AUTOTOOLS Howto

이승윤
INZEN (http://www.inzen.com)

sylee (at) inzen.com

$Date $
Copyright © 2002 by

autoconf, automake 사용에 대한 소개

고친 과정
고침 1.012002년 5월 7일고친이 sylee
둘째버젼

$Revision : $

1. autotools requirement

현재 contents 버젼

- autoconf 2.53
- automake 1.6

예젼 autoconf 에서는 입력값으로 configure.in 을 사용하였으나 .in 은 configure 의 입력값으로 사용되고 있기때문에 오해를 피하기 위해 현재는 configure.ac를 사용하게 되었다.

주로 *.in 파일들은 configure의 입력값으로 취급되고 있으며, 이 파일들이 configure 과정을 거쳐서 실제 원하는 파일이 된다. 예를 들어 script.in 이라는 파일은 configre 를 거쳐서 script 라는 파일이 될수있다.

따라서 이들과 configure.in 을 구분하기 위해 현재는 configure.ac 라고 부르게 되었다.

2. 개요

1. configure.ac 에 Makefile 이 필요로 하는 사항들을 기록한다.

2. Makefile.am 에 이들 가변수를 사용하여 Makefile 의 초안을 작성한다.

3. configure 실행시에 configure.ac 에 지정한 항목들이 check 되면서 Makefile.am 에서 필요한 정보들이 유효한 값들로 치환된다.

4. Makefile.am 이 유효한 값들을 가지면서 Makefile 이 된다.

3. configure.ac

- autoconf, automake 의 입력값

  autoconf 는 이를 받아서 configure 파일을 만든다.

  automake 는 이를 makefile.am 과 함께 받아 makefile.in 을 만든다.

- 보통 이미 지정된 macro 를 사용하여 원하는 정보를 얻거나 설정할수있다.

  예

  AC_CANONICAL_SYSTEM : 현재 시스템에 대한 정보를 가져온다.

  AC_PROG_CC : cc 가 사용가능한지를 check

  AC_CHECK_HEADERS : 지정하는 header file 들이 시스템에 있는지 검사한다. HAVE_NAME_H 를 만들어준다.

  AC_TYPE_ : 지정한 type definition 을 확인한다.

  AC_SUBST : configure.ac 에서 사용한 변수들이 configure 시 *.in 파일에 지정한 변수를 찾아 유효값을 넣어준다.

  AC_CONFIG_FILES : 생성될 파일들을 지정한다. 흔히 Makefile.

4. Makefile.am

- automake 의 입력값
- 기본적인 Makefile 의 틀을 지니고 있지만, configure 에 따라 결정되는 변수들을 사용하고 있다.
- Makefile.am 은 해당 소스를 컴파일 하고자 하는 위치면 어디든 놓이게 된다.
- 가장 상위 directory 의 Makefile.am 에는 컴파일할 하위 directory 를 정하게 된다.

- configrure.ac 에서 정의되고 configure 를 통해 유효값들을 가지게 되는 변수를 가져오기 위해 @variable@ 을 사용한다. _PROGRAMS, _SCRIPTS, _SOURCES 등 지정된 primary 들이 있고 앞에 인스톨될 위치나 프로그램명, 프로그램의 소스등을 지정하게 되어있다.
- 예

```
hello.c 의 Makefile.am

PFLAG = @PFLAG@ : configure 를 통해 알아낸 값 가져옴

bin_PROGRAMS = world :  컴파일후의 프로그래명 지정
world_SOURCES = hello.c :  해당 프로그램의 소스지정

AM_CFLAGS = $(PFLAG) :  컴파일시의 flag 지정 macro
```

5. how it works

아래 순서로 진행된다.

- aclocal : configure.ac --> aclocal.m4
- autoheader : configure.ac + aclocal.m4 --> config.h.in
- autoconf : configure.ac + aclocal.m4 --> configure
- automake : configure.ac + aclocal.m4 + Makefile.am --> Makefile.in
- configure : Makefile.in --> Makefile

그림 **1. autotools input/output diagram**

6. 참고

autoconf,automake 를 실행하기 전에 이들을 보조하는 역할을 수행할 수 있는 것으로 aclocal, autoheader, autoscan이 있다.

6.1. aclocal

- configure.ac 에서 지정하는 macro 에 대한 보조역할을 한다. 기본적으로 사용할 수 있는 macro 에 대한 정보를 제공한다.

6.2. autoheader

- config.h.in 을 만들고 이는 후에 config.h 가 된다.
- configure.ac 에서 지정된 확인 사항들에 대한 정보를 config.h 에 남기게 된다.예를 들어 dirent.h 라는 header file 이 있는지 없는지를 AC_CHECK_HEADERS 를 검사했다면 이는 config.h 에 #difine HAVE_DIRENT_H 1 로 정의된다.
- config.h 는 configure 정보에 의존하는 함수나 header, 변수를 쓰는 곳에는 는 include 되어야 한다. 이를 통해 HAVE_NAME macro 를 써서 상황에 맞는 coding 을 해둠으로써 portable 한 code 를 만들수있게된다.

6.3. autoscan

- configure.ac 의 초안이라 할수있는 configur.scan 을 만들어 준다. configure.ac 가 configure 를 만들어서 필요한 정보를 check 한다고 할때 모든 사항을 거의다 check 해주는 configure.scan 은 우효할수있으나 실제로는 필요한 기능만을 가지는 configure.ac 를 직접작성하는 것이 좋다고 생각된다. configure.scan 은 너무 복잡하다.더군다나 100% 신뢰 가능하다고 생각되지 않고있다.

7. 간단한 예제 - minimal project

작은 예제를 통해서 autotool 을 사용하는 방법을 보이겠습니다.

흔히 가장 작은 예로 hello.c 를 autotool 을 이용하여 컴파일하고 install 하는 예를 많이 들어보이고 있는데 여기서는 이보다 조금 더 autoconf automake 의 능력을 발휘할 수 있는 예를 들어 보이겠습니다.

흔히 open source project 를 할때 현재 install 하고자 하는 machine 에 상관없이 configure, make, make install 을 통해서 간단하게 프로그램을 install 할수 있었는데 여기서는 이를 간단히 구현해보고자 합니다.더군다나 project 진행상 가장 기본적으로 필요한 기능입니다.

간단히 하기 위해 solaris 와 linux 에 대해서만 보이겠습니다.solaris 에서 실행될 파일과 linux 에서 실행될 파일이 필요합니다. 물론 같은 기능을 하는 파일로 같은 이름을 사용하여도 상관없습니다.

이들이 놓이게 될 장소와 이들을 가지고 원하는 작업을 수행하게 될 autotools 관련 파일들이 놓이게 될 위치를 아래에서 보이도록 하겠습니다.

7.1. 소스의 위치

소스들(solaris 에서 실행될 파일은 solaris.c 라 하고 linux 에서 실행될 파일을 linux.c 라고 하자.) 을 배치 할때는 운영체제 별로 directory 를 만들어서 관리하는 것이 좋다.(현재 가장 중요한 구분요소는 운영체제별로 build 가 이루어지게 하는 것이기 때문이기도 하다.) solaris.c 는 solaris directory 에 linux.c 는 linux directory 에 보관한다.

```
. project directory
./solaris/solaris.c
./linux/linux.c
```

project directory 에서 각각의 program source 들이 위치할수 있는곳은 바로 하나 아래의 dirctory 까지 이다. 즉 depth 1 의 directory 만 허용되며 그 이상의 깊이를 가지는 directory 안에 있는 소스에 대해서는 configure 로 Makefile 을 만들수 없어 실제로는 autoconf, automake 를 이용하기 위해서는 각 소스들은 depth 1 의 directory 에만 위치해야 한다. 이 문제는 automake 1.4 를 사용할 경우 고려되어야 하며 1.5 이후의 버젼에서는 이 문제가 해결되었다. 1.5 이상의 automake 를 사용하고 있다면 subdirectory 에 위치한 소스에 대해서도 makefile을 만들수 있고 subdirectory 의 파일을 상위 directory 에서 불러와 사용 할 수 있게 되었다.

7.2. autotools 입력 화일들의 위치

autoconf 의 입력화일인 configure.ac 는 project directory 에 위치하도록 한다. 즉 가장 상위 directry 에 위치하도록 하고 automake 의 입력파일이면서 configure 의 입력값이 될 Makefile.am 은 각각의 소스가 위치한 곳에 놓도록 한다. 이 Makefile.am 들이 각 소스에 대한 Makefile 이 된다고 생각할때 이 위치에 대한 이견은 없을것이다.

```
. project directory
./configure.ac
./Makefile.am

./solaris/solaris.c
./solaris/Makefile.am

./linux/linux.c
./linux/Makefile.am
```

이제 각 directory 에 위치한 configure.ac, Makefile.am 이 어떻게 작동하는 지 살펴보자.

7.3. 최상위 directory 의 Makefile.am 의 역할

최상위 directory의 Makefile.am 은 configure 시 build 할 directory 를 판단한다. SUBDIRS 라는 변수를 가지고 있으며 여기에 지정된 directory 에 대해 Makefile.am 을 가지고 Makefile 을 만들게 된다. 우리는 configure 결과에 따라 시스템이 solaris 인지 linux 인지를 판단하여, solaris directory 를 build 할 것인 지 linux directory 를 build 할 것인지를 정해야 한다.즉 이 SUBDIRS 는 configure 시에야 알 수 있는 값이므로 일단 적당한 변수로 두고 configure 시 유효한 값으로 치환되도록 해야한다.

configure 시 build directory 를 결정하도록 하며 이 directory 의 값을 DIRS 라는 변수에 저장하도록 하고, 최상의 Makefile.am 에서는 이 값을 가져다 쓰기 위해 다음과 같이 쓰면 된다.

```
SUBDIRS=@DIRS@
```

이제 configure 에서 이 DIRS 값을 어떻게 결정하고 Makefile.am 에 넘겨주는지 보도록 한다.

7.4. configure.ac 의 역할

configure시에 많은 정보를 얻어야 하며 이를 바탕으로 Makefile이 만들어 진다고 할때 configure 시에 check 해야 할것들을 미리 정해주는 configure.ac 의 역할이 autotools 사용의 효용성을 결정한다고 할수도 있다. 각각의 check 사항들은 macro 로 정의되어 있으며 그에 대해서는 필요할경우마다 하나씩 찾아가면서 공부할수있다. 즉 하나의 macro 를 알고 모르는것이 그렇게 중요하지는 않으며 원하는 check 사항이 생길때마다 macro 를 조사하는 것으로도 가능하다.

이번 예에서는 각 platform 별로 build 하기를 원하기 때문에 configure 시 현재의 platform 을 알아야 하고 이를 바탕으로 build 할 subdirectory, 즉 최상위 Makefile.am 에서 SUBDIRS 에 들어가게될 DIRS 의 값을 결정해 주어야 한다. 기본적인 여러 macro 들이 있겠지만 일단 이과정이 어떻게 처리되는지를 보자.

```
AC_CANONICAL_SYSTEM

case "$target" in
i?86-*-linux*)
DIRS="linux" ;
AC_SUBST(DIRS)
;;
*solaris*)
DIRS="solaris" ;
AC_SUBST(DIRS)
;;
*)
echo unsupported system : $target
;;
esac
```

AC_CANONICAL_SYSTEM 이라는 macro 를 통해서 target 이라는 shell 변수에 현재 system 의 정보를 기록하게 된다. 이 정보는 configuration name 형식으로 cpu-manufacturer-operation_system 으로 구성되며, 위의 case 문은 이를 바탕으로 쓰여졌다. 즉

AC_CANONICAL_SYSTEM 이라는 macro 의 실행결과 생긴 target 변수의 내용을 바탕으로 target 에 따른 행동을 할 수 있게 된다.

이제 case 문 안으로 들어가면 DIRS 라는 변수에 linux 또는 solaris 를 assign 하고 이를 AC_SUBST() macro 를 통해 대치시키고 있다. 이 macro 는 parameter 로받은 변수에 대해 Makefile.am 이나 *.in 에서 이 변수명을 쓰는 값들을 현재 assign 한 값으로 대치시켜 준다. 이번 예제의 경우는 최상위 directory의 Makefile.am 에서 SUBDIRS=@DIRS@ 의 DIRS 의 값을 바꾸게 된다.

따라서 configure 를 거치게 되면 최상위 Makefile.am 의 SUBDIRS 에는 해당 머신에따라 linux 혹은 solaris 가 들어가게 된다. 편의상 현재 configure 가 일어나고 있는 머신을 linux 라고 할때 SUBDIRS=linux 가 되어서 make 를 하게 될 경우 이 Makefile (최상위 directory 의 Makefile.am 에서 만들어지는 Makefile) 에 의해 linux directory 만 빌드가 이루어 지게 된다. solaris directory 에 대한 빌드는 이루어지지 않으며 따라서 platform 을 따져서 원하는 source 를 빌드할수 있게 된다.

7.5. 하부 directory 의 Makefile.am 의 역할

기본적인 사용을 위해서는 다음 두줄로도 충분합니다.

```
./linux/Makefile.am

bin_PROGRAMS = linux
linux_SOURCES = linux.c
```

기본적으로 Makefile.am 에도 쓸수 있는 값들이 정해져 있다. _PROGRAMS, _SCRIPTS, _DATA, _SOURCES primry 들을 쓸 수 있는데 undersquare 앞에 특정 directory 를 정해 주어야 하는데 이 directory 가 해당 프로그램이나 파일들이 install 될 위치를 가르킨다. 위의 Makefile.am 에서는 linux.c 파일을 compile 해서 linux 라는 이름의 프로그램으로 만든후 이를 bin directory 에 install 하게 된다.

여기서 bin directory는 위의 configure.ac 에서 정의된 default prefix 아래의 bin directory 를 가르키게 된다. 즉, configure.ac 에서

```
AC_PREFIX_DEFAULT(/usr/eostk)
```

라고 정의하였다면, 이 bin 은 /usr/eosrk/bin 이 되고 linux 프로그램은 그 아래에 install 된다.

_SCRIPTS 는 script 를 install 하기 위해 쓰이고 _DATA 는 그냥 파일들을 install 하게 될 경우에 쓰인다. 둘다 _SOURCES 를 필요로 하지 않기 때문에 비슷하나 _SCRIPTS 는 실행 가능하고 _DATA 는 실행불가능하기때문에 따로 쓰고 있다.

이렇게 쓰여진 Makefile.am 은 configure 시 최상위 Makefile.am 에서 지정하는 SUBDIRS 에 포함될 경우만 build 가 이루어지는데 최상위 Makefile.am 처럼 지정된 변수가 있고 이를 configure.ac 에서 치환해 준다면 해당 유효값이 들어가서 Makefile 이 만들어 지게 된다.(실제로는 Makefile.am 은 automake 를 통해 Makefile.in 로 만들어지고 configure 시 이를 입력값으로 취해서 Makefile 이 만들어 진다.)

7.6. 실행순서

위의 과정을 실행시키는 방법은 아래와 같다.

```
aclocal
autoheader
autoconf
automake --add-missing -copy
./configure
make
make check
make install
```

흔히 configure 전까지의 과정을 묶어서 bootstrap 또는 autogen.sh 라는 script 로 만들어서 한꺼번에 실행시킨다. end user 는 단지 ./configre, make, make install 만을 하면 특정 소스를 build 할 수 있는데 autogen.sh 를 end user 가 할수있어야 한다 없어야 한다는데에는 여러 의견이 있다.

7.7. 요약과 전체 흐름 정리

지금까지 예제를 보면 최상위 Makefile.am 에 configure 시에 build 될 subdirectory 를 정하도록 하였으며 이 값을 알아내기 위해 configure.ac 에서 현 시스템을 검사하고 이 값을 넘겨주도록 되어있다. 즉 configure 실행후 make 를 하게 되면 최상위 Makefile 에 의해서 지정된 directory 만 build 가 이루어져 platform 에 따라 알맞는 소스가 build 되고 install 될 수 있다.

지금은 directory 단위로 특정 directory 를 build 할지 안할지를 결정하였지만 autotools 을 좀더 미시적은 범위에서 이용한다면 하나의 코드에 대해서도 define 문을 이용해서 선별적으로 코드가 build 되게 할 수 있다.

8. 기본 template - configure.ac, Makefile.am

configure.ac, Makefile.am 을 작성할때 기본적으로 적어야하는 macro 등을 설명하겠다. 위의 전체 흐름을 이해했으면 지금부터의 사항들은 한번 읽고 지나쳐도 된다. 그러나 기본인 만큼 이대로 쓰지 않을경우 전혀 작동하지 않는다. --;

8.1. configure.ac

configure.ac 에서는 check 사항들을 test 하는 일반적인 순서가 정해져있다. 이 순서를 따르면서 필요한 사항들만 검사하면 된다. 대략적인 순서는

- Boilerplate : 기본적인 macro. AC_INIT 이 있는데 이는 꼭 맨 앞에 쓰여져 있어야 한다. AM_INIT_AUTOMAKE, AC_CONFIG_HEADER, AC_REVISION 이 이에 속한다. 흔히 AC_INIT 과 AM_INIT_AUTOMAKE, AM_CONFIG_HEADER 는 꼭 들어간다.

- Options
- Programs : configure 과정이나 build 과정상에 필요한 프로그램들을 check 한다. AC_CHECK_PROG, AC_PATH_TOOL 이 있다.
- Libraries
- Headers : 필요한 header file 들이 존재하는지를 검사한다.
- Typedefs and structures
- Functions
- Output

위의 항목들은 autoscan 을 통해서 configure.scan 이라는 파일을 생성하게 되면 그대로 나오게 된다. 이 autoscan 은 configure.ac 파일의 작성을 쉽게 해주고자 하는 도구이다. 허나 아주 자세한 것까지 출력을 해주고 필요없는 것들이 많기때문에 configure.ac 는 직접 작성하는것이 좋다고 생각된다. 간단히 예를 들어보면

```
# Process this file with autoconf to produce a configure script.
AC_INIT(eostk, 1.0, sylee@inzen.com)
AM_INIT_AUTOMAKE(eostk, 1.0)
AM_CONFIG_HEADER(config.h:config.h.in)

# Checks for programs.
AC_PROG_CC
AC_PROG_INSTALL

# Checks for libraries.
AC_CHECK_LIB(socket, connect)

# Checks for header files.
AC_HEADER_DIRENT

# Checks for typedefs, structures, and compiler characteristics.
AC_TYPE_UID_T
AC_TYPE_PID_T

# Checks for library functions.
AC_HEADER_STDC
AC_CHECK_FUNCS(strcpy bcopy)

# Finally, make output files
AC_CONFIG_FILES(
        Makefile
        solaris/Makefile
        linux/Makefile
)
AC_OUTPUT
```

순이 될수있다. 이 순서대로 필요한 check 사항을 추가시킬수도 있고 필요없는것은 제외시킬수도 있다.configure 시에 모든 필요한 정보들이 수집된다고 할때 이의 초안이 되는 configure.ac 파일이 가장 중요하다고 할수있다.

8.2. Makefile.am

Makefile.am 의 기초는 위에서 설명한 것으로 충분하다고 생각한다. 추가적으로 알고있으면 좋은 사항들을 적어보면 check_ prefix 와 TESTS primary, install-exec-hook 이 있다.

- check_ prefix, TESTS primary : 이들은 make check 으로 실행이 되는 파일들을 지정할수있다. 흔히 testsuit 이 이에 해당하는데 여기에 지정된 프로그램들은 실제로는 install 되지 않고 build time 에만 실행이 가능하다. 즉 make check 을 하면 TESTS 에 정의된 프로그램들을 찾아서 build 시켜서 실행시키게 된다. 이때 install 은 일어나지 않는다.
- install-exec-hook target : make install 을 하면 흔히 지정된 동작이 일어나게 된다. build 가 끝난후 이를 지정한 directory 에 복사하거나 하는데 이를 마친후 추가적인 작업을 하고자 할 경우 install-exec-hook 에 그 행동을 지정해 주면 된다. 이를 install hook 이라고 한다.

```
check_PROGRAMS = test1
test1_SOURCES = test1.c

check_SCRIPTS = test2

TESTS = test1 \
test2


install-exec-hook :
chmod +x INSTALL.solaris
./INSTALL.solaris
```

이렇게 하면 test1 이라는 c 프로그램과 test2 라는 script 가 make check 시 실행될 TESTS 파일로 지정이 되고 make check 이 불려지기 전까지는 build 되지 않는다. make check 이 불러지면 test1 이 빌드되고 test1 test2 순으로 실행된다. 그러나 install 은 되지 않는다.

install-exec-hook 이라는 target 은 make install 을 정상적으로 실행한후 실행된다. 현재 directory 가 solaris 라고 할때 solaris 의 make install 과정을 마친후에 실행이 된다. 이 후에 solaris2 라는 directory 를 빌드하게 되어있다 하더라도 이 과정은 해당 directory 의 install 을 마친후에 일어난다. 즉 solaris 의 make install 을 마친후에 이 과정이 실행된다. 그 후에 solaris2 를 build 하게 된다.

9. 미시적 autotools 의 이용

이 문서는 multi platform 에서 해당 platform 에 맞는 build 를 하기 위한 howto 만을 제공하고 있다. 하지만 앞에서 잠깐 이야기했듯이 autotools 을 미시적인 범위까지 사용하면서, config.h 를 효과적으로 사용한다면 하나의 code 를 얼마든지 다르게 build 할 수 있다.

예를 들어 앞장의 configure.ac 를 볼때,

```
AC_CHECK_FUNCS(strcpy bcopy)
```

라는 부분이 있다. 이 macro 를 실행하고 나면 config.h 에 현재 configure 가 일어난 system에 strcpy, bcopy 라는 함수가 있는지 없는지에 대한 기록이 남게 된다.따라서 개발자가 미리 혹시나 strcpy 나 bcopy 를 가지고 있지 않는 system에 대한 처리를 해줄수가 있다.

config.h 의 내용

```
#define HAVE_BCOPY 1
```

strcpy 를 사용하는 code 시작 부분 혹은 그 code 가 include 하는 header file

```
#if !HAVE_STRCPY
# if HAVE_BCOPY
# define strcpy(dest,src) bcopy(src, dest, 1+strlen(src))
# else
error no strcpy or bcopy
# endif
#endif
```

위처럼 하면 strcpy 를 가지고 있지 않은 machine 에 대해 이를 bcopy 를 이용해 구현할 수 있게 해줄 수 있다. 즉 작은 범위에까지 잘 이용한다면 완벽히 system independent 한 code 를 만들 수 있게된다.

10. 참고자료

GNU AUTOCONF, AUTOMAKE, and LIBTOOL(-Gary V.Vaughan, Ben Elliston, Tom Tromey, and Ian Lance Taylor- New Riders) 이라는 책을 참고하였다. 이와 함께 http://www.gnu.org/manual 에 있는 autoconf, automake 를 간략히 보았다.

전체적으로 돌아가는 개념을 익히기 위해서는 위의 책이 좋고 기타 자세한 option 이나 macro 에 대해서는 gnu manual 을 찾아보면서 작업하는게 좋다고 생각 된다.. <--- 개인적인 생각입니다.

류창우씨께서 그리신 개념도를 첨부합니다. 역시 도움이 되리라 생각합니다.

그림 **2. autotools input/output by 1999 Changwoo Ryu**