Autoconf y Automake: Cómo logré construir un tarball

Por shicefgo.

1. La idea

Este escrito ha surgido de la idea de anotar lo aprendido para llegar a realizar un paquete instalable de la forma habitual en GNU/Linux, es decir, utilizando los comandos: **tar -zxvf nombre_paquete.tar.gz**, entrando en el directorio correspondiente y ejecutando **./configure**, **make** y, como root, **make install** (recuerda: "no uses el nombre de root en vano"). Tomar apuntes para uno mismo es algo que está bien, pero pienso que estaría todavía mejor si, con un pequeño esfuerzo añadido, esos apuntes contribuyeran a que alguien más pueda orientarse en los vericuetos que uno ha ido explorando. Siempre que leo documentación técnica, me encuentro que casi todo está escrito en la forma: "haz esto y luego aquello, después esto otro, y así." Es decir, una serie de instrucciones bien explicadas por alguien que conoce de qué va la cosa. Cómo no soy experto en nada (alguien dijo que un experto es aquél que sabe cada vez más sobre menos, hasta que termina sabiéndolo absolutamente todo sobre nada), he pensado que, en lugar de decirte lo que hay que hacer, voy a contarte cómo lo he hecho yo. Si quieres, puedes seguir mis pasos bajo tu propio riesgo, y puedes contar con que me esforzaré para que la lectura de "esto" no te resulte demasiado tediosa.

2. Qué es un tarball

Se acostumbra a denominar *"tarball"* a un paquete preparado para ser compilado e instalado, normalmente bajo un sistema Unix-GNU/Linux, cuyo nombre suele terminar en ".tar.gz". La palabra empezó siendo utilizada para referirse a los archivos construidos usando el comando **tar**. Originalmente, esos archivos eran guardados en cinta, pero en la actualidad **tar** se utiliza ampliamente para acumular una gran cantidad de archivos en uno solo preservando la información del sistema de ficheros, como permisos del usuario y del grupo, fechas, y la estructura de directorios. De acuerdo con la filosofía Unix de "un trabajo, un programa", el comando **tar** no comprime, sólo empaqueta. Para comprimir se utiliza **gzip**, del cual procede el sufijo ".gz", y también **bzip2**, que da origen al sufijo ".bz2". Y ya que estamos,

quizás le venga bien a alguien saber como descomprimir “del tirón” los archivos comprimidos con bzip2: **tar -jxvf nombre_archivo.tar.bz2**.

La ventaja de utilizar las autotools es que podemos poner nuestro programa a funcionar en cualquier sistema con sabor Unix y, más generalmente, en cualquier sistema que tenga soporte para ellas, consiguiendo una buena portabilidad y no teniendo que realizar el trabajo más que una sola vez. ¿Que ya sabías todo esto? ¡Pues habérmelo dicho! :-).

3. Lo que hay que tener

Además del programa que vayamos a “empaquetar”, necesitamos tener instalados, obviamente, un compilador y los paquetes **autoconf** y **automake**. El compilador al que voy a referirme será **gcc**, y el lenguaje de programación el C, aunque lo mismo podría servir para C++. También será útil disponer de autoconf-archive, autobook (el libro de las autotools), libtool, y no sería una mala idea instalarse la documentación relativa a estos paquetes.

También se asume alguna idea de programación en la shell Bash, y un somero conocimiento de los makefiles, como por ejemplo el detalle de que una orden ejecutable debe ir precedida de una tabulación.

4. Las primeras dificultades

Cuando conseguí escribir en C algo casi digno de ser llamado “un programa” (el almanaque, sin ir más lejos), me encontré con que, debido a una acumulación de circunstancias, llegó a encontrarse disponible para quien lo quisiera a través de los foros de Fentlinux, y la compilación se efectuaba de una forma bastante chapucera, con un Makefile de andar por casa que, si bien compilaba, no permitía chequear dependencias y avisar al usuario de que le faltaba tal o cual cosa, en caso de que así fuera.

Entonces llegó el día en que me decidí a meterle mano al asunto. Era un día como otro cualquiera, no recuerdo que tuviese nada de especial, pero me pareció que se había adelantado un poco a mis previsiones, ya que no tenía ni idea de por donde empezar. Cuando le meto mano a algo es porque ya creo saber lo suficiente como para intentarlo, lo que me ha llevado a no pocas frustraciones y a leer el inglés contra mi voluntad, ya que eso me hace perder más tiempo que leyendo en español, entre que encuentro el significado literal y consigo interpretarlo, que no sé por qué el inglés tiene que ser tan raro. Y cuando utilizo un traductor automático, me salen cosas como esto: “`Por supuesto, los ajustes de defecto no satisfarán cada uno, así que las opciones se pueden agregar a ’ configuran ’ la escritura, y las cosas como compile las banderas pueden ser pased en variables del environmnet.`” Y esta es de las que se entienden.

A continuación mostraré el esquema general del Makefile que conseguía compilar mi programilla antes de que hubiese logrado utilizar las autotools. (Aviso: Una auténtica y verdadera chapuza, pero lo hice yo sólo). La compilación se realizaba a partir de tres archivos: un Makefile y otro llamado makeinclude en

el directorio raíz de la aplicación, y otro Makefile en el directorio de los archivos fuentes (src). Lo del makeinclude se basaba en la genial idea de que, si llegaba a desarrollar el proyecto en varios directorios con archivos fuentes a compilar en cada uno de ellos (iluso que era uno), sólo tendría que incluirlo en el Makefile de cada directorio con fuentes, con alguna pequeña modificación tal vez, y trabajo de escritura que me ahorraba. Sin duda que esta idea tendrá sus detractores, pero como sólo tuve que utilizar un directorio de archivos fuentes, funcionó a la perfección. Veamos ahora el código del Makefile principal:

```
DIRS = src

all:
   for dir in $(DIRS); do\
       echo "=== Compilando en $$dir ===";\
       (cd $$dir; $(MAKE) $(MFLAGS)) || break;\
   done

clean:
    rm -f ./src/*.o
```

El del makeinclude:

```
CC = gcc
NOMBRE_EJECUTABLE = ../almanaque
OPTIM =  -Wall -Wunused -Wmissing-prototypes -O2
GTKFLAGS = ‘pkg-config --cflags gtk+-2.0‘
GDAFLAGS = ‘pkg-config --cflags libgda‘
GTKLDADD = ‘pkg-config --libs gtk+-2.0‘
GDALDADD = ‘pkg-config --libs libgda‘
```

Y, por último, el Makefile dentro del directorio src:

```
include ../makeinclude

FUENTES = \
gtkmisc.c \
fecha.c \
cantidad.c \
gregoriano.c \
main.c

OBJS = ${FUENTES:.c=.o}
CFLAGS = ${OPTIM} ${GTKFLAGS} ${GDAFLAGS}
LDADD = ${GTKLDADD} ${GDALDADD}
EJECUTABLE = ${NOMBRE_EJECUTABLE}

all: ${OBJS}
    ${CC} -o ${EJECUTABLE} ${OBJS} ${LDADD}

.c.o:
    ${CC} ${CFLAGS} -c $<
```

Como se dijo antes, las líneas "`${CC} -o ${EJECUTABLE} ${OBJS} ${LDADD}`" y "`${CC} ${CFLAGS} -c $<`" deben ir precedidas de una tabulación, no de espacios en blanco. Esto me permitía compilar y también borrar los archivos .o con un **make clean**, pero cuando modificaba un archivo de cabecera (.h), tenía que compilarlo todo con **make -W**. Un esquema como este, o parecido, puede ser útil y hasta recomendable al comienzo de un proyecto para compilar las primeras veces, pero una vez que la cosa va tomando forma y, sobre todo, si queremos poner nuestro trabajo a disposición de otras personas, se hace necesario utilizar algo mejor hecho y, por qué no, más profesional. No es mi propósito entrar en los detalles de la creación de Makefiles, de ahí que no comente el código anterior, para eso hay en internet gran cantidad de tutoriales a partir de los cuales cualquiera que ponga el suficiente interés podrá obtener los conocimientos necesarios para crearse los suyos propios. Así pues, mis primeras dificultades estaban relacionadas con los archivos de cabecera, que no eran tenidos en cuenta, y con la imposibilidad de chequear dependencias, ya que lo de la portabilidad ni me lo había planteado.

5. Así fue como lo hice

Una vez situado en el directorio raíz de la aplicación (no lo he dicho antes, pero es altamente conveniente organizar el proyecto dentro de un directorio, a partir del cual "cuelguen" los demás que sean necesarios, entre ellos el de los fuentes, típicamente llamado `src`), ejecuté los siguientes pasos, no sé si exactamente por este orden, pero este es el orden que pienso seguir en lo sucesivo.

Primero creo unos archivos necesarios para la correcta ejecución de automake con el siguiente comando:

~$ touch README AUTHORS NEWS THANKS ChangeLog

En el momento de su creación estarán vacíos, lógicamente, y su contenido suele ser:

- `README`: Explicaciones que el autor considere necesario incluir para la correcta compilación e instalación del programa, así como cualquier otra información técnica útil al usuario final.
- `AUTHORS`: Los nombres del autor, o autores, de la aplicación.
- `NEWS`: Cambios visibles al usuario, con los más recientes al principio del archivo (ordenados de último a primero).
- `THANKS`: Agradecimientos especiales del autor o autores a personas o entidades que de alguna manera hayan contribuido a la realización del proyecto, pero que no pertenezcan a `AUTHORS` o a `MAINTAINERS`.
- `ChangeLog`: Un registro, ordenado por fechas y también de último a primero, de todos los cambios que se van realizando en la aplicación, sean o no visibles al usuario.

Opcionalmente se pueden incluir unos archivos más, ya que automake funcionará perfectamente aunque no existan: `MAINTAINERS` y `HACKING`. El primero contendría los nombres de todos los responsables del proyecto, no sólo de los autores del código y diseño, y en el segundo irían instrucciones para otros programadores que quieran realizar modificaciones en el programa. En él se pueden incluir cosas tales

como notas sobre el estilo de programación adoptado (por si alguien no lo advierte leyendo el código), para qué tipo de cambios sería conveniente pedir permiso a los autores, y cosillas de esa índole.

Ninguno de estos archivos ha de ser completado obligatoriamente, aunque sí es bastante conveniente preocuparse de que por lo menos el `README` y el `ChangeLog` contengan la información que les concierne de la manera más correcta posible. Sólo es obligatorio que existan los cinco primeros, los creados con el comando **touch**, porque si faltase alguno de ellos al ejecutar **automake** se producirían errores. El archivo `THANKS` no es necesario para **automake**, pero sí lo será para ejecutar **make dist** cuando todo esté listo para crear el paquete a distribuir.

Ha llegado el momento de decir que mi proyecto contenía un total de cuatro subdirectorios a distribuir junto con el raíz. El raíz se llamaba (y se llama) `almanaque`, y los otros cuatro son: `almanaque/src`, `almanaque/gtkrc`, `almanaque/dat` y `almanaque/img`.

A continuación, creo y edito los correspondientes `Makefile.am` para cada uno de los directorios mencionados, según se describe:

./Makefile.am

```
SUBDIRS = src gtkrc dat img
EXTRA_DIST = AUTHORS ChangeLog NEWS README THANKS \
gtkrc/almanaque.gtkrc dat/santoral.xml img/amq*
```

En la primera línea, simplemente indico los subdirectorios que contienen archivos a incluir en el paquete, que pasan a formar parte de la variable “SUBDIRS” y en la segunda (sólo son dos, obsérvese el carácter ’\’) indico los archivos extra que formarán parte del paquete en la variable “EXTRA_DIST”. No es necesario mencionar aquí el contenido del directorio `src`, aunque sí el directorio mismo.

src/Makefile.am

```
bin_PROGRAMS = almanaque

almanaque_SOURCES = \
fecha.h \
fecha.c \
cantidad.h \
cantidad.c \
gtkmisc.h \
gtkmisc.c \
calendario.h \
almanaque.h \
gregoriano.c \
main.c
```

```
CFLAGS = -D'DATADIR="$(datadir)"' -O2 -Wall -Wunused \
-Wmissing-prototypes

INCLUDES = -DXTHREADS -I/usr/include/gtk-2.0 \
-I/usr/lib/gtk-2.0/include -I/usr/X11R6/include \
-I/usr/include/atk-1.0 -I/usr/include/pango-1.0 \
-I/usr/include/freetype2 -I/usr/include/glib-2.0 \
-I/usr/lib/glib-2.0/include -I/usr/include/libxml2

LDADD = -lgtk-x11-2.0 -lgdk-x11-2.0 -latk-1.0 \
-lgdk_pixbuf-2.0 -lm -lpangoxft-1.0 -lpangox-1.0 \
-lpango-1.0 -lgobject-2.0 -lgmodule-2.0 -ldl \
-lglib-2.0 -L/usr/lib -lxml2 -lz -lpthread
```

Este es el archivo a partir del cual se produce la compilación. A la variable bin_PROGRAMS le asigno el nombre elegido para el binario resultante. En almanaque_SOURCES se almacenan los nombres de todos los archivos que deben intervenir en la compilación y que estén en el directorio ./src, claro. Si el proyecto se llamase, por ejemplo, lavacapaca, la variable la hubiese llamado lavacapaca_SOURCES, obviamente. Aquí ya he incluido los archivos de cabeceras, de modo que cuando modifique uno de ellos y a continuación recompile, sólo se compilarán los fuentes que tengan ese archivo incluido.

La siguiente línea son los "flags", o señales, que le pasamos al compilador (CFLAGS), y he resaltado en ella una parte porque la considero de especial importancia: **-D'DATADIR="$(datadir)"'**. Lo que hace el "flag", o señal, **-D** es crear una macro de nombre `DATADIR` que se puede utilizar en un archivo fuente igual que si hubiese estado definida en él con la directiva `#define`, siendo su valor el obtenido de `$(datadir)`, en este caso el destino para la instalación indicado con **--prefix=PREFIX**, siendo PREFIX el destino elegido por el usuario o /usr/local por defecto. Es decir, que si el usuario ejecuta: **./configure --prefix=/home/yo/aqui/mismo**, podré capturar en el código fuente, a través de la macro DATADIR, ese camino para obtener los datos que necesito, como el santoral, los colorines y los iconos. Eso está hecho tal que así en el archivo gregoriano.c:

```
strncpy (elSantoral, DATADIR, 55);
strncat (elSantoral, "/almanaque/santoral.xml", 27);

/* Aquí van unas líneas que no afectan al ejemplo. */

doc = xmlParseFile (elSantoral);
if (doc == NULL) {
   mensaje (NULL, GTK_MESSAGE_INFO, GTK_BUTTONS_OK,
   "No se encuentra el santoral.\n");
   return;
}
```

Esto permite no tener que presuponer que el usuario va a efectuar la instalación por defecto en `/usr/local`. Las autotools crearán, en caso de que no existan, un directorio llamado `$DATADIR/bin` donde se copiará el binario, y otro llamado `$DATADIR/share` donde se copiarán los directorios y los archivos que le hayamos dicho en el `./Makefile` principal. El compilador **gcc** permite crear la macro comenzando con una comilla simple ('DATADIR=) y encerrando la

expresión $(datadir) entre comillas dobles, para finalizar con otra comilla simple. Tal vez este "juego de comillas" no funcione en otro compilador. El resto de señales son para optimizar el código, avisar de todos los `warnings`, chequear variables declaradas y no usadas, y para verificar que todas las funciones tengan su prototipo. Las dos líneas siguientes indican, respectivamente (y como ya sabrás porque si estás leyendo esto se supone que programas), los directorios donde hay archivos de cabecera necesarios para la compilación (INCLUDE =), y los directorios donde están las librerías con las que hay que enlazar la salida del compilador para terminar de construir el binario (LDADD =).

gtkrc/Makefile.am

```
pkgdata_DATA = almanaque.gtkrc
```

dat/Makefile.am

```
pkgdata_DATA = santoral.xml
```

img/Makefile.am

```
pkgdata_DATA = amq*
```

Los tres archivos anteriores contienen lo mismo: la variable pkgdata_DATA, a la que se le asigna el nombre de los archivos que queremos incluir en el paquete a distribuir y que se encuentren en el mismo directorio que el archivo Makefile.am respectivo.

El siguiente paso es el que me parece más complicado: se trata de crear el archivo `configure.ac`, que contendrá lo necesario para, a su vez, crear el script `configure`. Utilizo las siguientes órdenes:

```
~$ autoscan
~$ cat autoscan.log
~$ rm autoscan.log
~$ mv configure.scan configure.ac
```

Al utilizar por primera vez el comando **autoscan** obtuve los siguientes errores:

```
autom4te: configure.ac: no such file or directory
autoscan: /usr/bin/autom4te failed with exit status: 1
```

Pero no les hice ni caso, dado que ya tenía una primera plantilla para mi `configure.ac` llamada `configure.scan`. A continuación compruebo si hay algo interesante en `autoscan.log` y, como está completamente vacío, lo elimino. Luego renombro el `configure.scan` a `configure.ac`, y me dispongo a continuar con el proceso. En este punto, mi `configure.ac` generado por **autoscan** luce así:

```
#                                               -*- Autoconf -*-
```

```
# Process this file with autoconf to produce a configure script.

AC_PREREQ(2.59)
AC_INIT(FULL-PACKAGE-NAME, VERSION, BUG-REPORT-ADDRESS)
AC_CONFIG_SRCDIR([src/fecha.c])
AC_CONFIG_HEADER([config.h])

# Checks for programs.
AC_PROG_CC

# Checks for libraries.
# FIXME: Replace ‘main’ with a function in ‘-ldl’:
AC_CHECK_LIB([dl], [main])
# FIXME: Replace ‘main’ with a function in ‘-lm’:
AC_CHECK_LIB([m], [main])
# FIXME: Replace ‘main’ with a function in ‘-lpthread’:
AC_CHECK_LIB([pthread], [main])
# FIXME: Replace ‘main’ with a function in ‘-lxml2’:
AC_CHECK_LIB([xml2], [main])
# FIXME: Replace ‘main’ with a function in ‘-lz’:
AC_CHECK_LIB([z], [main])

# Checks for header files.
AC_HEADER_STDC
AC_CHECK_HEADERS([stdlib.h string.h])

# Checks for typedefs, structures, and compiler characteristics.
AC_C_CONST
AC_STRUCT_TM

# Checks for library functions.
AC_FUNC_MALLOC
AC_CHECK_FUNCS([floor memset modf pow])

AC_CONFIG_FILES([Makefile
                 dat/Makefile
                 gtkrc/Makefile
                 img/Makefile
                 src/Makefile])
AC_OUTPUT
```

Efectúo algunas modificaciones y pruebas y, al darlo por concluido, queda así:

```
#                                               -*- Autoconf -*-
# Process this file with autoconf to produce a configure script.

AC_PREREQ(2.59)

# AC_INIT(FULL-PACKAGE-NAME, VERSION, BUG-REPORT-ADDRESS)
AC_INIT()
```

```
almanaque_MAJOR=0
almanaque_MINOR=6
almanaque_MICRO=2

almanaque_VERSION=$almanaque_MAJOR.$almanaque_MINOR.$almanaque_MICRO
AC_SUBST(almanaque_MAJOR)
AC_SUBST(almanaque_MINOR)
AC_SUBST(almanaque_MICRO)
AC_SUBST(almanaque_VERSION)
PACKAGE=almanaque
VERSION=$almanaque_VERSION

AM_INIT_AUTOMAKE($PACKAGE,$VERSION)
AM_CONFIG_HEADER(config.h)

# AC_CONFIG_SRCDIR([src/fecha.c])
# AC_CONFIG_HEADER([config.h])

# Checks for programs.
AC_PROG_CC

AC_CYGWIN
AC_PROG_INSTALL

# Checks for libraries.
# FIXME: Replace ‘main’ with a function in ‘-ldl’:
AC_CHECK_LIB([dl], [main])
# FIXME: Replace ‘main’ with a function in ‘-lm’:
AC_CHECK_LIB([m], [main])
# FIXME: Replace ‘main’ with a function in ‘-lpthread’:
AC_CHECK_LIB([pthread], [main])
# FIXME: Replace ‘main’ with a function in ‘-lxml2’:
AC_CHECK_LIB([xml2], [main])
# FIXME: Replace ‘main’ with a function in ‘-lz’:
AC_CHECK_LIB([z], [main])

# Checks for header files.
AC_HEADER_STDC
AC_CHECK_HEADERS([stdlib.h string.h])

AC_PATH_X
AM_PATH_GTK_2_0(2.0.0, , AC_MSG_ERROR(No instalado GTK+-2.0))
AM_PATH_XML2(2.0.0, , AC_MSG_ERROR(No instalado libxml2))

# Checks for typedefs, structures, and compiler characteristics.
AC_C_CONST
AC_STRUCT_TM

# Checks for library functions.
AC_FUNC_MALLOC
AC_CHECK_FUNCS([floor memset modf pow])

AC_CONFIG_FILES([Makefile
```

```
                    dat/Makefile
                    gtkrc/Makefile
                    img/Makefile
                    src/Makefile])
AC_OUTPUT
```

Los cambios realizados en el archivo `configure.ac` definitivo, los he denotado así: **`en esta forma`** las líneas añadidas, comentadas con '#' las líneas suprimidas y, cuando la línea ha sido modificada, la original la comento *`# en esta forma`* y justo debajo está como queda tras la modificación.

Y sigo con las órdenes:

```
~$ aclocal crea el archivo aclocal.m4
~$ autoheader crea el directorio autom4te.cache y el archivo config.h.in
~$ autoconf crea el script configure
~$ automake -a crea los archivos Makefile.in y stamp-h.in
               así como los enlaces: config.guess, config.subs, COPYING,
               INSTALL, install-sh, missing y mkinstalldirs.
~$ ./configure crea el archivo config.h y los Makefile para cada directorio
               donde haya un Makefile.am, el archivo configure.status
               y el stamp-h.
~$ make compila los fuentes y crea el binario.
~$ su paso a ser root.
~# make install instala el programa y sus archivos de datos.
~# exit dejo de ser root.
~$ almanaque compruebo que funciona.
~$ su vuelvo a ser root.
~# make uninstall compruebo que se desinstala.
~# exit
~$ make clean elimino los archivos .o, innecesarios para el paquete
             a distribuir.
~$ make dist obtengo el paquete almanaque.version.tar.gz.
```

Ya he cumplido el objetivo, pero voy a anotarme en otra sección algunos de los significados y propósitos de todo lo hecho a partir de la creación del `configure.ac`.

6. Tomando nota de lo hecho

Cualquier conocimiento de estas herramientas, comúnmente llamadas autotools, resultará incompleto si se desconoce el lenguaje de programación de macros m4. Sin embargo, pienso que se puede sobrevivir sin ese conocimiento, a no ser que, por ejemplo, se desarrolle una biblioteca de funciones propia y la misma se quiera incluir en los procesos de configuración, o, en el peor de los casos, que se necesite un paquete del que no hay forma de hallar su correspondiente macro para `configure.ac`. Entonces, o se escribe el código necesario para la comprobación directamente en el lenguaje de la shell, o se crea una macro m4. También está el caso de aprenderlo por gusto, que seguramente sería el mío si dispusiese de tiempo para ello. Apuntaré unas sucintas notas sobre el significado de las macros m4 que uso en el

archivo `configure.ac`, más que nada para tenerlas a mano y evitarme el tener que releerlo todo en inglés (que no es mi fuerte) una vez más.

AC_INIT()

Ejecuta el proceso de inicialización para generar el script `configure`. Esta macro puede recibir un argumento opcional, que será el nombre de un archivo del directorio base del proyecto, para asegurarse de que el directorio es el correcto.

AC_SUBST(nombre)

Recibe un argumento, que será el nombre de una variable de la shell. Por ejemplo, para la versión del paquete, lo he hecho así: Creo tres variables, llamadas almanaque_MAJOR=0, almanaque_MINOR=6 y almanaque_MICRO=2, que son "reunidas" en la variable almanaque_VERSION para formar la cadena "0.6.2". (No es obligatorio hacer esto así, simplemente he encontrado esta manera mas cómoda para ir reflejando los sucesivos cambios de versión, pero puede hacerse "todo en uno"). Cuando se produzca un cambio en la versión, sólo tendré que modificar la variable correspondiente en el `configue.ac` para que ese cambio quede reflejado en todo el paquete, código fuente incluido. El archivo config.h, generado por el script `configure`, incluye una serie de macros del tipo **#define** del lenguaje de programación C, entre ellas las pasadas como argumentos en la macro AM_INIT_AUTOMAKE($PACKAGE,$VERSION). Como se puede ver en el `configure.ac`, $PACKAGE contiene la cadena "almanaque" y $VERSION la cadena "0.6.2". Incluyendo en los archivos fuentes (.c o .h) la directiva **#include <config.h>**, tengo a mi disposición el contenido de PACKAGE y VERSION. Por curiosidad, he intentado pasarle otras variables a `config.h`, o modificar los valores de las que ya tiene y que no sean PACKAGE y VERSION, pero no he podido o sabido hacerlo, y no voy a perder otra semana en ello. (Para mí, y dicho sea de paso, una semana de dedicación efectiva a mis aficiones, entre ellas la informática, puede suponer unas siete u ocho horas de tiempo real).

AM_INIT_AUTOMAKE($PACKAGE,$VERSION)

Realiza todo el proceso de inicialización requerido por **automake** e incluye en el archivo `config.h` las variables PACKAGE y VERSION como macros con sus correspondientes valores:

```
#define PACKAGE (valor de $PACKAGE)
#define VERSION (valor de $VERSION)
```

AM_CONFIG_HEADER(config.h)

Puede recibir más de un argumento. Indica que se quiere utilizar el archivo `config.h`, y los demás que se le indiquen, como archivos “de cabecera” (.h) en nuestro código fuente. No tengo idea, ni se me ha ocurrido, ni lo he visto, cómo hacerlo para más de un archivo.

AC_PROG_CC

Comprobaciones sobre el compilador de C.

AC_CYGWIN

Esto lo puse porque tenía en un equipo el "otro" Sistema Operativo con cygwin instalado. Conseguí que compilara, pero ya he eliminado cygwin debido a un formateo obligado del disco que lo contenía. Cygwin es un sistema que permite trabajar como si estuviéramos en Linux estando en "el

otro". En realidad seguimos estando en "el otro", por lo que tampoco es que sea una panacea. Está indicado para quienes se dan de cabeza contra el teclado cada vez que escriben: **C:\> ls**, y cosas por el estilo. ;-)

AC_PROG_INSTALL

Busca un programa `install` y si no lo encuentra asume que el paquete tendrá un `install-sh` disponible.

AC_CHECK_LIB([biblioteca], [función])

Busca en la biblioteca del primer argumento la función cuyo nombre coincida con el segundo. Del nombre de la biblioteca hay que suprimir las letras "lib", y si el nombre de la función es "main", simplemente busca si la biblioteca existe (esto último lo he deducido por mi cuenta, no te fíes mucho).

AC_HEADER_STDC

Define la macro STDC_HEADERS si el sistema tiene los archivos de cabecera del estándar C ANSI.

AC_CHECK_HEADERS([stdlib.h] [string.h])

Comprueba si existen los archivos de cabecera especificados.

AC_PATH_X

Comprueba si está instalado el sistema gráfico.

AM_PATH_GTK_2_0(2.0.2, , AC_MSG_ERROR(No instalado GTK+-2.0))

Comprueba la existencia de gtk+-2.0 o versión superior. Si no la encuentra, lanza el mensaje pasado a AC_MSG_ERROR(mensaje).

AM_PATH_XML2(2.0.0, , AC_MSG_ERROR(No instalado libxml2))

hace lo mismo que la anterior para libxml-2.0 o superior.

AC_C_CONST

Si el compilador de C soporta la palabra clave **const**, esta macro define la macro **const** al string “const”. Si no, la define como una cadena vacía.

AC_STRUCT_TM

Busca la estructura `tm` en el archivo `time.h`.

AC_FUNC_MALLOC

Busca la función `malloc`.

AC_CHECK_FUNCS([floor memset modf pow])

Busca las funciones indicadas en el argumento.

AC_CONFIG_FILES([archivos])

Genera los archivos Makefile que le pasamos como parámetro a partir de los Makefile.am. (Esto también lo he deducido yo, así que lo mismo no es exactamente eso, pero a simple vista es lo que parece) :-)

AC_OUTPUT

Debe estar al final del `configure.ac` y, bueno, parece ser que esta macro es la que hace todo el trabajo "práctico" indicado por las otras. (Ojo, eso es lo que a mí me parece; de nuevo, no te fíes mucho e investiga por tu cuenta si quieres saber toda la verdad, que yo, de momento, me conformo con lo aprendido hasta ahora). :-)

7. Unos cuantos detallitos

No hay que dejar espacios entre el nombre de la macro y el paréntesis de apertura para sus argumentos. (Dos días me llevó averiguar por qué puñetas no funcionaba la copia utilizada en las pruebas para escribir esto, mientras que el original lo hacía perfectamente. ¡Dos días! Bueno, en realidad fueron un par de horas. :-))

Las macros incluidas por mí (es decir, que no proceden directamente del `configure.scan`) que buscan las bibliotecas de las X, de GTK, de XML y otras zarandajas varias que se me haya “olvidado” explicar detalladamente, las he encontrado navegando hábilmente por internet, armado de mucha paciencia y repitiéndome incesantemente: “tranquilo, que ya me dijo mi abuelo que, quien busca, halla” XD.

También habrás observado que en el Makefile de andar por casa, los “flags” ocupan un par de líneas, y en el definitivo, que se supone que es el bueno, ocupan cerca de media pantalla. Esto es lo mismo, simplemente he expandido las expresiones del archivo “casero” y las he pegado en el “profesional”. Así he podido eliminar bibliotecas duplicadas, como por ejemplo `lm`, aunque tal vez **gcc** sea lo suficientemente inteligente como para no enlazarlas dos veces, a mí me gusta más así. O a lo peor es que no me funcionaba de la otra forma, ya no me acuerdo y no voy a cambiarlo.

Hay muchos sitios donde el archivo `configure.ac` figura como `configure.in`. El primer nombre es más moderno que el segundo, pero se aceptan los dos. Por otro lado, he descubierto observando hábilmente, que los archivos con sufijo “.ac” parecen depender de **autoconf** y los que tiene sufijo “.am” parecen depender a su vez de **automake**. Lo digo en serio :-).

La utilización del argumento “-a” al ejecutar **automake** sólo es necesaria la primera vez que se ejecuta para el paquete en cuestión. Cuando se retoca algo en el `configure.ac` una vez que todo el proceso ha sido llevado a cabo, se puede ejecutar **autoreconf** y, después, por si acaso, automake y el resto de la secuencia. Se supone que no sería necesario si sólo se retoca un archivo acabado en “.am”, (pero sí habría que ejecutar **automake**, claro) aunque yo lo hago siempre por si acaso. Antes de ejecutar **aclocal** (o cuando te apetezca, pero es el caso que aparece antes en los manuales que he consultado) se puede

hacer: **ifnames src/*.[ch]** para ver cosas por pantalla. A mí no me suele aclarar nada especial, pero lo mismo a tí sí.

Si te has tomado las molestias de descargar los fuentes del paquete que utilizo como ejemplo, el almanaque, que están en www.fentlinux.com, y comprobarlo con lo te estoy contando, quizás observes que sus archivos no son exactamente iguales a los aquí descritos, incluso cambia el número de versión. Bueno, no te preocupes, que en la próxima versión (algún día), se actualizarán y, mientras tanto, puedes hackear un poco si te apetece, que suele ser divertido.

8. Y hasta aquí hemos llegado

Pues hasta aquí mis apuntes, más o menos puestos en limpio, sobre este asunto. Es un buen truco a seguir este de escribir las cosas como si las fueran a leer otros, así uno se entera mucho mejor de lo que quiso decir cuando haya pasado el tiempo y tenga que volver a consultarlos. Lo recomiendo muy mucho, y ya que se hace, no cuesta nada compartirlo.

Puedes obtener los fuentes del paquete `almanaque` de www.fentlinux.com si estás interesado en contrastar todo esto que aquí cuento con la realidad empírica de su existencia casuística. XD,XD

Espero sepas disculpar los errores que hayas encontrado, y cuenta con mi agradecimiento por tu paciencia.

9. Enlaces (“linkografía”)

Página de GNU Autoconf. (http://www.gnu.org/software/autoconf/)

Página de GNU Automake. (http://www.gnu.org/software/automake/)

Autobook, el libro de las autotools. (http://sources.redhat.com/autobook/autobook/autobook_toc.html)

Programación en el entorno GNOME: Las herramientas a través de un ejemplo. (http://libros.es.gnome.org/librognome/librognome/librognome/x545.html)

Fentlinux (todas las versiones del almanaque). (http://www.fentlinux.com/foros/viewtopic.php?t=2485)

Descarga directa de los fuentes de la última versión publicada del almanaque. (http://www.fentlinux.com/listing/almanaque/almanaque-0.6.0)

Este escrito se dió por terminado el 4 de septiembre de 2005.