autotools: Herramientas para la creación de proyectos Open Source

Germán Poo Caamaño
Universidad del Bío–Bío — <gpoo@ubiobio.cl>
GNOME Foundation — <gpoo@gnome.org>

22 de septiembre de 2004

Resumen

Este trabajo presenta la creación de un proyecto de desarrollo desde lo mínimo, usando herramientas como *automake*, *autoconf*, explicando la filosofía de trabajo y términando en dos ejemplos, uno básico y otro de mediana complejidad, de tal manera que el lector pueda generar proyectos personalizados y fáciles de mantener.

El objetivo es permitir y fomentar una base de desarrolladores que puedan utilizar bien las herramientas que Linux ofrece, que no solamente sirven para proyectos de programación, sino también a otros tipo, como lo es la generación de documentos.

Introducción

Las herramientas como *automake* y *autoconf* se encuentran disponibles en la mayoría de los proyectos open sources de hoy en día. La mayor ventaja en el uso de estas herramientas se debe a que ayudan a la portabilidad de las aplicaciones a nivel de código fuente, abstrayéndose en la medida de lo posible, de las versiones de las herramientas tradicionales disponibles en cada sistema operativo tipo Unix.

Cada vez que un usuario descarga el código fuente de una aplicación que se encuentra empaquetada, se encuentra comunmente con un script llamado *configure*, el cual al ejecutarse realiza todas las verificaciones y definiciones necesarias para que la aplicación se pueda compilar, y posteriormente instalar, con éxito. Por lo tanto, desde el punto de vista del usuario, el script *configure* es el inicio del proceso que dejará la aplicación funcional en su sistema.

Desde el punto de vista del desarrollador, el script *configure* constituye el resultado final de un proceso, que facilita la distribucion de su aplicación para dejarla disponible a la comunidad.

En este trabajo, se explicarán las herramientas que permiten al desarrollador facilitar su trabajo para, finalmente, dejar disponible un script de autoconfiguracion de su aplicación. Además, se explicará la estructura de directorios en un proyecto open source, la mejor manera de dividirlo y los lineamientos para enfrentar un proyecto que pueda ser mantenible a través del tiempo con distintos desarrolladores.

Este tema es relevante debido a que existen falencias en los desarrolladores a nivel nacional en el uso de este tipo de herramientas, a pesar que es un tema de interés y que han manifestado en más de una ocasión su intención de aprender. Por otra parte, está la competencia entre aprender a crear un proyecto y comenzar a programar inmediatamente, muchas veces acompañado por las barreras de entrada que impone la mantención de archivos *Makefile* y posteriormente el uso de macros *m4*.

En los proyectos en funcionamiento, la cantidad de archivos y directorios, de alguna forma u otra, contribuyen al distanciamiento de los nuevos contribuyentes al aprendizaje del uso de estas herramientas.

1. Estructura de un proyecto GNU

El proyecto GNU ha establecido estándares para el ordenamiento de los proyectos que han sido seguido por muchos otros proyectos, no necesariamente bajo el alero de la Free Software Foundation. Entre ellos se encuentra la disposición básica de archivos. De manera simplificada se muestra un proyecto “hola-mundo” para ejemplificar el contenido básico de gran parte de los proyectos open source.

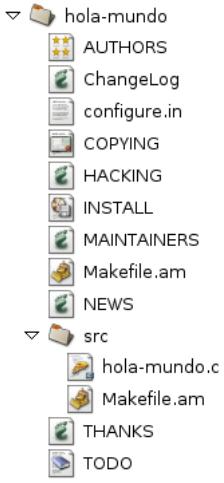

Figura 1: Estructura básica de un proyecto

1.1. Archivos de información de un proyecto

En la figura 1 se muestra el árbol con el contenido de un proyecto. El directorio raíz del proyecto contiene una serie de archivos de textos que permiten mantener un control del proyecto y sirve como primer canal de comunicación con el usuario.

1.1.1. Archivos de información obligatorios

NEWS, es un registro de los cambios visibles al usuario donde los cambios más recientes se deben colocar al inicio.

README, contiene una descripción general del paquete. También es posible indicar instrucciones especiales de instalación o recomendaciones para leer el archivo INSTALL.

AUTHORS, contiene la lista de nombres de quienes han trabajado en la aplicación.

ChangeLog, es un registro de todos los cambios que se han efectuado en la aplicación.

COPYING, especifica los permisos de copia y distribución de la aplicación. Es una buena idea permitir que automake cree este archivo, siempre que se desee licenciar bajo GPL.

INSTALL, instrucciones de instalación de la aplicación. automake provee un archivo genérico, en donde siempre es aconsejable personalizarlo de acuerdo a los detalles de cada aplicación.

1.1.2. Archivos de información opcionales

MAINTAINERS, contiene la lista de nombres de los responsables del proyecto para quien desee ponerse en contacto con ellos.

HACKING, contiene instrucciones para otros desarrolladores que quieran contribuir a la aplicación. Aquí se incluyen normas de sana convivencia como es el estilo de programación, tipos de autorización para efectuar cambios, etc.

VERSION, indica la versión del programa.

THANKS, contiene los créditos a las personas que han contribuido al proyecto pero que no son considerados autores.

TODO, listado de características que se necesitan llevar a cabo. Permite llevar un orden de prioridades entre los autores y facilitar que potenciales contribuyentes sepan dónde y cómo pueden contribuir. Además, para los usuarios que han solicitado alguna característica es una retroalimentación que indica que sus peticiones están siendo consideradas.

1.2. Archivos de programas y configuración

La estructura que se muestra, corresoponde a la visión del desarrollador. Cuando un programa se distribuye, se entregan en muchos casos archivos generados a partir de ciertos procesos. El caso mas característicos es el script configure, que es un script creado en forma automática.

configure.in, contiene las reglas de verificación y construcción del proyecto. Es la base para crear el script configure. Las reglas se escriben en macros *m4*.

Makefile.am, es el archivo que sirve como entrada al programa automake, quien se encargará de generar en forma automática el archivo Makefile, necesario para construir la aplicación. Debe existir un archivo Makefile.am por cada directorio del proyecto.

src/, directorio donde se almacena el código fuente de la aplicación. Eventualmente pueden existir más directorios o con otro nombre, pero lo estándar para proyectos que no son bibliotecas o que no son múltiples aplicaciones es denomilarlo *src*.

autogen.sh, script que permite automatizar la llamada a cada uno de los programas descritos en la sección 2. Este script no se muestra en la figura 1 dado que no es obligatorio.

2. Las herramientas a través de un ejemplo

A continuación se explica el funcionamiento de las herramientas de autoconfiguración de proyectos de desarrollo a través del proyecto "hola-mundo", que consistirá en armar el proyecto para una aplicación muy sencilla que sirva para mostrar la integración de las herramientas y finalizar con el paquete a distribuir a la comunidad.

Las herramientas funcionan en torno a los archivos configure.in y Makefile.am, y es su definición la que guiará el funcionamiento de cada uno de los programas de automatización. Lo importante, en un principio, es conocer como construir el archivo configure.in y cómo actúan los programas en todo a este archivo.

La secuencia de construcción consiste de los siguientes comandos: *aclocal*, *autoheader*, *autoconf* y *automake*, lo que finalmente termina generando el script configure y varios plantillas con extensión *.in* (Makefile.in entre los más importantes).

2.1. aclocal

automake, que es la última utilidad en ser invocada, provee macros para *autoconf*, programa que necesita conocer de su existencia. Para ello, se define como interfaz el archivo aclocal.m4, que será empleado por *autoconf* para buscar macros de *automake*.

aclocal es una utilidad que permite generar automáticamente el archivo aclocal.m4. *aclocal* busca macros en todos los archivos .m4 del directorio en donde se ejecuta y finalmente busca en el archivo configure.in. Todas esas macros se incluirán en el archivo aclocal.m4, así como todas las macros de las cuales dependen. De esta forma, se puede emplear el mismo archivo configure.in para el uso de *autoconf* y *automake* simultáneamente.

Para facilitar las personalización en proyectos que requieran macros propias, se ha definido el archivo acinclude.m4. Las macros allí definidas, pueden ser empleadas también en el archivo configure.in.

El archivo aclocal.m4 solo será generado si se encuentran macros de *automake*, es decir, aquellas que comiencen con el prefijo *AM_*.

Por ejemplo, un archivo configure.in básico (sin funcionalidad):

Programa 1: configure.in básico (sin funcionalidad)

```
1 AC_PREREQ(2.52)
2 AC_INIT(hola-mundo,0.1)
3
4 AM_AUTOMAKE_VERSION(1.7.2)
5
6 AC_OUTPUT(Makefile)
```

Después de ejecutar *aclocal*, se creará el archivo aclocal.m4 bastante pequeño, cuya macro, AM_AUTOMAKE_VERSION, sirve para indicar que se requiere la versión 1.7 de la API de *automake* a utilizar. En este ejemplo, se ha empleado para ilustrar el archivo aclocal.m4 generado.

Sin embargo, con en el ejemplo anterior no es posible utilizar *automake*, puesto que no se encuentran disponibles todas las macros que *automake* requiere. Para ello es necesaria la macro AM_INIT_AUTOMAKE[1], que debe ser la primera macro de *automake* en ser invocada.

Una disgresión entre AC_INIT y AM_INIT_AUTOMAKE. Ambas macros son capaces de inicializar el nombre del paquete y la versión (que más adelante se empleará).

La forma antigua de hacerlo es:

```
AC_INIT
AM_INIT_AUTOMAKE(programa,version)
```

La forma nueva de hacerlo es:

```
AC_INIT(programa,version)
AM_INIT_AUTOMAKE
```

Actualmente AM_INIT_AUTOMAKE acepta el nombre del paquete y la versión, pero no hay garantía que en el futuro siga funcionando de esa forma. Por lo tanto, es conveniente usar el nuevo formato.

2.2. autoheader

autoheader crea una plantilla con un conjunto de directivas #define que pueden emplearse desde los programas en C. Para indicar el nombre del archivo se debe emplear la macro AC_CONFIG_HEADERS. En versiones antiguas[2], se empleaba AM_CONFIG_HEADER, la cual está obsoleta.

El archivo configure.in quedará entonces:

[1] La excepción la constituye AM_AUTOMAKE_VERSION.

[2] Es conveniente verificar la versión de *automake* que se emplea para desarrollar. La versión 1.4 requiere de AM_CONFIG_HEADER. La versión en uso actual (1.7.6) acepta ambas, aunque recomienda AC_CONFIG_HEADERS.

Programa 2: configure.in básico

```
AC_PREREQ(2.52)
AC_INIT(hola-mundo,0.1)

AC_CONFIG_HEADERS(config.h)

AM_AUTOMAKE_VERSION(1.7)
AM_INIT_AUTOMAKE

AC_PROG_CC

AC_OUTPUT([
  Makefile
  src/Makefile
])
```

autoheader creará el archivo config.h.in, una plantilla que en un proceso posterior pasará a ser config.h El nombre del archivo es arbitrario, pero config.h ya es estándar en las aplicaciones. Dicho archivo puede emplearse en los programas a través de la directiva **#include** <config.h>.

Por lo tanto, la secuencia de ejecución será:

```
$ aclocal
$ autoheader
```

En este instante se han creado los archivos aclocal.m4, config.h.in y autom4te.cache. Los dos últimos, generados por autoheader. El resultado en config.h.in es el siguiente:

Programa 3: config.h.in

```
/* config.h.in.  Generated from configure.in
      by autoheader.  */

/* Name of package */
#undef PACKAGE

/* Define to the address where bug reports
    for this package should be sent. */
#undef PACKAGE_BUGREPORT

/* Define to the full name of this package.
    */
#undef PACKAGE_NAME

/* Define to the full name and version of
    this package. */
#undef PACKAGE_STRING

/* Define to the one symbol short name of
    this package. */
#undef PACKAGE_TARNAME

/* Define to the version of this package. */
#undef PACKAGE_VERSION

/* Version number of package */
#undef VERSION
```

Como se puede apreciar, es una plantilla, donde cada directiva **#undef** será reemplazada por los valores reales y cuyos valores serán obtenidos durante la ejecución del script *configure*, más adelante. De esta manera, se define en un solo lugar el nombre de la aplicación y su versión, facilitando la mantención y la liberación de nuevas versiones.

2.3. autoconf

autoconf es una herramientas que permite construir paquetes de programas de una manera más portable. Provee una seria de pruebas que se realizan en el sistema donde se instalará para determinar sus características antes de ser compilado, de tal forma que el código fuente de un programa puede adaptarse en mejor manera a los diferentes sistemas.

autoconf permite generar un script, llamado *configure*, que es el encargado de realizar las comprobaciones para determinar la disponibilidad y ubicación de todos los requisitos para la construcción exitosa de un programa. Cuando se encuentra bien construidos, permite instalar y desinstalar muy fácilmente las aplicaciones.

autoconf genera un script a partir de de las definiciones en configure.in, y sus macros se reconocen por el uso del prefijo *AC_*.

autoconf primero busca en las macros que tiene en forma predefinida y luego en el archivo aclocal.m4, si es que existe[3].

En estos momentos se pueden detallar las macros no explicadas del programa 2:

- AC_PREREQ, indica que la versión de *autoconf* requerida debe ser igual o mayor a 2.52:
- AC_INIT, es la macro encargada de inicializar las funcionalidades de *autoconf*. Recibe como parámetros el nombre de la aplicación y su respectiva versión. Cada que se libere una nueva versión, se debe incrementar el segundo parámetro, para indicar la nueva versión en desarrollo.
- AC_PROG_CC indica que se trata de programas en C y que se requiere de compilador y las bibliotecas básicas de desarrollo.
- AC_OUTPUT indica los archivos que finalmente deben ser generados por el script *configure*.

2.4. automake

automake es una herramienta que permite generar en forma automática archivos *Makefile*, de acuerdo a un conjunto de estándares, simplificando el proceso

[3] Recordar que aclocal.m4 se genera previamente con *aclocal*.

de organización de las distinas reglas así como proveyendo funciones adicionales que permiten un mejor control sobre los programas.

La utilidad make es ampliamente usada en el desarrollo de aplicaciones, ya que a partir de un archivo, llamado *Makefile*, que contiene reglas de dependencia es capaz de determinar las acciones a seguir, comúnmente determinar que programas deben compilarse para obtener la aplicación.

Sin embargo, es tedioso crear las reglas para que construya a través de varios subdirectorios, con seguimiento automático de dependencias. Si se trabaja en varios proyectos, prácticamente significa reinventar la rueda en cada uno de ellos, sin mecionar los problemas de portabilidad.

automake permite automatizar esta labor, para ello emplea un archivo Makefile.am como fuente, en donde se indica lo esencial a construir, y será necesario por cada uno de los directorios en que se necesite realizar alguna tarea. A partir de ello, generará una plantilla llamada Makefile.in, la que finalmente se traducirá en Makefile cuando se ejecute el script *configure*.

Los archivos Makefile.in que se generarán dependerán exclusivamente de lo que se indique en la macro AC_OUTPUT.

automake puede proveer los archivos que son estándares (ver la sección 1.1) en cada aplicación, y salvo que se desee realizar algún cambio, es posible trabajar transparentemente con lo que sugiera, para ello basta usar la opción '–add-mising' de *automake*. Esto creará enlaces simbólicos de los archivos, para realizar una copia es necesario añadir la opción '–copy'. Es importante revisar el archivo COPYING, puesto que contiene la licecia de uso de la aplicación. Por omisión, será GPL.

El archivo Makefile.am del directorio raíz contendrá:

Programa 4: Makefile.am del proyecto 'hola mundo'

```
1 SUBDIRS = src
2
3 EXTRA_DIST = AUTHORS ChangeLog NEWS README
      THANKS TODO
```

En donde se indica, que debe procesar el subdirectorio *src*. SUBDIRS es una variable genérica, en donde se especifican todos los subdirectorios que se deben procesar. De cierta forma, se indican los directorios en que hay dependencias que se deben satisfacer. Cuando se ejecute la utilidad *make*, ingresará primero a cada uno de los subdirectorios y así sucesivamente.

Luego, se indican todos los archivos que son parte extra de la aplicación y que se desean distribuir, además de la aplicación ejecutable.

Dentro del directorio *src*, también se encuentra un archivo Makefile. En este directorio se encuentra el programa propiamente tal, por lo tanto las reglas son distintas.

Programa 5: src/Makefile.am del proyecto 'hola mundo'

```
1 bin_PROGRAMS = hola-mundo
2
3 hola_mundo_SOURCES = hola-mundo.c
```

La variable bin_PROGRAMS indica como se llamará la aplicación final, el archivo binario (ejecutable). Si se desea generar una biblioteca y no un programa ejecutable, se debe emplear libexec_PROGRAMS.

Cabe notar que ha sido llamado 'hola-mundo'. Ese mismo texto se empleará como prefijo para otras reglas, tales como: *hola_mundo*_SOURCES[4], *hola_mundo*_LDADD, por nombrar las más comunes.

2.5. El programa

Como se trata de una aplicación de ejemplo, el programa "hola-mundo", imprimirá el nombre de la aplicación, su versión y el clásico "hola mundo". El nombre de la aplicación y versión se obtienen de las definiciones de la macro AC_INIT en el archivo configure.in y utilizados a través del archivo config.h.

Programa 6: Programa hola-mundo.c

```
1  #ifdef HAVE_CONFIG_H
2  #  include <config.h>
3  #endif
4
5  #include <stdio.h>
6
7  int
8  main (int argc, char **argv)
9  {
10         printf ("%s %s: Hola mundo\n",
               PACKAGE_NAME, PACKAGE_VERSION);
11 }
```

2.6. Finalizando la configuración

En este punto, ya se encuentran todos los archivos preparados para ejecutar el script *configure*. Hay que notar que, al momento de distribuir la aplicación, no es necesario que la contraparte disponga de *aclocal*, *autoheader*, *autoconf* y *automake;* el script contiene toda la información necesaria para realizar las verificaciones en forma autónoma.

[4]Notar que se reemplazó el símbolo '-' por '_' en el nombre de la regla.

Ya se encuentra todo preparado y resumiendo, la ejecución completa debiera ser:

```
$ aclocal
$ autoconf
$ automake --add-missing
$ ./configure
$ make
$ src/hola-mundo
```

Ya se encuentra nuestro programa compilado y funcionando. Si se desea distribuirlo, basta ejecutar:

```
$ make distcheck
[...]
==========================================

hola-mundo-0.1.tar.gz is ready for
    distribution
==========================================
```

De esta forma, se ha obtenido la aplicación empaquetada en un archivo listo para ser liberado como la versión 0.1 de hola-mundo.

Entre las reglas mas comunes se encuentran: make install, make uninstall, make clean, make distclean.

3. Internacionalización de aplicaciones

Si se desea que la aplicación esté disponible en diferentes idiomas, es necesario internacionalizarla, de tal forma de separar todas las cadenas de textos que aparecen en la aplicación y que puedan ser manipuladas por el equipo de traducción respectivo. El proceso de internacionalizar una aplicación también se conoce como i18n. A su vez, el proceso de traducir mensajes de un programa se conoce como l10n.

La idea básica de i18n es marcar aquellas cadenas de texto para que puedan ser traducidas. Para ello existen funciones especiales, donde la más conocida es gettext.

3.1. intltool

La utilidad gettext es la que permite extraer cadenas desde programas, sin embargo,intltool es una herramienta que extiende su funcionalidad, permitiendo la traducción de archivos desktop (empleados en el menú), glade, gconf, xml, entre otros. Es posible escribir programas traducibles sin emplear intltool, pero es mucho más cómodo utilizar la infraestructura existente en GNOME, aunque se utilice para proyectos no GNOME.

Intltool permite:

1. Detectar las herramientas necesarias para la configuración y construcción de la aplicación.
2. Extraer las cadenas a traducir
3. Mezclar las traducciones en la aplicación final

3.1.1. Cambios en la estructura de directorios

Se debe añadir el directorio «po», donde se ubicarán el archivo con todas las cadenas traducibles (hola-mundo.pot) y los archivos en cada idioma (es.po, fr.po, etc.). Los archivos esenciales dentro de este directorio son:

POTFILES.in, contiene una lista de todos los archivos en el proyecto que contienen cadenas a traducir.

POTFILES.skip, contiene una lista de los archivos que no deben ser considerados como traducibles. Es útil cuando hay archivos de programas que están siendo usados, pero que aún permanecen en el proyecto y así evitar esfuerzo innecesario por parte del equipo de traductores.

ChangeLog, un registro de todos los cambios efectuados en el manejo de i18n y l10n.

A través de intltool-update se puede obtener los nombres de los archivos que tienen cadenas a traducir, lo cual se detalla en la sección 3.1.5.

3.1.2. Cambios en configure.in

Para añadir soporte de i18n con intltool se requiere añadir algunas macros al archivo configure.in[5], las cuales se muestran en el listado 7. Lo primero es añadir la macro AC_PROG_INTLTOOL, que indica que se hara uso de esta herramienta y, en forma particular, se requiere al menos de la versión 0.23.

Programa 7: configure.in con soporte i18n

```
1  AC_INIT(hola-mundo,0.2)
2
3  AM_INIT_AUTOMAKE
4
5  AM_CONFIG_HEADER(config.h)
6
7  AC_PROG_CC
8
9  AC_PROG_INTLTOOL([0.23])
10
11 GETTEXT_PACKAGE=hola-mundo
12 AC_SUBST(GETTEXT_PACKAGE)
```

[5] Se ha cambiado la versión a 0.2 para esta versión "internacional".

```
13 AC_DEFINE_UNQUOTED(GETTEXT_PACKAGE, "
       $GETTEXT_PACKAGE",[The gettext package])
14
15 ALL_LINGUAS="es"
16 AM_GLIB_GNU_GETTEXT
17
18
19 AC_OUTPUT([
20     Makefile
21     src/Makefile
22     po/Makefile.in
23 ])
```

Entre la línea 11 y 13 se muestra la forma para definir símbolos al preprocesador de C. La línea 11 asigna un valor de texto a la variable GETTEXT_PACKAGE, la macro AC_DEFINE_UNQUOTED se encarga de dejar disponible el valor para GETTEXT_PACKAGE para los programas, en términos de programación, será equivalente a:

```
#define GETTEXT_PACKAGE "hola-mundo"
```

el cual se registrará en el archivo config.h. La macro AC_SUBST permite que se pueda emplear @GETTEXT_PACKAGE@ dentro de los archivos Makefile.am y puedan acceder a su valor. En este caso, será útil para el Makefile que se generará en el directorio “po”.

Posteriormente, en la línea 15, se ha definido una cadena vacía para ALL_LINGUAS, debido a que aún no hay idiomas disponibles. Normalmente contendrá los códigos de los idiomas separados por espacios, por ejemplo:

```
ALL_LINGUAS="de es fr sv"
```

En la línea 16, la macro AM_GLIB_GNU_GETTEXT se encarga de realizar las verificaciones necesarias para el uso de gettext. Además se encarga de definir los símbolos HAVE_GETTEXT y ENABLE_NLS en el archivo config.h. La macro AM_GLIB_GNU_GETTEXT es propia del entorno de desarrollo GTK+, sin embargo es bastante. Si se quiere evitar su uso, puede ser reemplazada por AM_GNU_GETTEXT, con prestaciones menores.

Finalmente, línea 22, se ha añadido un nuevo archivo de salida: po/Makefile.in.

3.1.3. Cambios en Makefile.am

Se añade el directorio «po» en la lista de subdirectorios en los cuales make debe ingresar. Además, se han añadido las utilidades de intltool como archivos extras.

Programa 8: Makefile.am con soporte i18n

```
1  SUBDIRS = src po
2
3  EXTRA_DIST = AUTHORS ChangeLog NEWS README \
4           intltool-extract.in \
5           intltool-merge.in   \
6           intltool-update.in
7
8  CLEANFILES = \
9           intltool-extract \
10          intltool-merge \
11          intltool-update
```

También es necesario modificar el archivo Makefile.am en el directorio src. Allí se añade la variable INCLUDES, en donde se define el lugar donde residirán las cadenas traducidas. El símbolo allí indicado será utilizado dentro del programa.

Además, se añade un nuevo archivo dentro de los programas fuentes: hola-mundo-i18n.h, descrito más adelante.

Programa 9: src/Makefile.am con soporte i18n

```
1  bin_PROGRAMS = hola-mundo
2
3  INCLUDES = \
4           -DHOLA_MUNDO_LOCALEDIR=\""$(datadir)
                /locale"\"
5
6  hola_mundo_SOURCES = hola-mundo.c \
7           hola-mundo-i18n.h
```

3.1.4. Cambios en el código fuente

Como se indicó en la sección 3.1.3, se ha añadido un archivo de cabecera, cuya línea esencial es la 9, donde se asocia _() como un sinónimo de la función gettext() y de esta manera simplificar la lectura de los programas respecto a las cadenas que se han marcado como traducibles.

Programa 10: src/hola-mundo-i18n.c

```
1  #ifndef __HOLA_MUNDO_I18N_H__
2  #define __HOLA_MUNDO_I18N_H__
3
4  #ifdef HAVE_CONFIG_H
5  #  include <config.h>
6  #endif
7
8  #ifdef ENABLE_NLS
9  #  define _(String) (const char *) gettext(
       String)
10 #else
11 #  define _(String) (String)
12 #endif
13
14 #endif /* __HOLA_MUNDO_I18N_H__ */
```

En caso que no este disponible el conjunto de herramientas de internacionalización, se ha definido de tal forma que no provoque problemas en compilar.

El programa hola-mundo.c también sufre modificaciones:

Programa 11: src/hola-mundo.c con soporte i18n

```
1  #ifdef HAVE_CONFIG_H
2  #  include <config.h>
3  #endif
4
5  #include <stdio.h>
6  #include "hola-mundo-i18n.h"
7
8  int
9  main (int argc, char **argv)
10 {
11
12 #ifdef ENABLE_NLS
13         bindtextdomain (GETTEXT_PACKAGE,
               HOLA_MUNDO_LOCALEDIR);
14         bind_textdomain_codeset (
               GETTEXT_PACKAGE, "UTF-8");
15         textdomain (GETTEXT_PACKAGE);
16 #endif
17
18         printf (_("%s %s: Hello world\n"),
               PACKAGE_NAME, PACKAGE_VERSION);
19 }
```

Como se aprecia, se usa la ENABLE_NLS para comprobar si se está compilando con soporte i18n, en cuyo caso define el dominio o ámbito de las traducciones, normalmente reducido al dominio del programa y que se encuentra definido en GETTEXT_PACKAGE, el cual fue declarado previamente en el archivo configure.in (ver listado 7 en la sección 3.1.2). También se especifica el directorio donde se encuentran las traducciones, la cual fue definida en el archivo src/Makefile.am. Además, se especifica el conjunto de caracteres a emplear, en este caso UTF-8.

Finalmente, en la línea 18, se puede apreciar que la cadena ha sido marcada para ser traducida al colocarla dentro de la función _(). Desde el punto de vista de programación, no recarga con nombres de función largos ni distrae la lectura para lo que es una tarea rutinaria en proyectos grandes.

El texto de la función printf ahora aparece en inglés, lo cual es lo recomendable, pueso que es mucho más sencillo encontrar traductores de inglés a otro idioma que de español a otro idioma.

3.1.5. Preparación del ambiente

Dentro de la sencuencia de comandos, es necesario añadir glib-gettextize[6], que crea el archivo po/-Makefile.in.in, e intltoolize, para disponer de las herramientas para manejar las cadenas a traducir.

```
$ aclocal
$ autoheader
$ glib-gettextize --copy --force
$ intltoolize --copy --force --automake
$ autoconf
$ automake --add-missing
```

Con esto se dispone de la infraestructura básica para tener una aplicación disponible en distintos idiomas. Sin embargo, aún falta un trabajo que realizar en el directorio “po”, y que consiste en determinar el contenido del archivo POTFILES.in. Para ello, dentro de dicho directorio basta invocar el comando:

```
$ intltool-update --maintain
```

el cual mostrará los archivos que tienen cadenas a traducir y que no se encuentran en el archivo POTFILES.in. Para facilitar el trabajo, también se genera un archivo llamado “missing”[7] que puede ser añadido al final de POTFILES.in.

En estos momentos ya es posible ejecutar el script configure y compilar la aplicación.

3.1.6. Ultimos pasos: traducir las cadenas

A estas alturas se dispone de toda la infraestructura para poder comenzar a traducir la aplicación. Por lo tanto, es necesario generar el archivo maestro que contiene todas las cadenas a traducir y que puede ser empleado para comenzar el soporte para un nuevo idioma. Dentro del directorio «po» basta ejecutar:

```
$ make update-po
$ mv hola-mundo.pot es.po
```

De esta forma se generará el archivo hola-mundo.pot[8] que es una plantilla, el cual empleamos para comenzar la traducción al español (es.po). Para que las cadenas se actualicen de acuerdo a como cambia la aplicación y que además se distribuya, es necesario agregar “es” en la variable ALL_LINGUAS del archivo configure.in.

Finalmente, tras ejecuta make distcheck, se tendrá la versión 0.2 del programa hola-mundo.

4. Personalización de un ambiente de desarrollo

El trabajo de scripts como configure establecen el directorio /usr/local como predeterminado. Sin embargo, si se quiere probar la aplicación instalada se

[6] En caso que no se disponga, se puede emplear gettexttize que es más limitado.

[7] Este archivo puede ser borrado, porque solo es para informar en un instante dado.

[8] El nombre hola-mundo se obtiene de la definición de GETTEXT_PACKAGE en el archivo configure.in.

requieren privilegios para ubicarlos en dicho directorio. Por otro lado, las aplicaciones en desarrollo eventualmente pueden provocar conflictos con bibliotecas u otros programas.

Por lo tanto, es conveniente personalizar un ambiente en donde instalar las aplicaciones de desarrollo, aprovechando las potencialidades de configure para establecer un directorio distintos que no requiera privilegios ni pueda tener repercuciones en el funcionamiento del sistema.

El proceso es muy sencillo, y sin embargo, es muchas veces desconocido. A través de dos scripts veremos que fácil resulta. Primero se debe elegir un directorio donde realizar estas instalaciones, por ejemplo en /home/gpoo/desarrollo se puede ubicar los directorios con diversos proyectos, y la instalación se puede realizar en /home/gpoo/desarrollo/install. Dentro de install se creará una jerarquía similar a lo que existe en /usr, esto se hará en forma automática al ejecutar make install.

El primer script define un conjunto de variables de ambiente, para que el directorio personalizado tenga preferencia y no se mezcle involuntariamente con los del sistema.

```
#!/usr/bin/bash

BASE=/home/gpoo/desarrollo

export PATH=$BASE/install/bin:${PATH}
export LD_RUN_PATH=$BASE/install/bin
export LD_LIBRARY_PATH=$BASE/install/lib
ulimit -c unlimited
```

Suponiendo que el primer script se encuentra en ~/bin y se llama install-test.sh, bastaría añadir la siguiente línea en el el archivo .bashrc[9] (nuestro segundo script):

```
alias go-install='source ~/bin/install-test.
    sh'
```

Para instalar la aplicación, es necesario indicarle al script configure cual es el lugar donde se desea alojar:

```
$ ./configure --prefix=/home/gpoo/desarrollo
    /install
```

Cuando se desee probar el programa recién instalado, sólo se debe ejecutar go-install y luego ejecutar la aplicación.

5. Conclusiones

Las herramientas como autoconf y automake en un principio pueden parecer difíciles de entender y manejar, pero se ha visto que en realidad es bastante sencillo, más aún si se compara con todas las características que provee, como lo es la verificación de dependencias para construir una aplicación, automatización de procesos de instalación, desintalación, y distribución. Por otra parte, estas herramientas se integran muy bien con el proceso de traducción de aplicaciones a través de programas como gettext e intltool.

Si bien, las macros mostradas son las más sencillas, también son las de uso común, y en la medida que cada desarrollador emplee las autotools podrá ir añadiendo macros más complejas para tareas más específicas.

[9] O el que corresponda al shell preferido por el programador.

Referencias

[1] Gary V. Vaughan, Ben Elliston, Tom Tromey, Ian Lance Taylor. “GNU autoconf, automake and libtool, 2000, http://sources.redhat.com/autobook/

[2] Malcolm Tredinnick, “Internationalising GNOME applications”, october 2003, http://www.gnome.org/~malcolm/i18n/

[3] Christian Rose, “L10N Guidelines for Developers”, 2002-2003, http://developer.gnome.org/doc/tutorials/gnome-i18n/developer.html