra 's' y nos mostrara la informacion del paquete

Maintainer name : usuario

Email-Address   : ***@una_compania.com

Date   : Tue, 08 Dec 2015 19:21:32 -0300

Package Name   : gobstones

Version   : 0.8

License   : blank

Type of Package : Single

Si esta todo correcto, apretar 'enter' para finalizar la creación.

Esto creará una carpeta dentro de <paquete>-<versión> llamada 'debian' (Distinta a la que teniamos al momento de generar el paquete binario, la cual era 'DEBIAN', con mayúscula). Dentro la misma podemos ver que se crearon varios archivos, los que nos interesan de estos son: changelog y control.

Además podemos ver que en la carpeta superior del directorio <paquete>-<versión> se generó un archivo <paquete>_<versión>.orig.tar.xz.

4. Modificando los archivos generados

4.1 Changelog

Al abrir el archivo veremos algo como lo siguiente

<paquete> (<versión>-<versión-en-distro>) <distro>; urgency=low

* <comentario>

 -- <usuario> <<***@***>> Mon, 07 Dec 2015 18:39:37 -0300

Las cosas que se generaron automáticamente son:

<paquete>: el nombre del paquete;

<versión>: la versión que pusimos en el directorio <paquete>-<versión>;

<distro>: es la distribución en la que estamos trabajando en Ubuntu, podriamos cambiarlo por otra como 'wily' (Ubuntu 15.10), 'trusty' (Ubuntu 14.04), 'vivid' (Ubuntu 15.04), etc.;

<comentario>: un comentario auto-generado que deberiamos cambiar por algún comentario de la versión;

<usuario>: el usuario upstream;

<***@***>: el mail del usuario upstream.

Por convención, se agrega <versión-en-distro> luego de la <versión> el cual indica la versión en Debian y la versión en Ubuntu, de la siguiente manera <versión-debian>ubuntu<versión-ubuntu>. Por ejemplo , 0ubuntu1, esto sería la versión cero de debian (ya que no hay versión en debian) y la versión 1 de ubuntu.

4.2 Control

Es el mismo que se explicó en la creación del paquete binario, con la diferencia que es obligatorio indicar algo en el campo 'Section:', en nuestro caso 'devel'.

Otras categorías disponibles a la fecha son: admin, cli-mono, comm, database, debug, devel, doc, editors, education, electronics, embedded, fonts, games, gnome, gnu-r, gnustep, graphics, hamradio, haskell, httpd, interpreters, introspection, java, kde, kernel, libdevel, libs, lisp, localization, mail, math, metapackages, misc, net, news, ocaml, oldlibs, otherosfs, perl, php, python, ruby, science, shells, sound, tasks, tex, text, utils, vcs, video, web, x11, xfce, zope.

5. El archivo <paquete>.install

Dentro de la carpeta 'debian/' se crea un archivo llamado <paquete>.install, por ejemplo 'gobstones.install'. Este archivo tendrá las directivas de donde copiar (en el caso de nuestro proyecto, fue solo copiar) los archivos que contiene nuestro paquete a los ficheros del sistema del usuario. La directiva '%: dh $@' en el archivo rules se encarga de ejecutar la acción de copiado, descripta en el archivo <paquete>.install.

Entonces el archivo <paquete>.install debería verse parecido a lo siguiente:

fichero/dentro/del/paquete fichero/en/el/sistema/del/usuario

archivo/genérico/en/paquete/foo.py fichero/en/el/sistema/del/usuario/

Para mas información de este comando podemos ver consultarlo escribiendo en la consola 'man dh_install'.

6. Utilización de una llave GPG

6.1 Generar llave

Generamos una llave GPG, e ingresamos una frase clave para la misma. Los pasos paa crear la llave se vieron en clase (para saber como crear una https://fedoraproject.org/wiki/Creating_GPG_Keys/es#Creando_llaves_GPG).

Y la publicamos en los servidores Ubuntu mediante el comando

gpg --keyserver keyserver.ubuntu.com --send-keys <key id>

Nota: para saber el id de la llave podemos escribir 'gpg --fingerprint' en la consola y nos mostrará todas nuestras llaves, el id en nuestro caso sería '1234567A' en la sección 'pub = 1234A/1234567A'.

Este proceso puede demorar.

6.2 Subiendo nuestra llave a Launchpad

Luego debemos subir nuestra llave generada a nuestra cuenta en Launchpad. Si no tienen una,

deerían crearla en https://launchpad.net/.

Una vez en nuestra cuenta vamos a la sección de llaves OpenPGP y pegamos nuestra huella de clave en la sección que dice 'Fingerprint'. La huella de clave esta debajo de la key id cuando la consultamos anteriormente. Esta luce algo como '27E0 7815 B47C 0397 90D5 8589 27D9 A27B F3F9 6058'.

Luego de esto, Launchpad nos enviará un mail cifrado a la cuenta con la que nos registramos para verificar la llave que acabamos de ingresar desde la web.

6.3 (OPCIONAL) Herramienta para desencriptar le mail de verificación

Si no tenemos manera de desencriptar el mail desde el navegador, entonces se puede instalar un plugin llamado 'enigmail' para la herramienta 'thunderbird' mediante el comando 'sudo apt-get install enigmail'.

Simplemente debemos loguearnos con nuestro mail en el gestor, ingresar en el mail encriptado. En este paso nos pedirá la frase que ingresamos al momento de crear la llave GPG.

7. Configurar el SHELL

Ahora debemos indicarle a nuestro sistema quien somos para que cuando creemos el paquete contenga nuestros datos. Para eso, vamos al archivo ubicado en nuestra carpeta personal ~/.bashrc y pegamos lo siguiente debajo de todo del archivo

export DEBFULLNAME="<Nombre apellido>"

export DEBEMAIL="<dirección_de@mail.com>"

8. Generamos el paquete source

Nos dirigimos al fichero <paquete>-<versión> y ejecutamos el comando 'dpkg-buildpackage -S -sa -k<key ID> -rfakeroot'. Con la opción -k<key id> estamos firmando el paquete con nuestra llave personal, más específicamente los archivos '_source.change' y '.dsc'. Este paso es obligatorio para subirlo al repositorio PPA.

De esta manera ya tendremos nuestro paquete source generado en el fichero superior.

8.1 (OPCIONAL) Testear nuestro paquete localmente antes de subirlo

Para asegurarnos que estamos subiendo un paquete totalmente funcional, es decir, que instale correctamente y que se adecue a nuestras espectativas; podemos utilizar la herramienta pbuilder. La podemos conseguir mediante el comando 'sudo apt-get install pbuilder'.

Primero debemos crear un ambiente de testeo totalmente limpio para nuestro paquete. Para eso ejecutamos 'pbuilder-dist <distro> create'. Donde <distro> es una distribución de nuestra sistema UNIX, como se describió anteriormente (wily, vivid, etc.).

Este proceso puede demorar bastante debido a que se descargarán los archivos necesarios para una instalación mínima.

Luego nos dirigimos al directorio que posee el paquete source generado y ejecutamos 'pbuilder-dist <distro> build <paquete>_<versión>.dsc'. Este paso generará un paquete binario para la

distribución que específicamos, asi que tambíen puede demorar.

Una vez terminado este proceso vamos al directorio en nuestra carpeta personal '~/pbuilder/<distro>_result/' y ejecutamos el comando sudo dpkg -i<paquete>_<versión>.deb. Esto instalará el paquete en nuestro sistema de la misma manera que si la estuviéramos instalando desde el repositorio.

9. Subiendo nuestro paquete source al repositorio

En nuestra cuante de Launchpad, mas específicamente en la sección de Personal Package Archive, elegimos la opción 'create a new PPA', donde nos pedirá el nombre del repositorio, el nombre con el que será visible y una descripción del mismo.

Una vez creado vamos al fichero que contiene nuestro paquete generado, que debería contener el archivo '_source.changes'. En el mismo directorio ejecutamos el comando 'dput ppa:<usuario-launchpad>/<nombre-ppa> <source.changes>'.

Donde <usuario-launchpad> es nuestro usuario de Launchpad (launchpad.net/~<usuario-launchpad>); <nombre-ppa> el nombre de nuestro repositorio PPA; <source.changes> el archivo '_source.changes'.

Pendientes:

Agregar al ppa, el source compatible para mas versiones de linux.

Verificar que si se sube una nueva version al repositorio ppa, se puede actualzar desde el comando sudo apt-get update.

Si es posible, agregar a los repositorios de Ubuntu (por ejemplo) Gobstones para que ya no sea necesario instalarlo de forma externa.

Conclusión:

Empaquetando Gobstones pudimos entender el proceso de empaquetado con una apicación sensilla ya que no es compilable. Por otro lado, sirve como aporte para que los que recien comienzan puedan instalar gobstones en linux ya sea, de una manera mas parecida a lo que es en linux, o instalarlo con un ppa, y de esta manera el programa queda instalado correctamente y es ejecutable de manera amigable.