

Entendiendo autoconf y automake: Parte 1

Germán Poo Caamaño

[<gpoo@ubiobio.cl>](mailto:gpoo@ubiobio.cl)

Se describe el funcionamiento básico de las herramientas autoconf y automake.

Se otorga permiso para copiar, distribuir y/o modificar este documento bajo los términos de la Licencia de Documentación Libre GNU, versión 1.1 o cualquier versión posterior publicada por la Free Software Foundation. No hay Secciones Invariantes ni Textos de Portada o Contraportada. Puedes consultar una copia de la licencia en <http://www.gnu.org/copyleft/fdl.html>.

Tabla de contenidos

1. [Introducción](#)
2. [aclocal](#)
3. [autoheader](#)
4. [autoconf](#)
5. [automake](#)
6. [Una aplicación de ejemplo: El proyecto "hola mundo"](#)

Tabla de ejemplos

1. [Un archivo configure.in básico](#)

Tabla de contenidos

1. [Introducción](#)
2. [aclocal](#)
3. [autoheader](#)
4. [autoconf](#)
5. [automake](#)
6. [Una aplicación de ejemplo: El proyecto "hola mundo"](#)

1. Introducción

aclocal genera el archivo aclocal.m4, a partir del cual acceder a las macros de Automake, es decir, prepara el camino para usar Automake.

2. aclocal

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 las macros de las cuales dependen.

El archivo acinclude.m4 sirve para especificar macros locales (propias del proyecto) en configure.

El archivo aclocal.m4 solo será generado si se encuentran macros de Automake, es decir, que comiencen con AM_

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

Ejemplo 1. Un archivo configure.in básico

```
AC_PREREQ(2.52)
AC_INIT(miprograma, 0.1)

AM_AUTOMAKE_VERSION(1.7.2)

AC_OUTPUT(Makefile)
```

Después de ejecutar aclocal, se creará el archivo aclocal.m4 bastante pequeño, y sólo es de ejemplo. Si deseamos usar Automake, entonces debemos añadir la macro AM_INIT_AUTOMAKE, para añadir todas las funcionalidades de Automake.

En ese caso, hemos puesto la macro AM_AUTOMAKE_VERSION para indicar que requerimos de la versión 1.7.2 de automake. En este ejemplo, lo hemos empleado, para ver como es el archivo aclocal.m4 generado.

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 emplearemos).

La forma antigua de hacerlo es:

```
AC_INIT
AM_INIT_AUTOMAKE(miprograma, version)
```

La forma nueva de hacerlo es:

```
AC_INIT(miprograma, 0.1)
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.

3. autoheader

Autoheader crea una plantilla con un conjunto de directivas #define que pueden emplearse desde C. Para indicar el nombre del archivo se debe emplear la macro AC_CONFIG_HEADERS. En versiones antiguas, se empleaba AM_CONFIG_HEADER, la cual está obsoleta. La forma de usarlo es:

```
AC_CONFIG_HEADERS(config.h)
```

Con lo cual Autoheader creará el archivo config.h.in y 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.

En nuestro ejemplo, tenemos entonces:

```
AC_PREREQ(2.52)
AC_INIT(miprograma,0.1)

dnl AM_CONFIG_HEADER is obsolete, we use AC_CONFIG_HEADERS instead
AC_CONFIG_HEADERS(config.h)

AM_AUTOMAKE_VERSION(1.7)
AM_INIT_AUTOMAKE

AC_OUTPUT(Makefile)
```

Si deseamos crear el archivo config.h ejecutaremos:

```
$ aclocal
$ autoheader
```

Vemos que sea ha creado el archivo aclocal.m4, config.h.in y autom4te.cache. Los dos últimos, generados por autoheader. El resultado en config.h.in es el siguiente:

```
/* 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
```

Para utilizar Autoheader, requeriremos libtool, el cual se explica más adelante.

4. autoconf

autoconf genera un script de configuración a partir de de las definiciones en configure.in, que finalmente nos permitirá compilar la aplicación de acuerdo a la configuración del sistema (respecto a la ubicación de archivos e implementaciones de funciones, etc.)

Autoconf primero busca en las macros que tiene en forma predefinida y luego en el archivo aclocal.m4, si existe, que anteriormente generamos con aclocal.

Después de ejecutar:

```
$ autoconf
```

Vemos que se ha creado el script 'configure' y que tiene permisos de ejecución.

5. automake

La utilidad 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, comunmente 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, etc. Si se trabaja en varios proyectos, significa, prácticamente, reinventar la rueda en cada uno de ellos, sin mencionar los problemas de portabilidad.

Automake permite automatizar esta labor.

Automake emplea el archivo Makefile.am como fuente, en donde se indica lo esencial a construir, y será necesario por cada uno de los directorios en que necesitemos realizar

alguna tarea. A partir de ello, generará una plantilla llamada Makefile.in.

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

Así, después de eliminar la macro AC_CONFIG_HEADERS (mas adelante se explicará nuevamente) obtendremos el siguiente resultado:

```
$ automake
configure.in: required file `./install-sh' not found
configure.in: required file `./mkinstalldirs' not found
configure.in: required file `./missing' not found
Makefile.am: required file `./COPYING' not found
Makefile.am: required file `./INSTALL' not found
Makefile.am: required file `./NEWS' not found
Makefile.am: required file `./README' not found
Makefile.am: required file `./AUTHORS' not found
Makefile.am: required file `./ChangeLog' not found
```

Nos indica que faltan archivos, que corresponden al de un proyecto. Los 5 primeros, automake los puede proveer porque son estándares en cada aplicación, y salvo que queramos realizar algún cambio, podemos quedarnos con lo que nos sugiera, para ello utilizaremos la opción '--add-missing' de automake. Sin embargo, los 4 últimos son propios de la aplicación y debemos crearlos, considerando que:

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

README, tiene una descripción general del paquete. También es posible indicar instrucciones especiales de instalación.

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.

Adicionalmente, se recomienda incorporar los archivos MAINTAINERS y HACKING, donde:

MAINTAINERS, contiene la lista de nombres de los responsables del proyecto.

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

Intentamos nuevamente:

```
$ touch NEWS README AUTHORS ChangeLog
$ automake --add-missing
configure.in: installing `./install-sh'
configure.in: installing `./mkinstalldirs'
configure.in: installing `./missing'
Makefile.am: installing `./COPYING'
Makefile.am: installing `./INSTALL
```

Es importante revisar el archivo COPYING, puesto que contiene la licencia de uso de la aplicación. Por omisión, será GPL.

Luego, se encuentran todos los archivos preparados para ejecutar el script configure. Hay que notar que, cuando destruyamos nuestra aplicación, no es necesario que la contraparte disponga de alocal, autoheader, autoconf y automake.

6. Una aplicación de ejemplo: El proyecto "hola mundo".

Creemos el ambiente para que el clásico programa "Hola mundo" podamos distribuirlo fácilmente. Para ello, situaremos nuestro programa "hola_mundo.c" en el subdirectorio 'src'. Por lo tanto, necesitaremos dos archivos 'Makefile.am', uno para el directorio raíz del proyecto y otro para el subdirectorio 'src'.

El archivo configure.in quedará de la siguiente forma:

```
AC_INIT(hola-mundo,0.1)

AM_INIT_AUTOMAKE

AC_PROG_CC

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

Hemos indicado que el nombre de nuestro proyecto se llama 'hola-mundo' y la versión inicial es la 0.1. Hemos añadido la macro AC_PROG_CC, la cual nos proveerá de los verificaciones necesarias para asegurarnos la disponibilidad de un compilador de C.

Finalmente indicamos, que requerimos generar los archivos Makefile del directorio raíz y del subdirectorio src.

Nuestro proyecto partirá con los siguiente archivos:

```
$ ls
AUTHORS ChangeLog configure.in Makefile.am NEWS README src
$ ls src
hola_mundo.c Makefile.am
```

El archivo Makefile.am del directorio raíz contendrá:

```
SUBDIRS = src
```

```
EXTRA_DIST = AUTHORS ChangeLog NEWS README
```

En donde se indica, que debe procesar el subdirectorío src. SUBDIRS es una variable genérica, en donde se especifican todos los subdirectoríos que se deben procesar.

Luego, indicamos todos los archivos que son parte extra de la aplicación y que deseamos que se distribuya.

Y el contenido del archivo src/Makefile.am será:

```
bin_PROGRAMS = holamundo
holamundo_SOURCES = hola_mundo.c
```

La variable bin_PROGRAMS indica como se llamará la aplicación final, nuestro archivo binario (ejecutable), si deseamos generar una biblioteca y no un programa ejecutable, emplearemos libexec_PROGRAMS.

Cabe notar que le hemos llamado 'holamundo'. Ese mismo texto se empleará para definir otras variables, como holamundo_SOURCES, holamundo_LDADD, por nombrar las más comunes.

Nuestro programa en C, contiene:

```
#include <stdio.h>

int
main (int argc, char **argv)
{
    printf ("Hola mundo\n");
}
```

Ya se encuentra todo preparado. En breve, ejecutaremos:

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

El detalle es:

```
$ aclocal
$ autoconf
$ automake --add-missing
configure.in: installing `./install-sh'
configure.in: installing `./mkinstalldirs'
configure.in: installing `./missing'
Makefile.am: installing `./COPYING'
Makefile.am: installing `./INSTALL'
src/Makefile.am: installing `./depcomp'
$ ./configure
checking for a BSD-compatible install... /home/gpoo/bin/install -c
checking whether build environment is sane... yes
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
checking for C compiler default output... a.out
checking whether the C compiler works... yes
checking whether we are cross compiling... no
checking for suffix of executables...
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ANSI C... none needed
checking for style of include used by make... GNU
checking dependency style of gcc... gcc
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: executing depfiles commands
$ make
Making all in src
make[1]: Entering directory `/home/gpoo/tmp/autotools/src'
source='hola_mundo.c' object='hola_mundo.o' libtool=no \
depfile='.deps/hola_mundo.Po' tmpdepfile='.deps/hola_mundo.TPo' \
depmode=gcc /bin/sh ./depcomp \
gcc -DPACKAGE_NAME="hola-mundo" -DPACKAGE_TARNAME="hola-mundo" -DPACKAGE_VERSION="0.1" -DPACKAGE_STRING="hola-mundo 0.1" -DPACK
gcc -g -O2 -o holamundo hola_mundo.o
make[1]: Leaving directory `/home/gpoo/tmp/autotools/src'
make[1]: Entering directory `/home/gpoo/tmp/autotools'
make[1]: No se hace nada para `all-am'.
make[1]: Leaving directory `/home/gpoo/tmp/autotools'
$ src/holamundo
Hola mundo
```

Y ya se encuentra nuestro programa compilado y funcionando.

Si queremos distribuirlo, entonces ejecutamos:

```
$ make distckeck
[...]  
=====  
hola-mundo-0.1.tar.gz is ready for distribution  
=====
```

Es decir, nos ha empaquetado un archivo listo para liberar la versión 0.1 de hola-mundo.

Además, podemos ejecutar `make install`, `make uninstall`, `make clean`, `make distclean`.