## <https://www.gnu.org/software/automake/manual/html_node/Hello-World.html#Hello-World>

<https://wiki.embeddedarm.com/wiki/Jessie_armhf_Cross_Compile>

## Cấu trúc chương trình:

- Thư mục helloworld:

- configure.ac

- Makefile.am

- README

- Thư mục src

- Makefile.am

- main.c

- Tạo file main.c trong thư mục src (sau này có thể có thêm thư mục man cho các man pages, thư mục data để chứa data files v.v.).

- Tạo file README ghi một số documentation cho chương trình.

- Tạo file Makefile.am trong thư mục src

- Tạo file Makefile.am trong thư mục helloworld

- Tạo file configure.ac trong thư mục helloworld để chứa các lệnh Autoconf để tạo script configure

## Chạy chương trình:

- tại thư mục helloworld, chạy lệnh autoreconf --install

- install: ./install-sh, ./missing, ./compile, ./depcomp

- chạy lệnh ./configure

- chạy lệnh make

- chạy chương trình: src/hello

- chạy lệnh make distcheck

- Lệnh autoreconf chỉ chạy lần đầu khi GNU Build System chưa tồn tại. Sau đó mỗi lần sửa Makefile.am, configure.ac chỉ cần chạy lệnh make

- autoreconf là một script để gọi autoconf (tạo configure từ configure.ac), automake (tạo Makefile.in từ Makefile.am và configure.ac).

### Các option của lệnh autoreconf:

'autoreconf' chạy các lệnh 'autoconf', 'autoheader', 'aclocal', 'automake', 'libtoolize' và 'autopoint' (khi cần) để update GNU Build System tại các thư mục được chỉ định.

- --help, -h: in ra các option.

- --version, -V: in ra version của Autoconf

- --verbose, -v: in ra tên của từng thư mục mà autoreconf sẽ kiểm tra và chạy dòng lệnh trên đó.

- --debug, -d: không bỏ đi các file tạm.

- --force, -f: viết lại các script configure và header configuration mà mới hơn các file đầu vào của chúng (configure.ac, aclocal.m4).

- --install, -i: cài đặt các file hỗ trợ bị thiếu trong package. Mặc định, các file sẽ được copy sau đó thay đổi với --symlink. Nếu thành công lệnh này sẽ gọi đến các lệnh automake --add-missing, libtoolize, autopoint v.v.

- --no-recursive: không rebuild các file trong thư mục con để cấu hình.

- --symlink, -s: khi sử dụng với --install, cài đặt liên kết tượng trưng tới các file bị thiếu thay vì copy chúng.

- --make, -m: khi các thư mục đã đưọc cấu hình, update cấu hình bằng các chạy lệnh './config.status --recheck' và './config.status' sau đó chạy lệnh 'make'.

- --include=dir, -I dir: Thêm thư mục dir vào cuối đường dẫn include. Khi có quá nhiều lệnh gọi lại thì chuyển sang aclocal và autoheader nội bộ.

- --prepend-include=dir, -B dir: Thêm thư mục dir vào cuối đường dẫn include. Khi có quá nhiều lệnh gọi lại thì chuyển sang aclocal và autoheader nội bộ.

- --warnings=category, -W category: Báo cáo những warning đến category (một danh sách được phân cách bằng dấu phẩy).

- category options:

+ cross: liên quan đến các vấn đề khi biên soạn chéo.

+ obsolete: báo cáo việc sử dụng các cấu trúc đã lỗi thời.

+ portability: các vấn đề về tính portable.

+ syntax: cấu trúc cú pháp đáng ngờ (mặc định, để tắt mặc định ta truyền vào option: -W none,obsolete).

+ all: báo cáo tất cả các warning

+ none: báo cáo none.

+ error: báo cáo warning như các lỗi

+ no-category: tắt

### Các option của lệnh make

- make all: Build các chương trình, thư viện, tài liệu v.v. (giống với lệnh make).

- make install: install những gì cần phải được install, copy các file từ cây của package đến thư mục hệ thống.

- make install-strip: giống với lệnh make install, nhưng sau đó strip cả các biểu tưọng debug. Một số người muốn trade space cho những thông báo lỗi có ích.

- make uninstall: ngược lại với lệnh make install, xoá các file đã install (lệnh này cần được chạy từ cùng một cây build đã install).

- make clean: xoá từ cây build các file đã được build bởi lệnh make all.

- make distclean: xoá cả những gì mà lệnh ./configure đã tạo ra.

- make check: chạy bộ test nếu có.

- make installcheck: kiểm tra các chương trình hoặc thư viện đã cài đặt nếu được hỗ trợ.

- make dist hoặc make distcheck: tạo lại package-version.tar.gz từ các file source.

### File configure.ac:

- được đọc bởi autoconf (để tạo configure) và automake (để tạo Makefile.in)

- những macros có tiền tố AC\_ là Autoconf macros, những macros bắt đầu bằng AM\_ là Automake macros.

- hai dòng đầu:

AC\_INIT([amhello], [1.0], [bug-automake@gnu.org])

AM\_INIT\_AUTOMAKE([-Wall -Werror foreign])

dùng để khởi chạy Autoconf và Automake. AC\_INIT lấy ra các thông số như tên gói, version, email liên hệ khi có bug trong gói (email này cũng xuất hiện ở cuối kết quả chạy lệnh ./configure --help).

- Các đối số của AM\_INIT\_AUTOMAKE là một danh sách các option:

- -Wall và -Werror yêu cầu automake bật tất cả các warning và báo cáo chúng giống như các error. (để cho an toàn và tránh bỏ sót issue).

- foreign option thông báo cho Automake rằng gói này không theo chuẩn GNU (gói GNU thường phải có thêm các file như ChangeLog, AUTHORS v.v.).

Ngoài ra có thể khai báo theo kiểu sau:

(AC\_INIT(src/main.cc)

PACKAGE=helloworld

VERSION=0.1.0

AM\_INIT\_AUTOMAKE($PACKAGE, $VERSION)

)

- Dòng AC\_PROG\_CC để tạo configure script tìm kiếm trình biên dịch C và định nghĩa các biến CC với tên của nó. Tệp src/Makefile.in (được tạo bởi Automake) sẽ sử dụng các biến CC này để build chương trình hello, nên khi configure tạo src/Makefile từ src/Makefile.in thì nó sẽ định nghĩa CC với giá trị đã tìm được. (nếu Automake yêu cầu tạo Makefile.in có dùng CC mà configure.ac chưa định nghĩa thì nó sẽ yêu cầu bạn gọi nó trong AC\_PROG\_CC).

(Nếu muốn sử dụng trình biên dịch C++ thì khai báo AC\_PROG\_CXX, các biến CXX, GXX và CXXFLAGS sẽ được set).

- Dòng AC\_CONFIG\_HEADERS([config.h]) được gọi để script configure tạo ra file config.h (tập hợp các '#define' mà các macros khác khai báo trong configure.ac)

- Ví dụ một đoạn config.h sau khi configure chạy như sau:

+ /\* Define to the address where bug reports for this package should be sent. \*/

#define PACKAGE\_BUGREPORT "bug-automake@gnu.org"

/\* Define to the full name and version of this package. \*/

#define PACKAGE\_STRING "amhello 1.0"

+ Như trong src/main.c có include config.h để sử dụng PACKAGE\_STRING. Trong những project thực tế, file config.h có thể rất lớn.

- AC\_CONFIG\_FILES macro khai báo một list các file mà configure sẽ tạo ra từ những template \*.in. Automake sẽ quét list này để tìm các file Makefile.am mà nó phải xử lý. (\*\*chú thích quan trọng: khi thêm mới một thư mục vào project của bạn, cần thêm cả Makefile của nó vào list này nếu không, Automake sẽ không bao giờ xử lý được file Makefile.am mới mà bạn tạo ra cho thư mục đó).

- AC\_OUTPUT: lệnh đóng.

#### Autoscan

- Có thể dùng lệnh autoscan để tạo file configure.ac (lệnh này sẽ kiểm tra các file source trong thư mục được đưa vào từ đối dòng lệnh hoặc thư mục hiện tại và tạo ra file configure.scan với các thành phần sơ bộ của một file configure.ac hoặc kiểm tra file configure.ac có sẵn và bổ sung nó).

- Khi sử dụng autoscan, ta phải tự sửa file configure.scan rồi đổi tên nó thành configure.ac. Autoscan có thể xuất ra các macro sai lệch, khi đó lệnh autoconf sẽ tạo ra warning và ta cần sửa lại chúng. Nếu ta muốn package dùng một file configuration header nào dó, cần gọi lệnh AC\_CONFIG\_HEADERS. Ta cũng có thể thay đổi hoặc thêm một số chỉ dẫn #if để làm cho chương trình hoạt động được với Autoconf.

- xem tệp autoscan.log để biết thêm lý do tại sao một macro được yêu cầu.

- autoscan sử dụng một số tệp dữ liệu (cài cùng Autoconf) để xác định macro nào cần xuất ra khi tìm thấy một ký hiệu đặc biệt nào đó trong file source của package.

- Một số tùy chọn của autoscan:

- --help, -h: in ra các option

- --version, -V: in ra version của autoconf

- --verbose, -v: in ra tên của các file mà nó kiểm tra và các ký hiệu tiềm năng mà nó tìm thấy.

- --debug, -d: không remove các file tạm.

- --include=dir, -I dir: chèn thư mục (dir) vào cuối đường dẫn include.

- --prepend-include=dir, -B dir: chèn thư mục (dir) vào đầu đường dẫn include.

### File Makefile.am

#### src/Makefile.am

###### bin\_PROGRAMS = hello

###### hello\_SOURCES = main.c

- File này chứa các hướng dẫn để Automake build và install helloworld.

- Một file Makefile.am có cấu trúc như một Makefile thông thường. Khi automake xử lý Makefile.am, nó sẽ sao chép toàn bộ các file vào Makefile.in (Makefile.in sẽ được chuyển thành Makefile bằng configure) và xử lý các biến bằng các tạo ra một số rule để build và thêm các biến khác.

- Những biến kết thúc bằng \_PROGRAMS là những biến đặc biệt dùng để liệt kê các chương trình mà Automake cần build. Theo cách gọi trong Automake thì hậu tố \_PROGRAMS được gọi là một primary, Automake cũng nhận các primary khác như \_SCRIPTS, \_DATA, \_LIBRARIES, v.v. tương ứng với các loại file khác nhau.

- phần ‘bin’ trong bin\_PROGRAMS cho automake biết rằng chương trình kết quả cần được cài vào bindir. GNU Build System sử dụng một tập hợp các biến để chỉ các thư mục đích cho phép người dùng tuỳ chọn:

|  |  |
| --- | --- |
| Biến thư mục | Giá trị mặc định |
| prefix | /usr/local |
| exec\_prefix | ${prefix} |
| bindir | ${exec\_prefix}/bin |
| libdir | ${exec\_prefix}/lib |
| … |  |
| includedir | ${prefix}/include |
| datarootdir | ${prefix}/share |
| datadir | ${datarootdir} |
| mandir | ${datarootdir}/man |
| infodir | ${datarootdir}/info |
| docdir | ${datarootdir}/doc/${PACKAGE} |

- Các chương trình cần được xây dựng từ các file source, do đó chương trình prog sẽ liệt kê trong một biến \_PROGRAMS, automake sẽ tìm biến có tên prog\_SOURCES là biến liệt kê các file source của nó. Có thể có nhiều hơn một file source, chúng sẽ được biên dịch và liên kết cùng với nhau.

- Automake cũng biết rằng các file source cần được phân loại khi tạo một tarball (không giống với xây dựng chương trình). Nên tác dụng khác của hello\_SOURCES là file main.c sẽ trở thành một phần của tarball được tạo ra khi chạy lệnh make dist.

#### The top-level Makefile.am

SUBDIRS = src

- Đây là một biến đặc biệt dùng để liệt kê tất cả các thư mục mà lệnh 'make' cần phải recurse trước khi xử lý thư mục hiện tại.

dist\_doc\_DATA = README

- yêu cầu file README phải được cài đặt trong thư mục doc. Các file được liệt kê với \_DATA không được tự động hiểu là một phần của tarball khi build bằng lệnh make dist nên ta cần thêm dist\_ để phân phối chúng. (nhưng file README có thể không cần thêm dist\_ vì automake tự động phân phối mọi file README. Xem thêm danh sách các tệp được tự động phân phối khác bằng lệnh automake --help).

## Cross compile - Biên dịch chéo

- Tạo ra một file nhị phân trên một nền tảng mà file này có khả năng chạy được trên một nền tảng khác.

--build=build: Package được build trên hệ thống này.

--host=host: Các chương trình và thư viện đã build ra sẽ được chạy trên hệ thống này.

- Khi '--host' được khai báo, 'configure' sẽ tìm kiếm trình biên dịch chéo phù hợp với nền tảng đó.

- Thường thì ta chỉ cần option --host và -build để biên dịch chéo. Chỉ có ngoại lệ là khi package được build chính là một trình biên dịch chéo, lúc này ta cần thêm option '--target' để chỉ kiến trúc mục tiêu (đích) của nó.

--target=target: Khi build một trình biên dịch: Hệ thống mà các công cụ sẽ tạo output.

--build: x86\_64-unknown-linux-gnu

- Chạy lệnh locate config.guess hiện ra một danh sách thư mục chứa file config.guess

- Sau đó vào thư mục /usr/share/automake-1.14/config.guess, chạy lệnh ./config.guess

- --build=type với type có thể là tên rút gọn của hệ thống, ví dụ như 'sun4' hoặc tên theo cấu trúc: 'cpu-company-system' với 'system' có thể là 'os', 'kernel-os'

--host: arm-linux-gnueabihf

- Ba machine:

- The build machine: nơi toolchain được build.

- The host machine: nơi toolchain sẽ được thực thi

- The target machine: nơi các mã nhị phân mà toolchain tạo ra được thực thi

- 4 kiểu build thông thường cho toolchain:

- Native build: build==host==target: được sử dụng để build gcc bình thường cho máy trạm. Ví dụ: build==host==target==x86.

- Cross-build: build==host!=target: được sử dụng để build toolchain mà sẽ hoạt động tại máy trạm của bạn nhưng tạo ra mã nhị phân cho target. Ví dụ: build==host==x86 target==arm

- Cross-native build: build!=host==target: được sử dụng cho toolchain sẽ hoạt động trên target và tạo ra mã nhị phân cho target. Ví dụ: build=x86 host==target==arm

- Canadian toolchain: build!=host!=target: được sử dụng khi build Kiến trúc A một toolchain chạy trên B nhưng tạo ra mã nhị phân cho Kiến trúc C. Ví dụ: build=x86 host=mac target=arm.