

Guía de iniciación de Autotools

Autoconf, Automake, libtool, etc.

Versión 1.0

Autor: Luis Díaz Más

<http://plagatux.es>

Esta obra está bajo una licencia Reconocimiento-No comercial 2.5 España de Creative Commons. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-nc/2.5/es/> o envíe una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.

Usted es libre de:



copiar, distribuir y comunicar públicamente la obra



hacer obras derivadas

Bajo las siguientes condiciones:



Reconocimiento. Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciadore (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).



Compartir bajo la misma licencia. Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

- Al reutilizar o distribuir la obra, tiene que hacer referencia al autor y al sitio Web del autor.
- Al reutilizar o distribuir la obra, tiene que dejar bien claro los términos de la licencia de esta obra.
- Alguna de estas condiciones puede no aplicarse si se obtiene el permiso del titular de los derechos de autor
- Nada en esta licencia menoscaba o restringe los derechos morales del autor.

Las fuentes en utilizadas para realizar este documento son las siguientes:

- <http://www.lrde.epita.fr/adl/autotools.html>
- <http://www.gnu.org/software/autoconf/manual/>
- <http://www.gnu.org/software/automake/manual/>
- <http://www.gnu.org/software/libtool/manual/>
- <http://www.bioinf.uni-freiburg.de/~mmann/HowTo/automake.html#doxSteps>

Índice general

1. Introducción	1
1.1. <i>Autoconf</i>	2
1.2. <i>Automake</i>	3
1.3. <i>Libtool</i>	3
2. Ejemplo sencillo: HelloWorld!	4
2.1. Contenido básico	5
2.1.1. <i>Configure.ac</i>	5
2.1.2. <i>Makefile.am</i>	5
2.1.3. <i>src/Makefile.am</i>	5
2.1.4. <i>src/main.c</i>	5
2.2. Preparando el paquete	5
3. Los dos paquetes principales	8
3.1. <i>Autoreconf</i>	8
3.2. Explicación del ejemplo Hello World	9
3.2.1. <i>Configure.ac</i>	9
3.2.2. <i>Makefile.am</i>	10
3.2.3. <i>src/Makefile.am</i>	10
4. Usando <i>Autoconf</i>	12
4.1. La estructura de <i>configure.ac</i>	12
4.1.1. Preludio	12
4.1.2. Comprobaciones de programas	13
4.1.3. Macros de acción de <i>Autoconf</i>	13
4.1.4. Comprobaciones de librerías	14
4.1.5. Comprobaciones de cabeceras	14
4.1.6. Comandos de salida	14
4.2. Descubriendo M4	15
5. Usando <i>Automake</i>	16
5.1. Usando <i>Automake</i> en el <i>configure.ac</i>	16
5.2. Convención para declarar objetivos	16
5.2.1. Ejemplo 1: Compilación de programas	17
5.2.2. Ejemplo 2: Librerías estáticas	18
5.2.3. Ejemplo 3: Librerías de conveniencia	18
5.3. Directorios	18

5.3.1. \$(srcdir) y VPATH	19
5.4. Banderas pre-objetivo	19
5.5. ¿Qué se distribuye?	19
5.6. Condicionales	20
6. Libtool	21
6.1. Librerías compartidas: Un infierno para la portabilidad	21
6.2. Configurando <i>Libtool</i>	21
6.2.1. Un ¡hello world! con <i>Libtool</i>	22
6.3. Contruyendo librerías estáticas o dinámicas	23
6.4. Los programas son realmente scripts	23
6.5. Versionando las librerías	23
7. Integrando doxygen con las <i>Autotools</i>	25

Introducción

Desde 1991 algunos desarrolladores de GNU trabajan en la elaboración de ciertos scripts para facilitar la vida a los desarrolladores de paquetes software GNU. El conjunto de todas estas herramientas es conocido como *Autotools*. Desde entonces el script `configure` es de uso obligatorio en cualquier paquete del proyecto GNU.

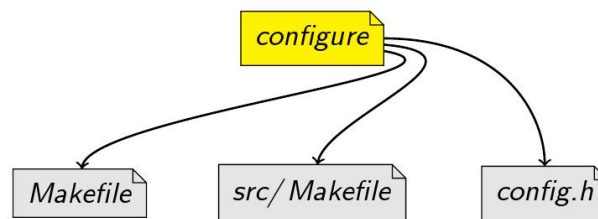


Figura 1.1: Salidas de la ejecución de `configure`

El script `configure` prueba el sistema para el conjunto de funciones, librerías y herramientas requeridas. Cuando se ejecuta genera un fichero `config.h` con todos los `#defines` y ficheros `Makefile` para construir el paquete (ver fig 1.1). Todo esto se realiza para que el procedimiento de instalación de estos paquetes para el usuario final sea tan simple como ejecutar los siguientes comandos:

```
~ % tar zxf paquete.tar.gz
~ % cd paquete
~/paquete % ./configure
...
~/paquete % make
...
~/paquete % sudo make install
...
```

Gracias al uso de las *Autotools* se simplifica tanto el trabajo de los usuarios como el de los mantenedores o desarrolladores de los paquetes. Con la simple ejecución del comando `make dist` se crea el paquete del proyecto software para poder ser redistribuido, mientras que con el comando `make distcheck` además de construirse el paquete se realizan las siguientes comprobaciones:

- Se asegura de que “`make clean`”, “`make distclean`”, y “`make uninstall`” no omiten ficheros.

- Comprueba que las instalaciones **DESTDIR** funcionan.
- Ejecutan la suite de tests (“make check” y “make installcheck”).

Dar salida a un paquete que falla en la ejecución del comando “make distcheck” significa estar dando salida a un paquete que fallará a varios usuarios. Es altamente recomendable probar dicho comando antes de publicar nuestro paquete en cualquier sitio.

Si intentaras imitar el sistema de construcción proporcionado por *Autotools* por tu propia cuenta descubrirás que:

- El sistema de construcción GNU tiene muchas características que nos costarían muchísimo tiempo de implementar por nosotros mismos. Algunos usuarios pueden esperar características que tu no has implementado.
- Implementar este sistema de forma portable es un trabajo difícil y exhaustivo. (Piensa en los scripts de sistemas que desconoces ...).
- Deberás de actualizar tu configuración para seguir los cambios de los estándares de codificación de GNU.

Las *Autotools* de GNU nos proveen de:

- Herramientas para crear el sistema de construcción GNU a partir de simple instrucciones.
- Un lugar donde se realizan correcciones y mejoras continuamente.

A continuación se presenta un breve resumen del cometido de las principales herramientas de *Autotools*.

1.1. *Autoconf*

Autoconf es una herramienta para producir shell scripts que configuran automáticamente paquetes de código fuente para adaptarse a varios tipos de sistemas Posix-like. Los scripts de configuración producidos por *Autoconf* son independientes de *Autoconf* cuando se ejecutan, por lo que sus usuarios no tienen que hacer nada con *Autoconf*.

Los scripts de configuración producidos por *Autoconf* no requieren intervención manual del usuario cuando se ejecutan; normalmente ni requieren que se especifique el tipo del sistema (lo reconoce automáticamente). En vez de ello, comprueban individualmente la presencia de cada una de las características requeridas. Como resultado, llega a un buen acuerdo con sistemas híbridos y/o personalizados.

En cada paquete software en el que se usa *Autoconf*, se crea un script de configuración a partir de una plantilla que lista las características del sistema que el paquete necesitará o usará. Si más tarde necesitamos ajustar el script por cualquier razón, solo necesitaremos cambiarlo en este script y no en todos los subdirectorios del proyecto.

El principal objetivo de *Autoconf* es hacer la vida del usuario más sencilla; hacer más sencilla la vida del desarrollador es solo un objetivo secundario. Dicho de otra manera, el principal objetivo no es hacer la generación del script `configure` de forma automática para los mantenedores; en vez de ello, el objetivo es hacer el `configure` fácil de tratar, portable y predecible para los paquetes suministrados a los usuarios, y en este sentido, *Autoconf* consigue su objetivo.

Autoconf no soluciona todos los problemas relacionados con la realización de paquetes software portables. Para soluciones completas, *Autoconf* debe ser usado en conjunto con otras herramientas de construcción de GNU como *Automake* y *Libtool*. Estas otras herramientas asumen trabajos como la creación de *Makefiles* recursivos con todos los objetivos estándar, linkado de librerías compartidas, etc.

1.2. Automake

Automake es una herramienta para generar automáticamente archivos *Makefile.in* a partir de archivos *Makefile.am*. Cada *Makefile.am* es básicamente una serie de definiciones de variables `make` con reglas que se lanzan ocasionalmente. Los *Makefile.in* generados están ajustados a los estándares de GNU sobre *Makefile*.

El objetivo de *Automake* es eliminar el mantenimiento de *Makefile*. Los ficheros habituales *Automake* contienen simplemente una serie de definiciones de variables. Cada uno de estos ficheros es procesado para crea un *Makefile.in*. Debería haber generalmente un *Makefile.am* por directorio en el proyecto.

Automake implica ciertas restricciones sobre un proyecto: por ejemplo, asume que el proyecto usa *Autoconf*, y fuerza a ciertas restricciones en el contenido del *configure.ac*. Además requiere de `perl` para generar los *Makefile.in*. Sin embargo, las distribuciones creadas con *Automake* no requieren `perl` para ser construidas.

1.3. Libtool

Libtool Simplifica el trabajo del desarrollador para encapsular las dependencias específicas de la plataforma, y la interfaz de usuario en un simple script. Está diseñado tal que la funcionalidad completa de cada tipo de host esté disponible a través de una interfaz genérica, de forma que las peculiaridades desagradables estén ocultas para el programador.

La interfaz consistente de *Libtool* es tranquilizadora ... los usuarios no necesitan leer la oscura documentación para poder tener su paquete fuente construido como una librería dinámica. Solo tendrán que ejecutar el script `configure` y *Libtool* realizará todo el trabajo sucio.

Ejemplo sencillo: HelloWorld!

Los proyectos software deberían estar organizados siempre que se pueda en subcarpetas, por lo que partiremos de esta idea. Idealmente, todos los ficheros fuente deberían estar en una carpeta llamada “src” dentro de la carpeta del proyecto, y con el resto de ficheros (como makefiles, scripts configure, y ficheros de texto) separados en la raíz del proyecto. Los proyectos con varios niveles de carpetas suelen ser llamados proyectos “profundos”. De momento empezaremos con el ejemplo más sencillo posible, un proyecto software con un programa típico de “hola mundo!”.

Los tres ficheros que deben generarse en este proyecto software son los siguientes:

- *configure.ac*.
- *Makefile.in*.
- *src/Makefile.in*.

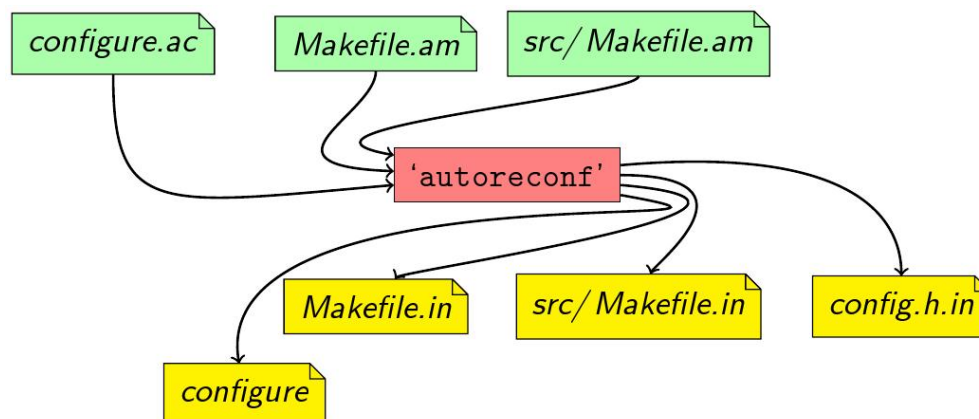


Figura 2.1: Ficheros a generar en un proyecto software

A partir de estos ficheros se genera el script *configure* y los ficheros *.in* con las definiciones y reglas necesarias para posteriormente generar los *Makefile* (ver fig. 2.1). Como veremos a continuación el contenido de estos ficheros para un proyecto sencillo es muy breve y muy similar para diferentes proyectos.

2.1. Contenido básico

En las siguientes subsecciones se muestra el contenido de cada uno de los ficheros básicos y se hará un breve comentario aquellas líneas que lo necesiten. Vamos a crear un directorio que se llame *hello* y vamos a generar los siguientes ficheros dentro de él.

2.1.1. Configure.ac

```
1 AC_INIT([amhello], [1.0], [bug-report@address])
  AM_INIT_AUTOMAKE([-Wall -Werror foreign])
3 AC_PROG_CC
  AC_CONFIG_HEADERS([config.h])
5 AC_CONFIG_FILES([Makefile src/Makefile])
  AC_OUTPUT
```

2.1.2. Makefile.am

```
SUBDIRS = src
```

2.1.3. src/Makefile.am

```
1 bin_PROGRAMS = hello
  hello_SOURCES = main.c
```

2.1.4. src/main.c

```
1 #include <config.h>
2 #include <stdio.h>
4 int main (void)
  {
6     puts ("Hello World!");
    puts ("This is " PACKAGE_STRING ".");
8     return 0;
  }
```

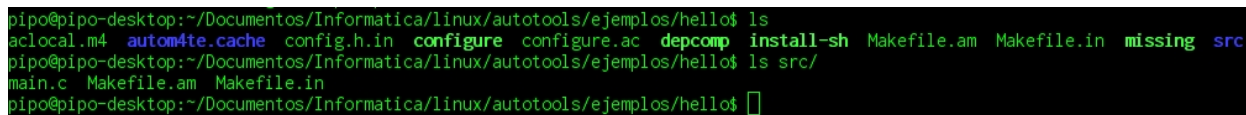
2.2. Preparando el paquete

Una vez que hemos generado dichos ficheros ejecutamos el comando:

```
$ autoreconf --install
configure.ac:2: installing './install-sh'
```

```
configure.ac:2: installing './missing'
src/Makefile.am: installing './depcomp'
```

Y observaremos que se ahora tenemos disponibles nuevos ficheros en nuestros directorios, tal y como muestra la Fig. 2.2.



```
pipo@pipo-desktop:~/Documentos/Informatica/linux/autotools/ejemplos/hello$ ls
aclocal.m4 autom4te.cache config.h.in configure configure.ac depcomp install-sh Makefile.am Makefile.in missing src
pipo@pipo-desktop:~/Documentos/Informatica/linux/autotools/ejemplos/hello$ ls src/
main.c Makefile.am Makefile.in
pipo@pipo-desktop:~/Documentos/Informatica/linux/autotools/ejemplos/hello$
```

Figura 2.2: Ficheros generados tras ejecutar **autoreconf -install**

Los ficheros *Makefile.in*, *src/Makefile.in*, *configure*, y *config.h.in* son las plantillas de configuración esperadas. *aclocal.m4* es un fichero con definiciones para las macros de terceros usadas en *configure.ac*. *depcomp*, *install.sh* y *missing* son herramientas auxiliares usadas durante la construcción. *autom4te.cache* es un directorio que contiene los ficheros caché de *Autotools*.

Ahora podemos ejecutar el script *configure* para configurar el programa.

```
pipo@pipo-desktop: $ ./configure
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking for gcc... gcc
...
checking dependency style of gcc... gcc3
configure: creating ./config.status
config.status: creating Makefile
config.status: creating src/Makefile
config.status: creating config.h
config.status: executing depfiles commands
```

Si no hay problemas, podemos ejecutar el comando *make* para realizar la compilación del programa. Cuando el programa se compile correctamente lo ejecutamos para ver que todo funciona correctamente.

```
pipo@pipo-desktop: $ make
make all-recursive
make[1]: se ingresa al directorio ...
Making all in src
make[2]: se ingresa al directorio ...
gcc -DHAVE_CONFIG_H -I. -I.. -g -O2 -MT main.o -MD -MP -MF
.deps/main.Tpo -c -o main.o main.c
mv -f .deps/main.Tpo .deps/main.Po
gcc -g -O2 -o hello main.o
```

```
...
pipo@pipo-desktop: $ src/hello
Hello World!
This is amhello 1.0.
```

Por último, y como comentamos anteriormente, ejecutando el comando `make distcheck` comprobamos si nuestro paquete está listo para distribuirse.

```
pipo@pipo-desktop: $ make distcheck
...
=====
amhello-1.0 archives ready for distribution:
amhello-1.0.tar.gz
=====
```

Los dos paquetes principales

Como se ha podido observar, el proceso de configuración y generación de un paquete es realmente sencillo haciendo uso de las *Autotools*. Esta simplicidad se la debemos principalmente a *GNU Autoconf* y *GNU Automake*. A continuación se muestran los principales comandos y características que nos ofrecen dichos programas.

■ Autoconf

- **autoconf**: Crea el `configure` a partir del `configure.ac`.
- **autoheader**: Crea el `config.h.in` a partir del `configure.ac`.
- **autoreconf**: Ejecuta todas las herramientas en el orden adecuado.
- **autoscan**: Examina los fuentes en búsqueda de problemas comunes de portabilidad, y de macros perdidas en `configure.ac`.
- **autoupdate**: Actualiza las macros obsoletas en `configure.ac`.
- **ifnames**: Reúne los identificadores de todas las directivas `#if/#ifdef/...`
- **autom4te**: El corazón de *Autoconf*. Maneja el lenguaje M4 e implementa las características usadas por la mayoría de las herramientas de arriba. Útil para crear algo más que un simple `configure`.

■ Automake

- **automake**: Crea los `Makefile.in` a partir de los `Makefile.am` y `configure.ac`.
- **aclocal**: Examina `configure.ac` para el uso de macros de terceros, y reúne las definiciones en `aclocal.m4`.

3.1. Autoreconf

Ha supuesto uno de los mayores avances dentro de las *Autotools*. La figura 3.1 muestra los comandos que se tenían que ejecutar anteriormente a que apareciera este comando, mientras que la figura 3.2 muestra el panorama actual.

Autoreconf nos ofrece las siguientes ventajas:

- No tienes que recordar la forma de interacción de todas las herramientas.
- Usa `autoreconf -install` para configurar el paquete inicialmente.

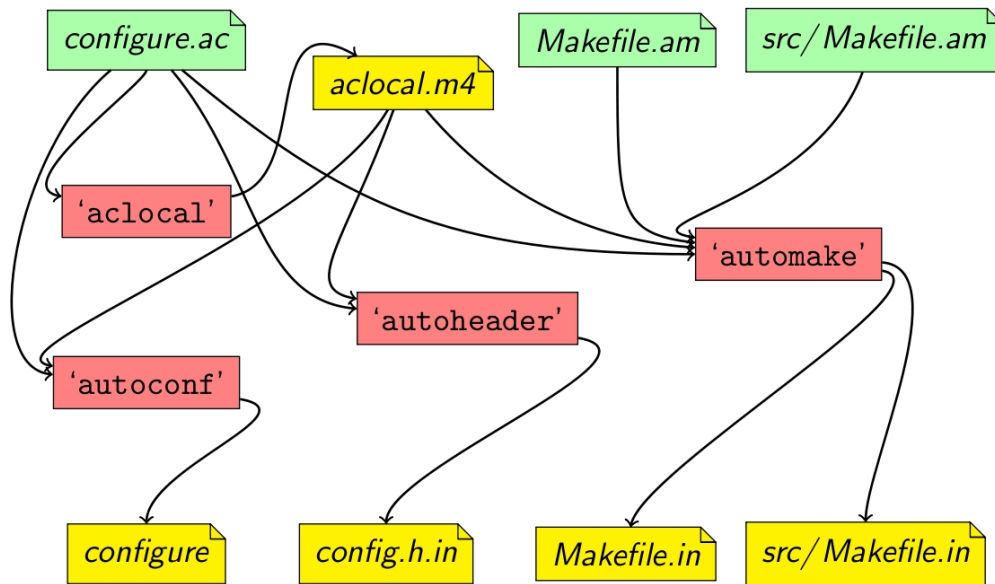


Figura 3.1: Comandos a ejecutar antes de la aparición de autoreconf

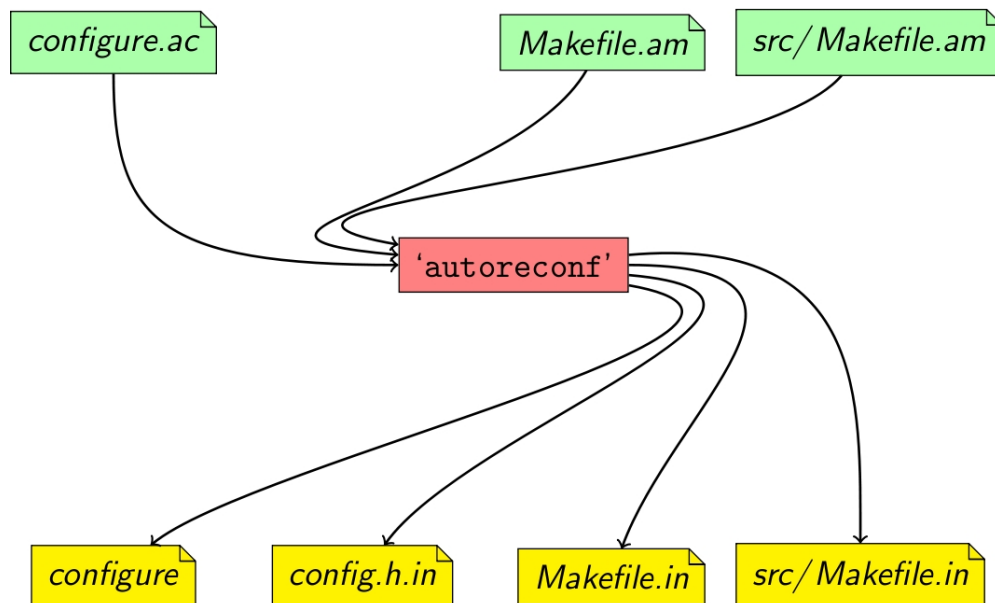


Figura 3.2: Comando a ejecutar con la aparición de autoreconf

- Contar con las reglas de reconstrucción (salida de *Makefiles* para volver a ejecutar las herramientas adecuadas cuando ejecutes algún fichero fuente).
- Solo necesitas una idea somera del propósito de cada herramienta para comprender los errores. (¿Qué herramienta se queja y sobre qué?)

3.2. Explicación del ejemplo Hello World

3.2.1. Configure.ac

```

1 AC_INIT([amhello], [1.0], [bug-report@address]) #Inicializa autoconf.
  #Especifica nombre del paquete, numero de version y direccion para
3  #reportar errores.
AM_INIT_AUTOMAKE([--Wall --Werror foreign]) #Inicializa automake.
5  #Activa los alarmas de automake y las reporta como errores.
  #foreign hace que se ignoren algunos estandares de codificacion GNU
7 AC_PROG_CC #Comprueba la existencia de un compilador de C
AC_CONFIG_HEADERS([config.h]) #Declara a config.h como cabecera de
9  #salida
AC_CONFIG_FILES([Makefile src/Makefile]) #Declara a Makefile y
11 #src/Makefile como ficheros de salida
AC_OUTPUT #Da salida a todos los ficheros declarados

```

A continuación se muestran las salidas con la opción foreign y sin ella.

- Con foreign

```

$ autoreconf --install
configure.ac:2: installing './install-sh'
configure.ac:2: installing './missing'
src/Makefile.am: installing './depcomp'

```

- Sin foreign

```

$ autoreconf --install
configure.ac:2: installing './install-sh'
configure.ac:2: installing './missing'
src/Makefile.am: installing './depcomp'
Makefile.am: installing './INSTALL'
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
Makefile.am: installing './COPYING'
autoreconf: automake failed with exit status: 1

```

3.2.2. Makefile.am

```
SUBDIRS = src #Construccion recursiva en src/
```

3.2.3. src/Makefile.am

```

1 bin_PROGRAMS = hello #Construccion de programas. Al tener el prefijo
  #bin, los programas se instalaran en bindir. Solo hay un programa
3  #a construir: hello.
hello_SOURCES = main.c #Para crear el programa hello, solo se compila
5  #el fichero main.c

```

En la Tabla 3.1 se puede apreciar como se organiza el sistema de ficheros habitualmente al trabajar en este tipo de proyectos software.

Directory variable	Default value
<code>prefix</code>	<code>/usr/local</code>
<code>exec-prefix</code>	<code>prefix</code>
<code>bindir</code>	<code>exec-prefix/bin</code>
<code>libdir</code>	<code>exec-prefix/lib</code>
...	
<code>includedir</code>	<code>prefix/include</code>
<code>datarootdir</code>	<code>prefix/share</code>
<code>datadir</code>	<code>datarootdir</code>
<code>mandir</code>	<code>datarootdir/man</code>
<code>infodir</code>	<code>datarootdir/info</code>
...	

Tabla 3.1: Jerarquía del sistema de ficheros estándar

Usando Autoconf

Autoconf es un procesador de macros. Convierte el *configure.ac*, que es un shell script que hace uso de instrucciones de tipo macro, en el *configure*, un shell script completo. *Autoconf* ofrece varias macros para realizar comprobaciones comunes de configuración. No es extraño tener un *configure.ac* sin constructores de shell, usando solo macros. El procesador real de macros es actualmente GNU M4. *Autoconf* ofrece una infraestructura encima de GNU M4, además del conjunto de macros.

4.1. La estructura de *configure.ac*

Podemos partir siempre de una plantilla como la siguiente:

```

1 # Preludio .
AC_INIT([amhello], [1.0], [bug-report@address])
3 # Comprobaciones para programas
# Comprobaciones para librerías
5 # Comprobaciones para ficheros de cabecera
# Comprobaciones para typedefs, structures, y
7 # características del compilador
# Comprobaciones para funciones de librería
9 # Ficheros de salida
AC_CONFIG_FILES([FILES])
11 AC_OUTPUT

```

4.1.1. Preludio

AC_INIT(PACKAGE, VERSION, BUG-REPORT-ADDRESS)

Inicialización obligatoria de *Autoconf*.

AC_PREREQ(VERSION)

Requerir una versión mínima de *Autoconf*. P.e. `AC_PREREQ(2.61)`

AC_CONFIG_SRCDIR(FILE)

Una comprobación de seguridad. `FILE` debe ser un fichero fuente distribuido, y esto se asegura de que `configure` no se ejecuta desde un espacio exterior.

P.e. `AC_CONFIG_SRCDIR([src/main.c])`.

AC_CONFIG_AUX_DIR(DIRECTORY)

Los scripts auxiliares como `install-sh` y `depcomp` deben encontrarse en el `DIRECTORY`. P.e. `AC_CONFIG_AUX_DIR([build-aux])`. Si esto no se incluye se toma por defecto el directorio actual.

4.1.2. Comprobaciones de programas**AC_PROG_CC, AC_PROG_CXX, AC_PROG_F77, ...**

Comprobaciones del compilador. (Maneja la búsqueda cruzada de compiladores si es necesario)

AC_PROG_SED, AC_PROG_YACC, AC_PROG_LEX, ...

Busca buenas implementaciones y configure `$SED`, `$YACC`, `$LEX`, etc.

AC_CHECK_PROGS(VAR, PROGS, [VAL-IF-NOT-FOUND])

Define `VAR` para el primer `PROGS` encontrado, o `VAL-IF-NOT-FOUND` en caso contrario.

```

1 AC_CHECK_PROGS([TAR], [tar gtar], [:])
   if test "$TAR" = :; then
3     AC_MSG_ERROR([Este paquete necesita tar.])
   fi

```

... y muchos más

4.1.3. Macros de acción de *Autoconf***AC_MSG_ERROR(ERROR-DESCRIPTION, [EXIT-STATUS])**

Imprime `ERROR-DESCRIPTION` (En `config.log`) aborta `configure`.

AC_MSG_WARN(ERROR-DESCRIPTION)

Lo mismo, pero no aborta.

AC_DEFINE(VARIABLE, VALUE, DESCRIPTION)

Imprime lo siguiente a `config.h`.

```

2 /* DESCRIPTION */
   #define VARIABLE VALUE

```

AC_SUBST(VARIABLE, [VALUE])

Define `$(VARIABLE)` como `VALUE` en `Makefile`.

```
AC_SUBST([FOO], [foo])
```

```

1 FOO=foo
   AC_SUBST([FOO])

```

```

2 AC_SUBST([FOO])
   FOO=foo

```

Todos son equivalentes.

4.1.4. Comprobaciones de librerías

`AC_CHECK_LIB(LIBRARY, FUNCT, [ACT-IF-FOUND], [ACT-IF-NOT])`

Comprueba si la librería `LIBRARY` existe y contiene la función `FUNCT`. Ejecuta `ACT-IF-FOUND` si así ocurre, `ACT-IF-NOT` en caso contrario.

```
2 AC_CHECK_LIB([efence], [malloc], [EFENCELIB--leference])
AC_SUBST([EFENCELIB])
```

... deberíamos usar más tarde `$(EFENCELIB)` en la regla de linkado.

Si `ACT-IF-FOUND` no se establece y la librería se encuentra, `AC_CHECK_LIB` hará que `LIBS="-lLIBRARY $LIBS"` y `#define HAVE_LIBLIBRARY`.

(*Automake* usa `$LIBS` para el linkado de todas las cosas.)

4.1.5. Comprobaciones de cabeceras

`AC_CHECK_HEADERS(HEADERS...)`

Comprobaciones de `HEADERS` y `#define HAVE_HEADER_H` para cada cabecera encontrada.

```
2 AC_CHECK_HEADERS([sys/param.h unistd.h])
AC_CHECK_HEADERS([wchar.h])
```

`#define HAVE_SYS_PARAM_H, HAVE_UNISTD_H, y HAVE_WCHAR_H.`

```
2 #if HAVE_UNISTD_H
# include <unistd.h>
#endif
```

`AC_CHECK_HEADER(HEADER, [ACT-IF-FOUND], [ACT-IF-NOT])`

Comprueba solo una cabecera

4.1.6. Comandos de salida

`AC_CONFIG_HEADERS(HEADERS...)`

Crea `HEADER` por cada `HEADER.in`. Use solo una de dichas cabeceras a no ser que conozca lo que está haciendo (*autoheader* crea `HEADER.in` solo para el primer `HEADER`). `HEADERS` contiene definiciones hechas con `AC_DEFINE`.

```
1 AC_CONFIG_HEADERS([config.h:config.hin])
```

Crearé `config.h` a partir de `config.h.in` (*DJGPP* suporta solo 1 punto).

`AC_CONFIG_FILES(FILES...)`

Crea `FILE` por cada `FILE.in`. `FILES` contiene definiciones hechas con `AC_SUBST`.

```
1 AC_CONFIG_FILES([Makefile sub/Makefile script.sh:script.in])
```

Automake crea `FILE.in` por cada `FILE` que tiene un `FILE.am`.

Esto también sirve para procesar ficheros que no son *Makefiles*.

4.2. Descubriendo M4

Sería muy útil conocer a fondo el lenguaje de programación M4. Podéis encontrar más detalles en <http://www.lrde.epita.fr/adl/autotools.html>

Usando *Automake*

Automake ayuda a la creación de *Makefile* portables y que cumplen el estándar GNU generando los *Makefile.in* a partir de los *Makefile.am*. Los *Makefile.am* siguen someramente la misma sintaxis que los *Makefile*, sin embargo, usualmente solo contienen definiciones de variables. *Automake* crea reglas de construcción a partir de dichas definiciones. Además puedes seguir añadiendo reglas *Makefile* a los ficheros *Makefile.am*, ya que *Automake* las preservará en la salida.

5.1. Usando *Automake* en el *configure.ac*

En la Sec. 2.1.1 aparece la macro que inicializa *Automake*, y que comprueba la presencia de las herramientas necesarias para su correcto funcionamiento. Entre las diferentes opciones que podemos incluir en `AM_INIT_AUTOMAKE` tenemos las siguientes:

```
1 AM_INIT_AUTOMAKE([OPTIONS])
```

- **-Wall**: Activa todas las alarmas (warnings).
- **-Werror**: Reporta las alarmas como errores.
- **foreign**: Relaja ciertos requerimientos del estándar GNU.
- **1.10.1**: Requiere una versión mínima de *Automake*.
- **dist-bzip2**: También crea archivos tar.bz2 durante `make dist` y `make distcheck`.
- **tar-ustar**: Crea archivos tar usando el formato ustar.

Mediante `AC_CONFIG_FILES` *Automake* crea un fichero `FILE.in` por cada `FILE.am`.

```
1 AC_CONFIG_FILES([Makefile sub/Makefile])
```

5.2. Convención para declarar objetivos

La declaración de objetivos en los ficheros *Makefile.am* sigue la siguiente forma:

```
1 option_where_PRIMARY = targets ...
```

Donde `PRIMARY` puede ser:

- **PROGRAMS**: Programas.
- **LIBRARIES**: Librerías estáticas.
- **LTLIBRARIES**: Librerías dinámicas.
- **HEADERS**: Cabeceras.
- **SCRIPTS**: Scripts.
- **DATA**: Datos adicionales.

`where` puede tomar los valores:

- **bin_**: Programas ejecutables que se instalan en $\$(bindir)$
- **lib_**: Librerías que se instalan en $\$(libdir)$
- **custom_**: Cosas que se instalan en $\$(customdir)$, donde dicho directorio debes definirlo previamente.
- **noinst_**: No instalable.

por último, `option` puede ser:

- **dist_**: Se distribuyen los `targets` (si no se especifica esta es la opción por defecto).
- **nodist_**: No se distribuyen.

5.2.1. Ejemplo 1: Compilación de programas

```
1 #Makefile .am
  bin_PROGRAMS = foo run-me
3 foo_SOURCES = foo.c foo.h print.c print.h
  run_me_SOURCES = run.c run.h print.c
```

Los programas `foo` y `run-me` se instalarán en $\$(bindir)$. Un detalle curioso, es que los caracteres no alpha-numéricos (como el guión o signo menos “-”) son siempre mapeados a guiones bajos “_”. *Automake* calcula automáticamente mediante la lista de ficheros fuente indicados, los objetos a construir y los linkados a realizar. Las cabeceras no se compilan, pero se indican aquí para que sean distribuidas.

5.2.2. Ejemplo 2: Librerías estáticas

```

#Makefile .am
2 lib_LIBRARIES = libfoo.a libbar.a
  libfoo_a_SOURCES = foo.c privfoo.h
4 libbar_a_SOURCES = bar.c privbar.h
  include_HEADERS = foo.h bar.h

```

Para construir librerías estáticas debemos añadir al *configure.ac* la macro `AC_PROG_RANLIB`. Dichas librerías se instalarán en `$(libdir)`. Las librerías siempre deben seguir la nomenclatura `lib*.a`. Las cabeceras públicas (línea 4) se instalarán en `$(includedir)`. Las cabeceras privadas (ficheros `.h` en líneas 2 y 3) no se instalarán como ficheros fuente ordinarios.

5.2.3. Ejemplo 3: Librerías de conveniencia

Son aquellas librerías que solo se usan a la hora de construir nuestro paquete.

```

#lib/Makefile .am
1 noinst_LIBRARIES = libcompat.a
3 libcompat_a_SOURCES = xalloc.c xalloc.h

```

```

#src/Makefile .am
1 bin_PROGRAMS = foo run-me
  foo_SOURCES = foo.c foo.h print.c print.h
3 run_me_SOURCES = run.c run.h print.c
5 LDADD=../lib/libcompat.a
  AM_CPPFLAGS = -I\$(srcdir)/../lib
7 #run_me_LDADD = ../lib/libcompat.a
  #run_me_CPPFLAGS = -I\$(srcdir)/../lib

```

`LDADD` se añade cuando se enlazan todos los programas. `AM_CPPFLAGS` contiene banderas para el preprocesador adicionales. Las dos últimas líneas comentadas hacen referencia a que se pueden usar `LDADD` y `CPPFLAGS` para programas independientes.

5.3. Directorios

Debemos tener un *Makefile* (uno por cada *Makefile.am*) por directorio y todos ellos deben estar declarados en *configure.ac*.

```

#configure .ac
2 AC_CONFIG_FILES([Makefile lib/Makefile src/Makefile
  src/dira/Makefile src/dirb/Makefile])

```

Cada *Makefile.am* debe establecer el orden en que sus directorios se recorren usando la variable `SUBDIRS`.

```

# Makefile .am
1 SUBDIRS = lib src

```

Si no especificamos el directorio actual mediante un punto “.”, este se compilará implícitamente después de los subdirectorios.

```

2 # src/Makefile.am
  SUBDIRS = . dira dirb

```

5.3.1. \$(srcdir) y VPATH

Es importante recordar que un fichero fuente no está necesariamente en el directorio actual. Hay dos árboles gemelos: el **build tree** y el **source tree**.

- *Makefile* y los ficheros objeto están en el **build tree**.
- *Makefile.in* y *Makefile.am* y los ficheros fuente están en el **source tree**.
- Si se ejecuta `./configure` en el directorio actual, los dos árboles serán el mismo.

Es posible que necesitemos utilizar la variable `srcdir` cuando especifiquemos banderas para herramientas, o escribamos comandos propios. P.e., para indicarle al compilador que incluya cabeceras de `/dir`, tendremos que escribir `-I$(srcdir)/dir`.

5.4. Banderas pre-objetivo

Asumamos que `foo` es un programa o librería:

- `foo_CFLAGS`: Banderas adicionales para el compilador de C.
- `foo_CPPFLAGS`: Banderas adicionales para el preprocesador (`-Is` y `-Ds`).
- `foo_LDADD`: Linkado de objetos adicional, `-ls` y `-Ls` (si `foo` es un programa)
- `foo_LIBADD`: Linkado de objetos adicional, `-ls` y `-Ls` (si `foo` es una librería)
- `foo_LDFLAGS`: Banderas de linkado adicionales.

El valor por defecto para `foo_XXXFLAGS` es `$(AM_XXXFLAGS)`.

5.5. ¿Qué se distribuye?

Los comandos `make dist` y `make distcheck` crearán un tarball que contendrá:

- Todos los ficheros fuente declarados usando `..._SOURCES`.
- Todas las cabeceras declaradas usando `..._HEADERS`.
- Todos los scripts declarados usando `dist_..._SOURCES`.
- Todos los datos declarados usando `dist_..._SOURCES`.
- Ficheros comunes como `ChangeLog`, `NEWS`, etc.
- Ficheros o directorios extra listados en `EXTRA_DIST`.

5.6. Condicionales

El uso de condicionales nos permite realizar paquetes incondicionales con construcciones condicionales.

```
#Programas condicionales
2 bin_PROGRAMS = foo
  if WANT_BAR
4   bin_PROGRAMS += bar
  endif
6 foo_SOURCES = foo.c
  bar_SOURCES = bar.c
```

```
1 #Fuentes condicionales
  bin_PROGRAMS = foo
3  foo_SOURCES = foo.c
  if WANT_BAR
5   foo_SOURCES += bar
  endif
```

En estos ejemplos se dan las siguientes peculiaridades:

- `bar` es construido si `WANT_BAR` es verdadero. (1^{er} ejemplo)
- `bar.o` es linkado en `foo` si `WANT_BAR` es verdadero. (2^o ejemplo)
- `foo.c` y `bar.c` son distribuidos a pesar de `WANT_BAR`.
- `WANT_BAR` se debe declarar y se le debe dar un valor en *configure.ac*.

```
#configure.ac
2 AC_CHECK_HEADER([bar.h], [use_bar=yes])
  AM_CONDITIONAL([WANT_BAR], [test "$use_bar" = yes])
```

6.1. Librerías compartidas: Un infierno para la portabilidad

Cada plataforma tiene su propia forma de implementar la idea de librerías compartidas o dinámicas, y por lo tanto hay varias herramientas necesarias para construir dichas librerías. *Libtool* delega a dichas herramientas específicas de cada plataforma y le da al desarrollador un conjunto simple de opciones para realizar todo el trabajo.

Algunos de los formatos de librerías dinámicas existentes son:

- libhello.so
- libhello.dll
- libhello.sl
- libhello.dylib

Mediante *Libtool* disponemos de un nuevo formato que abstrae a todos los demás: `libhello.la` (libtool archive). Un script es el encargado de traducir las operaciones involucradas con los ficheros `lib*.la` a operaciones correctas para el sistema actual usando la librería real. En *Makefile.am* simplemente tenemos que crear y linkar contra los ficheros `*.la`.

6.2. Configurando *Libtool*

El primer paso es indispensable, debemos llamar a la macro `AC_PROG_LIBTOOL` en *configure.ac*. Después, para declarar una librería dinámica mediante *Libtool* usaremos `_LTLIBRARIES` en los *Makefile.am*. Por último, podremos usar `_LDADD` para linkar contra los archivos locales generador con *Libtool*.

```
1 #Makefile .am
2 lib_LTLIBRARIES = libfoo.la
3 libfoo_la_SOURCES = foo.c foo.h etc.c
4
5 bin_PROGRAMS = runme
```

```

runme_SOURCES = main.c
7 runme_LDADD = libfoo.la

```

6.2.1. Un ¡hello world! con *Libtool*

```

1 /* lib/say.c */
#include <config.h>
3 #include <stdio.h>

5 void say_hello (void)
{
7     puts ("Hello World!");
    puts ("This is " PACKAGE_STRING ".");
9 }

```

```

1 /* lib/say.h */
void say_hello (void);

```

```

/* src/main.c */
2 #include "say.h"

4 int main (void)
{
6     say_hello ();
    return 0;
8 }

```

```

#lib/Makefile.am
2 lib_LTLIBRARIES = libhello.la
libhello_la_SOURCES = say.c say.h

```

```

1 #src/Makefile.am
AM_CPPFLAGS = -I(srcdir)/../lib/
3 bin_PROGRAMS = hello
hello_SOURCES = main.c
5 hello_LDADD = ../lib/libhello.la

```

```

1 #Makefile.am
SUBDIRS = lib src

```

```

AC_INIT([amhello], [2.0], [bug-report@address])
2 AC_CONFIG_AUX_DIR([build-aux])
AM_INIT_AUTOMAKE([--Wall --Werror foreign])
4 AC_PROG_LIBTOOL
AC_PROG_CC
6 AC_CONFIG_HEADERS([config.h])
AC_CONFIG_FILES([Makefile lib/Makefile src/Makefile])
8 AC_OUTPUT

```

6.3. Contruyendo librerías estáticas o dinámicas

Por defecto, se construyen tanto las librerías estáticas como las dinámicas. Esto se puede cambiar usando las siguientes macros:

- `AC_DISABLE_SHARED`: No construye las librerías dinámicas.
- `AC_DISABLE_STATIC`: No construye las librerías estáticas.

El usuario puede sobrescribir la configuración usando las opciones del `configure`:

- `-enable-share`: Construye librerías dinámicas.
- `-disable-share`: No lo hace.
- `-enable-static`: Construye librerías estáticas.
- `-disable-static`: No lo hace.

6.4. Los programas son realmente scripts

En el ejemplo anterior, `src/hello` puede ser un script dependiendo de la configuración de *Libtool*. El binario real se construye siempre pero *Libtool* lo oculta en el **built tree**. El script ejecuta el binario real y se organiza de tal forma que encuentra las librerías que aún no han sido instaladas. De esta forma `src/hello` puede ejecutarse por ejemplo, en un entorno de pruebas.

Al ser `src/hello` un script, este no puede ser directamente depurado con una herramienta como `gdb`

```
~/prueba % gdb -q src/hello
"src/hello": not in executable format: File format not recognized
(gdb)
```

Tenemos que añadir el siguiente prefijo: `libtool -mode=execute`

```
~/prueba % libtool --mode=execute gdb -q src/hello
(gdb)
```

6.5. Versionando las librerías

Dando diferentes versiones a las librerías podemos hacer que coexistan varias versiones de la misma librería. Esto asegura que los programas usen la librería que implementen la interfaz que requieran. Las interfaces son identificadas usando enteros. Un programa recuerda los números de interfaz de las librerías contra las que está linkado. Es importante tener en cuenta que los números de interfaz no son los números de lanzamiento. Tenemos las siguientes tripletas de valores:

- **CURRENT**: La última interfaz implementada.

- **REVISION**: El número de implementación de **CURRENT**. Está relacionado con el número de bugs corregidos.
- **AGE**: El número de interfaces implementadas menos uno.

Dichos números deben ser especificados usando `-version-info`.

```
#lib/Makefile.am
2 lib_LTLIBRARIES = libhello.la
  libhello_la_SOURCES = say.c say.h
4 libhello_la_LDFLAGS = -version-info CURRENT:REVISION:AGE
```

Integrando doxygen con las *Autotools*

Doxygen es un sistema de documentación para C++, C, Java, Objective-C, Python, IDL, algunas extensiones de PHP, C# y D. Proporciona una maquinaria genérica para generar un proyecto de documentación bien formateado directamente del código fuente y de los datos externos. La documentación resultante está disponible en varios formatos, p.e. pdf, html, chm, etc.

Para añadir el soporte de *Doxygen* a las *Autotools*, voy a explicar el ejemplo suministrado por la web de [Martin Mann](#). Con dicho ejemplo se proporciona una serie de parámetros que se pueden añadir al `configure` para generar de forma rápida y sencilla diferentes tipos de documentación mediante *Doxygen*. Dicho ejemplo puede ser descargado de [aquí](#).

Los pasos a realizar para integrar *Doxygen* con las *Autotools* en nuestro proyecto son los siguientes:

- Documentar nuestro código siguiendo la guía de documentación doxygen.
- Crear un soporte de *Autotools* para nuestro proyecto, tal y como hemos ido explicando a lo largo de este documento.
- Descargar el [Doxample-v2.1.tar.gz](#) en la carpeta raíz de nuestro proyecto.
- Incluir la línea `'include $(top_srcdir)/aminclude.am'` al fichero `Makefile.am` de nuestro directorio raíz.
- Añadir la línea `'DX_INIT_DOXYGEN($PACKAGE_NAME, MYDOXYGENCONFIG)'` a nuestro `configure.ac`. (`$PACKAGE_NAME` se establece automáticamente mediante `AC_INIT` representando al nombre del proyecto, y `MYDOXYGENCONFIG` es el fichero de configuración *Doxygen* que viene con *Doxample* para nuestro proyecto. `'DX_INIT_DOXYGEN` acepta un tercer parámetro opcional para indicar el directorio de salida para la documentación, que por defecto es `doxygen-doc`).
- Por cada modo de salida puedes establecer su soporte mediante las funciones `DX_HTML_FEATURE`, `DX_CHM_FEATURE`, `DX_CHI_FEATURE`, `DX_MAN_FEATURE`, `DX_RTF_FEATURE`, `DX_XML_FEATURE`, `DX_PDF_FEATURE`, `DX_PS_FEATURE` añadiendo la línea `DX_XXX_FEATURE(ON)` o `DX_XXX_FEATURE(OFF)` a tu `configure.ac` antes de la línea `DX_INIT_DOXYGEN(..)`.

Una vez generado el `configure`, podemos ver las opciones relacionadas con doxygen ejecutando `'./configure -help'`. Puedes usar dichas opciones para determinar el tipo de

salida por defecto de doxygen (p.e. '-disable-doxygen-doc' o '-enable-doxygen-pdf'). Con 'make doxygen-doc' generaras finalmente la documentación *Doxygen* determinada por el fichero de configuración *Doxygen*. Toda la documentación será situada en un nuevo subdirectorio llamado por defecto doxygen-doc.