ACTIVIDAD 17. Distribuir la aplicación

En esta actividad vamos crear un paquete para distribuir la aplicación.

Paso 1 – Instalando los paquetes necesarios

```
$sudo apt-get install build-essential automake gnupg lintian fakeroot pbuilder devscripts debhelper dh-make
```

Paso 2 – Crear un directorio de trabajo

Crearemos **un directorio** que se suele nombrar en inglés como "*sandbox*" . Mediante terminal:

```
$ mkdir sandbox && cd sandbox
```

Paso 3 – Nombrar correctamente el paquete

El nombre de directorio que contiene el programa debe cumplir cierta regla:

Nombre de programa + guión + versión

Por ejemplo un *programa* en la versión *0.1.1*, el nombre para el directorio sería *programa-0.1.1*

```
$ mkdir programa-0.1.1
```

Se copian los **archivos** de programa dentro.

Paso 4 – Comprimir el directorio del programa

Para pasos sucesivos necesitamos **comprimir** el directorio del programa con el siguiente comando:

```
$ tar czf programa-0.1.32.tar.gz programa-0.1.32
```

Paso 5 – Respaldar el programa original

Es importante hacer **una copia** del código fuente original, por si erramos en el proceso.
Normalmente, el **código de desarrollo** que procede de otra fuente superior a la original, en inglés se le llama "*upstream*". Por contra, el término "*downstream*" es el empaquetador de software que trabaja a partir de otro desarrollo superior. Por ejemplo: la **mayoría** del software de Ubuntu viene directamente de Debian, por lo que Debian es considerado *upstream* de Ubuntu. la mayoría de los paquetes de Debian vienen a su vez de otros proyectos, como GNU, Linux, KDE y GNOME, que son a su vez upstream de Debian. El *upstream* suele tener el siguiente formato: *<nombrepaquete>_<version>.orig.tar.gz*

Se utiliza un **guión bajo** en el nombre y que se comprime el directorio en formato **gzip**. El guión bajo es importante porque las herramientas de empaquetado buscarán un nombre de archivo con ese formato exacto. En nuestro ejemplo sería: *programa_0.1.1.orig.tar.gz*

```
$ cp programa-0.1.32.tar.gz programa_0.1.32.orig.tar.gz
```

Se pueden construir dos tipos de paquetes. Un **paquete fuente** contiene el código que puedes compilar en un programa. Un **paquete binario** contiene sólo el programa terminado.

Paso 6 – Crear el directorio debian

El directorio *debian* es donde se almacenará toda la **información de empaquetado**. Con ello conseguimos separar la parte de información de empaquetado de la parte de los archivos del programa fuente. Utilizaremos la herramienta **dh_make** para que haga el trabajo por nosotros.

Con dh_make tenemos los siguientes parámetros de configuración:

- *-c, –copyright <licencia>*

Usa el tipo de <licencia> para el archivo de copyright. <licencia> puede ser gpl, gpl2, gpl3, lgpl, lgpl2 lgpl3, artistic, apache or bsd.

- *-e, –email <dirección>*

Usa <dirección> el correo electrónico del campo Maintainer (desarrollador) para el archivo debian/control.

- -l, –library

Establece automáticamente la clase del paquete a biblioteca, saltando la pregunta.

- -s, –single

Establece automáticamente la clase del paquete binario único, saltando la pregunta.

- -i, –indep

Establece automáticamente la clase del paquete a independiente de la arquitectura, saltando la pregunta.

- -m, –multi

Establece automáticamente la clase del paquete a binario múltiple, saltando la pregunta.

- DEBEMAIL

Dirección de correo electrónico para usar en las entradas de control y changelog.

- DEBFULLNAME

Tu nombre, por ejemplo, “Pepe Lui” que aparecerá en las entradas de control y changelog.

- EMAIL

Dirección de correo electrónico para usar en las entradas de control y changelog, solo usado si DEBMAIL no esta establecido.

Debian usa el término desarrollador («maintainer») para la persona que hace paquetes, autor original («upstream author») para la persona que hizo el programa, y desarrollador original («upstream maintainer») para la persona que actualmente mantiene el programa fuera de Debian. Generalmente el autor y el desarrollador fuente son la misma persona.

Existen **algunas opciones más** que puedes ver con *man dh_make* pero generalmente estas suelen ser todas las necesarias para un empaquetado básico.

Nos vamos al directorio del **programa fuente:**

```
$ cd programa-0.1.1
```

Exportamos variables:

```
$ export DEBEMAIL=programa@gmail.com DEBFULLNAME="Pepe Lui"
```

Y ejecutamos el comando de dh_make:

```
$dh_make -e programa@gmail.com -c gpl3 -s
```

Nota*: es necesario poner dos veces el correo electrónico (tanto en DEBEMAIL como en la opción -e) ya que cada una tiene una utilidad distinta y el correo puede ser diferente. Pero si usas el mismo email, puedes usar la variable DEBEMAIL para ahorrartelo por ejemplo:*

```
#dh_make -e $DEBEMAIL -c gpl3 -s
```

Si da error prueba lo siguiente:

```
$dh_make -createorig
```

Ahora ya tendremos creado el directorio *debian/*

Paso 7 – Eliminar plantillas y editar archivos de empaquetado

En el directorio *debian*, se crean por defecto archivos de plantilla con extensión ***.ex***. En programas simples **no suelen ser necesarios** y podemos eliminarlos con los siguientes comandos:

```
$cd debian && rm *.ex *.EX
```

Aún **quedarán** algunos archivos adicionales en el directorio *debian* que pueden ser innecesarios:

- *README.Debian:* archivo de documentación para el usuario final de Debian y *Ubuntu*, algunas veces llamado simplemente README. No confundir con README.source, que es (si esta presente) la documentación para aquellos interesados en modificar el paquete fuente.
- *dirs:* Usado por dh_installdirs para crear los directorios necesarios.
- *docs:* Usado por dh_installdocs para instalar la documentación del programa.
- *info*: Usado por dh_installinfo para instalar la información de archivo.

Los archivos que realmente nos pueden ser importantes (para construir un paquete básico) son los siguientes archivos: *changelog, compat, control, copyright, y rules* en el directorio *debian*.

Paso 8 – Configurando archivo changelog

El archivo changelog es una lista de cambios hechos en el programa para cada versión. Tiene un **formato específico** en el que se escriben el nombre de paquete, version, distribución, cambios y quien hizo dichos cambios en un determinado momento. Además si tenemos una llave GPG, podemos asegurar que el nombre y email que aparecen en el changelog son los mismos que los dados por la llave GPG.

Una posible **plantilla** de changelog sería:

paquete (version) distribución; urgency=urgencia
  * *detalles de cambios*
  * *más detalles de cambios*
  * *incluso más detalles de cambios*
 -- *nombre del mantenedor < correo electrónico >[dos espacios] fecha*

El **formato** (especialmente la fecha) es importante. La fecha la puedes conseguir con usando el comando **date -R**.
Los puntos menores se indican con un guión "-", mientras que los puntos con mas importancia usan el asterisco "*".

Para nuestro paquete de ejemplo programa sería:

programa (0.1.1-0jessy1) jaunty; urgency=low

  * *Nueva liberación corrigiendo muchos fallos y mejoras.*

 -- *Pepe Lui < programa@gmail.com> Wed, 27 Aug 2009 22:38:49 -0700*

Se ha añadido *-0ubuntu1* a la versión. Eso es porque es la **versión de la distribución** que se usa para empaquetar actualizaciones.

Ubuntu y Debian tienen **ligeras diferencias** en las plantillas de empaquetado de versiones para evitar conflictos con los paquetes que proceden de la misma versión de código fuente. Si por ejemplo un paquete de Debian ha sido cambiado en Ubuntu, se añadirá *ubuntuX* (donde X es el

número de revisión de Ubuntu) al final de la versión de Debian.

Si la **versión** del paquete de Debian *programa 0.1.1-1* es cambiada por Ubuntu, el nombre de versión pasaría a ser *programa 0.1.1-1ubuntu1*. Pero si el paquete para la aplicación no existe en Debian, entonces la revisión de Debian seria cero, por tanto *programa 0.1.1-0ubuntu1*.

Por otro lado el comando **dch** (también conocido como *debchange,* del paquete devscripts) puede usarse para editar el changelog y actualizará la fecha automáticamente:

```
$dch -i
```

Es **aconsejable** ejecutar dicho comando estando en el directorio de fuentes (y no un subdirectorio) ya que el programa abrirá el changelog para modificarlo y crear una nueva línea. Por su parte, el parámetro "-i" incrementará el número de versión del paquete.La **utilidad** del comando radica principalmente en controlar los cambios de una versión a otra.

Por ejemplo, al *debianizar* el programa *programa 0.1.1* en el changelog nos pondrá la versión "*0.1.1-1"* indicando que es la primera versión pero al hacer "dhc -i" pasaría a ser la "*0.1.1-2"*.

Paso 9 – Configurando archivo control

El archivo de control contiene información para los gestores de paquetes (como *apt-get*, *synaptic...),* tiempo de compilación, información del mantenedor y mucho más. Un ejemplo para nuestro programa:

Source: programa
Section: devel
Priority: optional
Maintainer: PEPE LUI< programa@gmail.com>
XSBC-Original-Maintainer: PEPE LUI< programa@gmail.com>
Standards-Version: 3.7.3
Build-Depends: debhelper (>= 5)
Homepage: http://www.laquesea.es

Package: programa
Architecture: any
Depends: ${shlibs:Depends}
Description: El clásico paquete de prueba para un ejemplo. El paquete programa, es el paquete de prueba para debianizar software.

El primer párrafo da información sobre el código. A continuación se detallan el resto de líneas:

- **Source**: el nombre del paquete de código fuente.
- **Section**: los repositorios apt son divididos en secciones para facilitar la navegación y categorización del software. Algunos ejemplos son: “main/x11″ o “universe/web”.
- **Priority**: establece la importancia del paquete para los usuarios. Los valores pueden ser:
    - **Required** – los paquetes son esenciales para que el sistema funcione correctamente. Si se eliminan es probable que el sistema se dañe de una manera irrecuperable.
    - **Important** – el conjunto mínimo de paquetes para que el sistema sea usable. Eliminando estos paquetes no producira un daño irreparable en el sistema, pero generalmente son considerados como herramientas importantes sin las que cualquier instalación de Linux estaría incompleta.
    - **Standard** – autoexplicatorio.
    - **Optional** – esta categoría es para paquetes no requeridos o para la mayor parte de los paquetes. Sin embargo, estos paquetes no deben entran en conflicto entre sí.
    - **Extra** – los paquetes que pueden entrar en conflicto con otros paquetes en una de las categorías superiores. Tambien es usado para paquetes especiables que solo es útil para personas que ya conocen el propósito del paquete.
- **Maintainer**: el nombre del mantenedor del paquete y su dirección de email.
- **Standards-Version**: la versión de la política de Debian a la que el paquete se adhiere. Tan fácil como entrar la versión actual con apt-cache show debian-policy | grep Version.
- **Build-Depends**: uno de los campos mas importantes y a menudo la causa más frecuente de fallos en el código fuente, esta línea lista los paquetes binarios (con la versión si es necesario) que necesitan ser instalados para llevar a cabo la creación del paquete binario a partir del código fuente. Los paquetes que son esencialmente requeridos por build-essential no es necesario que sean incluidos.
- **Homepage**: una URL donde se encuentra más información del software.

El segundo párrafo es para el **paquete binario** que se construirá a partir del código fuente. Si existen múltiples paquetes binarios que son construidos desde el código fuente, habrá una sección para cada uno. La explicación para cada línea:

- **Package**: el nombre del paquete binario. En muchas ocasiones para programas simples, el paquete de código y el binario son idénticos.
- **Architecture**: las arquitecturas para las que el paquete binario sera construido. Algunos ejemplos son:
    - **all** – el código fuente no es dependiente de la arquitectura. Los programas que usan

Python o otros lenguajes interpretados deberían usar esta opción. El paquete resultante terminará con _all.deb.

- **any** – El código fuente es dependiente de la arquitectura y debería compilar en todas las arquitecturas soportadas. Habrá un archivo .deb para cada arquitectura (por ejemplo _i386.deb
- Pueden listarse un **subconjunto** de arquitecturas (i386, amd64, ppc, etc.) para indicar que código fuente es dependiente de la arquitectura y los que no funcionara para todas las arquitecturas soportadas por Ubuntu.

- **Depends**: La lista de paquetes de la que el paquete binario depende para su funcionalidad. Para el ejemplo, vemos ${shlibs:Depends}, que es una variable usada por dpkg-shlibdeps para añadir los paquetes de bibliotecas compartidos necesarios por los binarios a Depends:. En la página del manual para dpkg-source(1) and dpkg-shlibdeps(1) hay más información.
- **Recommends**: usado por los paquetes que son altamente recomendados y normalmente son instalados con el paquete. Algunos gestores de paquetes, más concretamente aptitude, automáticamente instalan los paquetes recomendados.
- **Suggests**: usado por paquete que son similares o útiles cuando el paquete es instalado.
- **Conflicts**: usado por paquetes que entrarán en conflicto con este paquete. Ambos no pueden ser instalados al mismo tiempo. Si uno de ellos es instalado, el otro será desinstalado.
- **Description**: la descripción corta (unos 60 caracteres y debe empezar en minúsculas) y larga son usadas por los gestores de paquetes. Nota que hay un espacio al principio de cada línea para la descripción larga (si se quiere dejar una línea en blanco se debe poner un sólo punto despues del espacio obligatorio). Mas información de como hacer una buena descripción en *http://people.debian.org/~walters/descriptions.html*

Paso 10 – Configurando archivo copyright

Es el archivo que proporciona la **información** sobre copyright. Generalmente esta información se encuentra en el archivo COPYING en el directorio del programa fuente. Este archivo debe incluir la información de los nombres de autores y el empaquetador, la URL desde la que viene el código fuente y una línea de Copyright con el año y el propietario del Copyright. Un ejemplo:

This package was debianized by {Tu nombre} < tu dirección de correo electrónico> {Fecha}
It was downloaded from: {URL de la página web}
Upstream Author(s): {Nombres y direcciones de correo de los autores} Copyright:
Copyright (C) {Años} by {Autores} {direccion de correo}
License:

{Añadir el texto de la licencia aqui. Para licencias GNU añadir el encabezado y un enlace al archivo correspondiente en /usr/share/common-licences.}

Packaging:

Copyright (C) {Años} by {Tu nombre} < tu dirección de correo electrónico> released under {la licencia que elegiste para empaquetar}

Para el programa de ejemplo:

This package was debianized by Pepe Lui < programator@gmail.com>

27 Aug 2009 16:36:39 +0200

It was downloaded from: http://www.programa.com/software/programa/

Upstream Author(s): Pepe Lui < programa@gmail.com>

Copyright:

Copyright (C) 2009 by Pepe Lui < programa@gmail.com>

License:

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This package is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program.

On Debian systems, the complete text of the GNU General Public License version 3 can be found in /usr/share/common-licenses/GPL-3'.

The Debian packaging is:

Copyright (C) 2009 by Pepe Lui < programa@gmail.com> and is licensed under the GPL version 3, see above.

Paso 11 – Configurando archivo rules

Uno de los archivos que necesitamos mirar es **rules.** Este archivo hace la mayoría del trabajo para crear el paquete. En realidad, es un archivo de Makefile con los objetivos para **compilar** e instalar la aplicación que después creará el archivo .deb para los archivos.

Las reglas definirán la forma en la que se compilará y empaquetará nuestro programa (puedes añadir todas las opciones que normalmente añadirías al ./configure de un programa, opciones de compilación, optimizaciones, etc). No es necesario hacer cambios aquí para paquetes básicos.

Generalmente **algunas de las siguientes modificaciones** son las más se realizan comúnmente (cambia ***programa*** por el nombre de tu aplicación):

FICHERO RULES

Modificar build-stamp

- Cambia del fichero original rules, este texto:

```
build-stamp: configure-stamp
    dh_testdir

    # Add here commands to compile the package.
    $(MAKE)
    #docbook-to-man debian/programator.sgml > programa.1
    touch $@
```

- Por este otro:

```
build-stamp: configure-stamp
    dh_testdir
    touch build-stamp
```

- ***Modificar clean***

- Cambia del fichero original rules, este texto:

```
clean:
    dh_testdir
    dh_testroot
    rm -f build-stamp configure-stamp
    # Add here commands to clean up after the build process.
    $(MAKE) clean
    dh_clean
```

- Por este otro:

```
clean:
    dh_testdir
    dh_testroot
    rm -f build-stamp configure-stamp
    dh_clean
```

- ***Modificar install***

  - Cambia del fichero original rules, este texto:

  install: build
  dh_testdir
  dh_testroot
  dh_prep
  dh_installdirs

  # Add here commands to install the package into debian/programa.
  $(MAKE) DESTDIR=$(CURDIR)/debian/programa install

  install: build
  dh_testdir
  dh_testroot
  dh_prep
  dh_installdirs

  # Add here commands to install the package into debian/programa.
  $(MAKE) DESTDIR=$(CURDIR)/debian/programa install

  - Por este otro (en los comentarios y continuación se explica):

  install: build
  dh_testdir
  dh_testroot
  dh_prep
  dh_installdirs

  # Add here commands to install the package into debian/programa.
  #$(MAKE) DESTDIR=$(CURDIR)/debian/programa install

  ## Creamos el directorio "programa" (y con -p no mostrar error
  # en caso de que exista) e instalamos (copiamos) el ejecutable en
  # /usr/share/programa.
  # Aunte este ultimo directorio no exista, ahora veremos que se crea en

```
# el fichero debian/dir de nuestro empaquetado
mkdir -p $(CURDIR)/debian/programa
cp programa.py $(CURDIR)/debian/programa/usr/share/programa/programa.py

## Si tienes más módulos, igualmente podrías copiarlos, por ejemplo constants.py
cp constants.py $(CURDIR)/debian/programa/usr/share/programa/constants.py

## Para imágenes, textos u otros datos, lo meteremos todo en una carpeta data
# crearemos el directorio y lo copiaremos recursivamente
mkdir -p $(CURDIR)/debian/programa/usr/share/programa
cp -r data/ $(CURDIR)/debian/programa/usr/share/programa

## Para que nuestro comando pueda ejecutarse de una manera amigable en el
# terminal, como por ejemplo escribiendo "programa" podríamos hacer
# un script de shell que lanzara el programa con "python programa" y
# guardar ese archivo en /usr/bin. Así que copiamos aquí el script a /usr/bin
mkdir -p $(CURDIR)/debian/usr/bin
cp $(CURDIR)/programa $(CURDIR)/debian/programa/usr/bin/programa

## Si tenemos un lanzador programa.desktop (o acceso directo)
# podemos copiarlo a /usr/share/applications/
cp programa.desktop $(CURDIR)/debian/programa/usr/share/applications/

## También podemos hacer un fichero de LEEME y de documentación e
# instalarnos en /usr/share/doc
mkdir -p $(CURDIR)/debian/usr/share/doc/programa
cp README $(CURDIR)/debian/usr/share/doc/programa/README
```

Las siguientes consideraciones que deber tener son:

$(CURDIR) es ~/sandbox/programa.

El directorio $(CURDIR)/debian existe.

El directorio $(CURDIR)/debian/programa todavía NO existe.

Paso 12 – Configurando archivo dirs

Este archivo es el encargado de crear los directorios en caso de que no existan (de ahí que quizás en

el paso anterior haya algunas líneas que sobren, pero aun no lo he comprobado). **Todos** los directorios que se vayan a crear durante el empaquetamiento deben indicarse en este archivo.

Los más comunes para nuestro ejemplo serían:

usr/bin
usr/share/programa
usr/share/applications

Paso 13 – Configurando archivo README.Debian

No confundir este programa con README.source como ya indique antes. Este sera el README de Debian, no el del empaquetado para aquellos que quieran modificarlo.

Un ejemplo de fichero creado originalmente en nuestro programa:

README.Debian
programa for Debian

-- Pepe Lui Fri, 28 Aug 2009 10:13:44 +0200

Que podríamos cambiar a:

README.Debian
programa for Debian

Comentarios, mejoras, fallos y cariño son bienvenidos en http://www.programa.com/software/programa/
-- Pepe Lui Fri, 28 Aug 2009 10:13:44 +0200

Paso 14 – Generar el paquete .deb

Parece que hay demasiados pasos, pero realmente son bastante **sencillos** y haciéndolo un par de veces puedes tener un empaquetado en 15 minutos. Después de todo este tedioso proceso simplemente hay que ejecutar el comando (desde el directorio de fuentes, no el de /debian):

```
$ dpkg-buildpackage -rfakeroot -us -uc
```

Es importante **NO** ejecutar este comando **nunca como root**, ya que podríamos crear conflictos como por ejemplo en los permisos de los archivos movidos por rules (recordad que es en realidad un Makefile) o en el firmado de paquetes.

Para tener privilegios root donde es necesario acudimos a la opción **-rfakeroot** que es dada por el

paquete fakeroot, de esta manera podemos actuar con privilegios sin crear conflictos.

La opción **-us** es para no firmar el paquete fuente y la opción -uc es para no firma el archivo .changes de cambios. Normalmente es **más cómodo** no firmar el paquete en la generación y luego si se desea firmarlo (o si la generación y la firma se lleva a cabo por distintas personas). El firmado es opcional, pero si deseas autentificar tu paquete y tener mayor seguridad, lee el último paso de este artículo.

Paso 15 – Firmado GPG de paquetes .deb

El firmado parece algo complicado, pero realmente son unos pocos comandos y normalmente solo hay que hacer este paso **una vez**.

Si aun no tienes un clave GPG, puedes crear una fácilmente con el siguiente comando:

```
$ gpg --gen-key
```

En el proceso elige la opción 1 (DSA y ElGamal), de tamaño 2048 y 0 para que la clave no caduque. Luego da tus identificadores como el nombre real, comentario y dirección de correo.

Por ejemplo:

Nombre y apellidos: Pepe Lui

Dirección de correo electrónico: programator@gmail.com

Comentario: My GPG key

Después se **generará** la clave (es necesario hacer alguna tarea como mover ventanas o abrir alguna aplicación, para que se genere aleatoriedad, si no te saldrá un mensaje y tendrás que esperar unos segundos para que intente generar la clave de nuevo.

Por ejemplo una posible salida sería (la clave es ficticia):

```
$ gpg –gen-key
```

gpg (GnuPG) 1.4.9; Copyright (C) 2008 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Por favor seleccione tipo de clave deseado:
(1) DSA y ElGamal (por defecto)
(2) DSA (sólo firmar)
(5) RSA (sólo firmar)

¿Su elección?: 1

El par de claves DSA tendrá 1024 bits.

las claves ELG-E pueden tener entre 1024 y 4096 bits de longitud.

¿De qué tamaño quiere la clave? (2048)

El tamaño requerido es de 2048 bits

Por favor, especifique el período de validez de la clave.

0 = la clave nunca caduca

= la clave caduca en n días

w = la clave caduca en n semanas

m = la clave caduca en n meses

y = la clave caduca en n años

¿Validez de la clave (0)? 0

Key does not expire at all

nunca caduca

¿Es correcto (s/n)? s

Necesita un identificador de usuario para identificar su clave. El programa construye el identificador con del Nombre Real, Comentario y Dirección de Correo Electrónico de esta forma:

Nombre y apellidos: Pepe Lui

Dirección de correo electrónico: programa@gmail.com

Comentario: My GPG key

Está usando el juego de caracteres `utf-8'.

Ha seleccionado este ID de usuario: "Pepe Lui (My GPG key) "

¿Cambia (N)ombre, (C)omentario, (D)irección o (V)ale/(S)alir? V

Necesita una frase contraseña para proteger su clave secreta.

Es necesario generar muchos bytes aleatorios. Es una buena idea realizar alguna otra tarea (trabajar en otra ventana/consola, mover el ratón, usarla red y los discos) durante la generación de números primos. Esto da al generador de números aleatorios mayor oportunidad de recoger suficiente entropía.

+++++.+++++.+++++...++++.+++++.+++++.+++++.++++++++++++++++.++++++

++++..++++++++++++++++++..+++++.+++++.+++++.++++++++++..++++++++

++...+++++.++++++++++>+++++++++.....+++++

No hay suficientes bytes aleatorios disponibles. Por favor, haga algún otro trabajo para que el sistema pueda recolectar más entropía (se necesitan 283 bytes más). Es necesario generar muchos bytes aleatorios. Es una buena idea realizar alguna otra tarea (trabajar en otra ventana/consola, mover el ratón, usar la red y los discos) durante la generación de números primos. Esto da al generador de números aleatorios mayor oportunidad de recoger suficiente entropía.

++++++++++++++++++++.+++++.++++...++++++++++++++++++++++++++.+++
+++++++...++++++++++...+++++.....+++++++++++++++.+++++.++++++++++.++
++++++++++++++++++++++++++++++++++>++++++++++>.++++++....>++++
+.......< ..+++++.....>+++++...< .+++++.....>...+++++..< +++++.....+++++^^^^^

gpg: clave 4DEEE67E marcada como de confianza absoluta

claves pública y secreta creadas y firmadas.

gpg: comprobando base de datos de confianza

gpg: 3 dudosa(s) necesarias, 1 completa(s) necesarias,

modelo de confianza PGP

gpg: nivel: 0 validez: 1 firmada: 0 confianza: 0-, 0q, 0n, 0m, 0f, 1u

pub 1024D/4DEEE67E 2009-08-28

Huella de clave = F154 9A31 C480 9E5B CD82 41E1 5984 5A83 1DEE E13E

uid Pepe Lui (Mi GPG Key)

sub 2048g/2FDE8A53 2009-08-28

Lo que realmente nos interesa ahora es el número que aparece después de 1024D/ en este caso 4DEEE67E.

Si has hecho este proceso pero no has apuntado este número, no importa porque siempre puedes consultarlo con el comando:

$ gpg –list-secret-keys

/home/programa/.gnupg/secring.gpg

---

sec 1024D/4DEEE67E 2009-08-28

uid Pepe Lui (Mi GPG Key)

ssb 2048g/2FDE8A53 2009-08-28

*Como **truco adicional**, si quieres exportar tu clave para que en todas las sesiones de terminal este presente, puedes utilizar los siguientes comandos poniendo el valor de tu clave:*

$export GPGKEY=4DEEE67E

$killall -q gpg-agent

$eval $(gpg-agent --daemon)

$source ~/.bashrc

Una vez hecho esto, seguramente no necesites hacerlo muchas veces más, ya que generalmente es suficiente con tener una misma clave.

Por tanto, vamos a firmar el ***código fuente*** *y el archivo .changes. Para ello, necesitamos tener instalado el paquete debsign:*

$sudo apt-get install debsign

Y para el programa de ejemplo (es necesario que la clave este pegada a la opción -k, sino no funcionará):

$debsign -k4DEEE67E programa_0.1.32-0ubuntu1.dsc

$debsign -k4DEEE67E programa_0.1.32-0ubuntu1_i386.changesdebcd