Autotools Tutorial

Mengke HU

ECE Department
Drexel University

ASPITRG Group Meeting

Outline

The Basics of Autotools

1. The purpose of autotools
   - It serves the needs of your users (checking platform and libraries; compiling and installing).
   - It makes your project incredibly portable—for different system platforms.
2. Why should we use autotools:
   - A lot of free softwares target Linux operating system.
   - Autotools allow your project to build successfully on future versions or distributions with virtually no changes to the build scripts.

The Basics of Autotools

1. The purpose of autotools
   - It serves the needs of your users (checking platform and libraries; compiling and installing).
   - It makes your project incredibly portable—for different system platforms.
2. Why should we use autotools:
   - A lot of free softwares target Linux operating system.
   - Autotools allow your project to build successfully on future versions or distributions with virtually no changes to the build scripts.

The Basics of Autotools

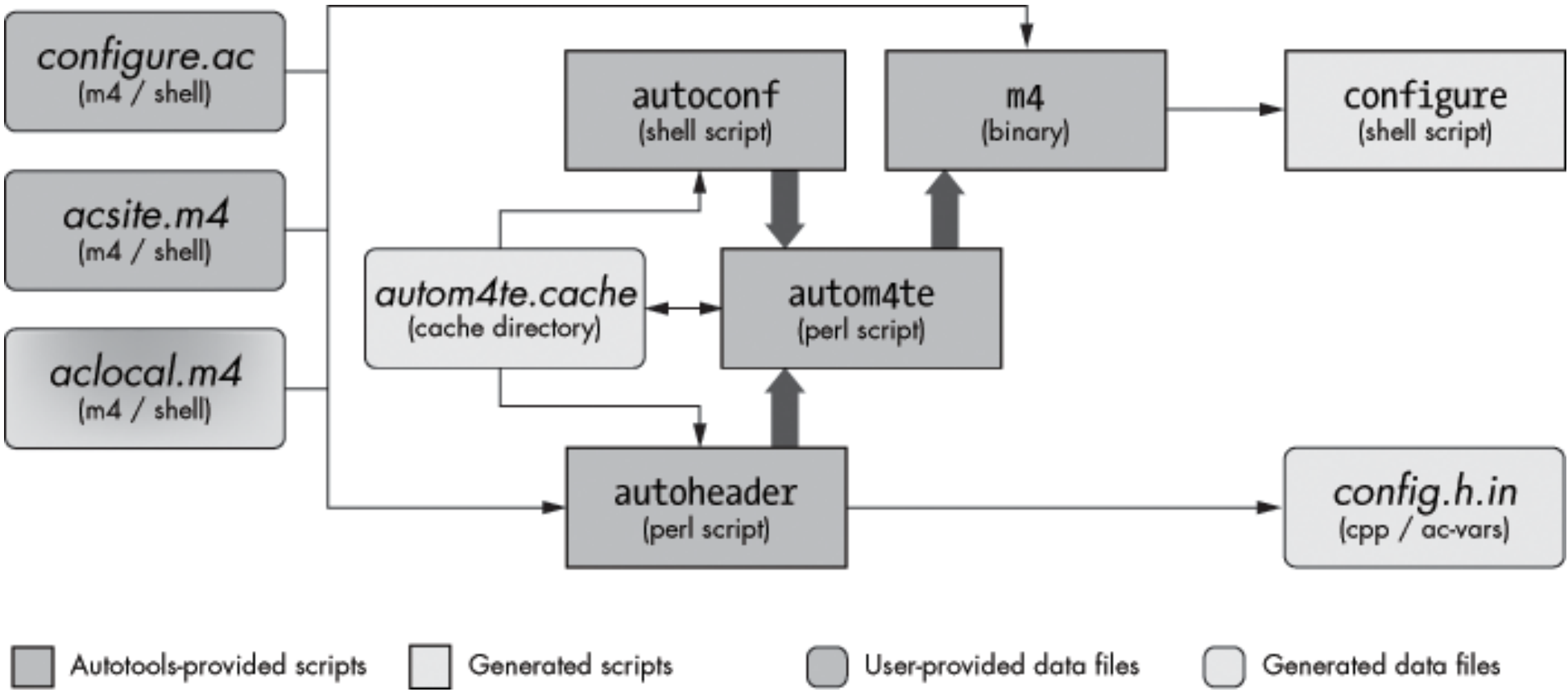
1. 3 GNU packages for GNU build system
   - Autoconf
     Generate a configuration script for a project
   - Automake
     Simplify the process of creating consistent and functional makefiles
   - Libtool
     Provides an abstraction for the portable creation of shared libraries
2. Basic steps (commends) to build and install software
   - tar -zxvf package_name-version.tar.gz
   - cd package_name-version
   - ./configure
   - make
   - sudo make install

The Basics of Autotools

1. 3 GNU packages for GNU build system
   - Autoconf
     Generate a configuration script for a project
   - Automake
     Simplify the process of creating consistent and functional makefiles
   - Libtool
     Provides an abstraction for the portable creation of shared libraries
2. Basic steps (commends) to build and install software
   - tar -zxvf package_name-version.tar.gz
   - cd package_name-version
   - ./configure
   - make
   - sudo make install

Autoconf

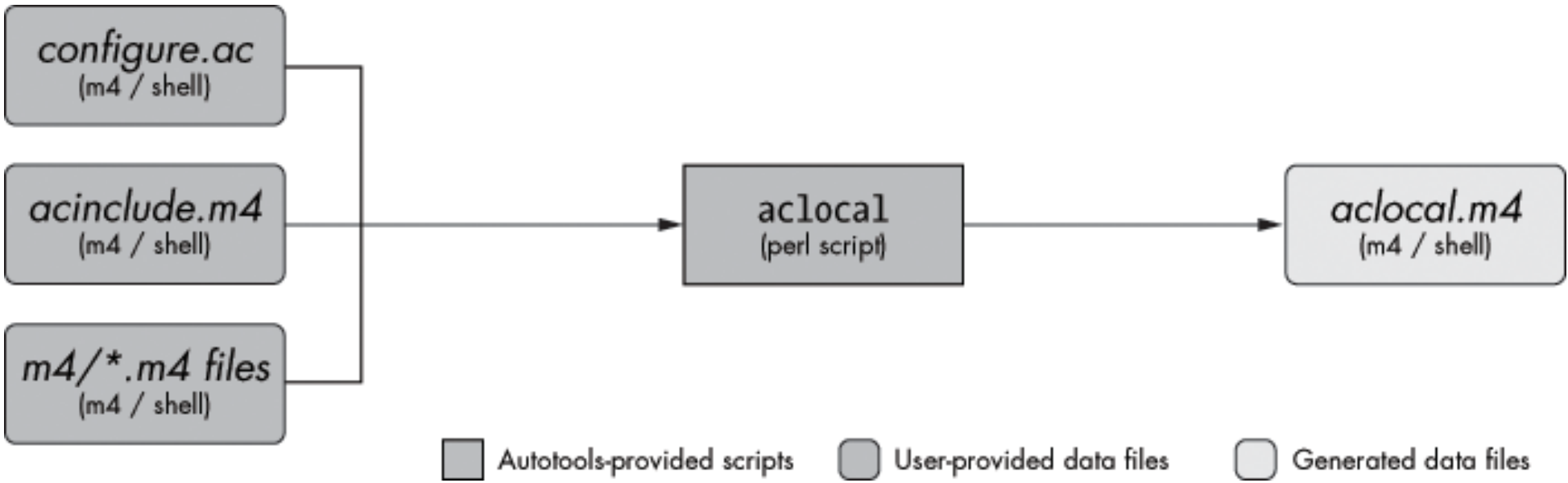
Data Flow Diagram for Autoconf



Automake

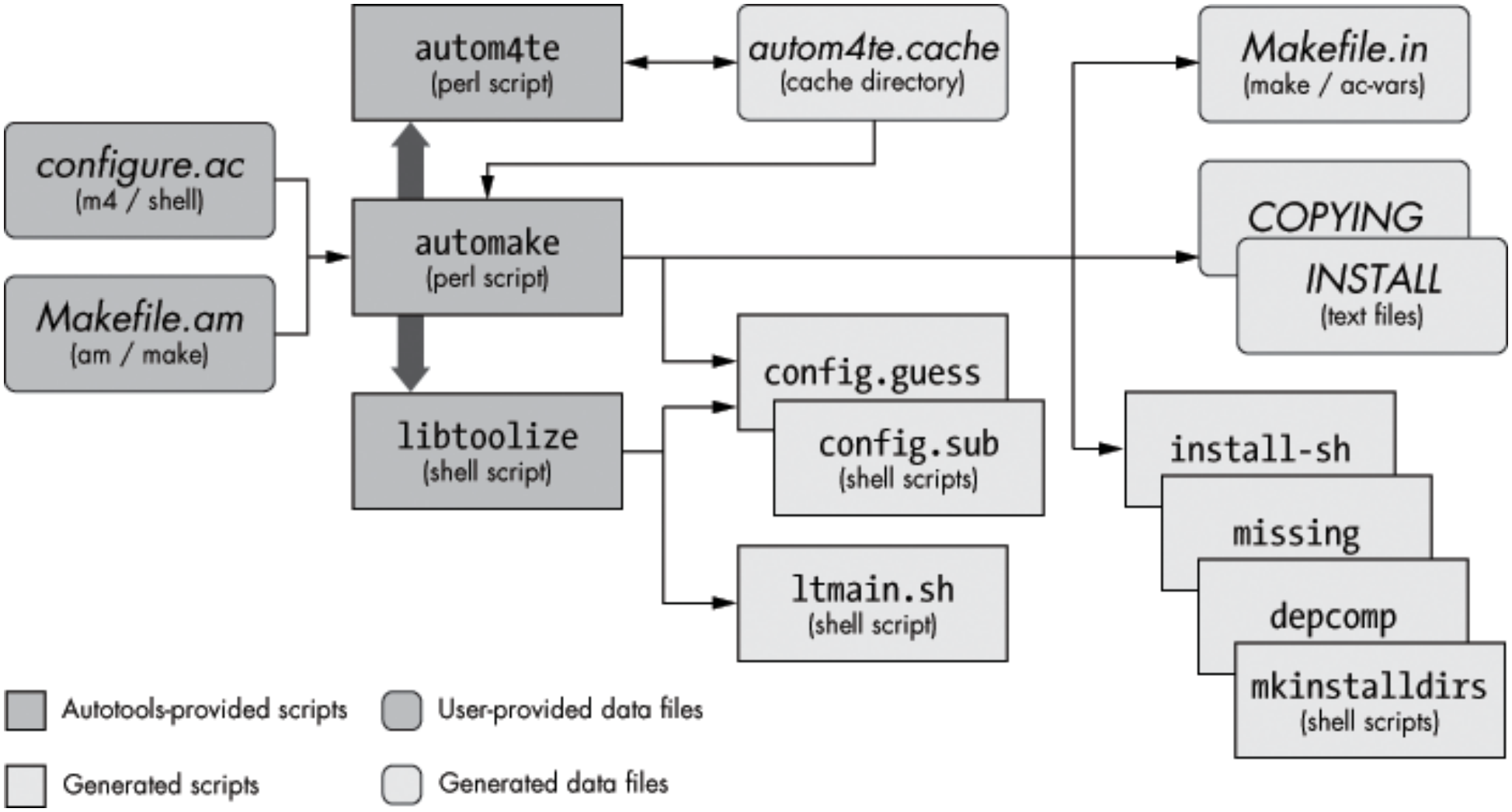
1. automake
   - automake
   - aclocal

Data Flow Diagram for aclocal



Libtools

Data Flow Diagram for automake and libtool



A simple project structure

- jupiter1.0
- Makefile
  - ./src/
    - Makefile
    - main.c

Makefile

- Makefile layout

  var1=val1
  var2=val2
  ...
  target1 : t1_dep1 ... t1_depN
  <TAB> shell-command1a
  <TAB> shell-command1b
  ...
  target2 : t2_dep1 ... t2_depN
  <TAB> shell-command2a
  <TAB> shell-command2b
  ...

Makefile Example

```
tarname         = jupiter
distdir         = $(tarname)-$(version)
# VPATH-related substitution variables
srcdir          = .
# Prefix-specific substitution variables
prefix          = /usr/local
exec_prefix     = ${prefix}
bindir          = ${exec_prefix}/bin
#CFLAGS         = -g -O0

all: jupiter
jupiter: main.c
        gcc -g -O2 -o jupiter ../../src/main.c
        #$(CC) $(CPPFLAGS) $(CFLAGS) -o $@ main.c
check: all
        ./jupiter | grep "Hello from .*jupiter!"
        @echo "*** ALL TESTS PASSED ***"
install:
        install -d $(DESTDIR)$(bindir) #DESTDIR is to dsupport the staged installation
        install -m 0755 jupiter $(DESTDIR)$(bindir)
        cp jupiter /usr/bin
        chown root:toot /usr/bin/jupiter #change file owner and group
        chmod +x /usr/bin/jupiter
uninstall:
        -rm $(DESTDIR)$(bindir)/jupiter
Makefile: Makefile.in ../config.status
        cd .. && ./config.status src/$@
../config.status: ../configure
        cd .. && ./config.status --recheck
clean:
        -rm jupiter
.PHONY: all clean check install uninstall
```

Autoconf Syntax

1. Autoconf
   - Autoconf use M4 macro language.
   - M4 is for the purpose of macro language processor.
2. Syntax for configure.ac file

   AC_INIT([PackageName], [version])
   AC_CONFIG_FILES([Makefile src/Makefile])
   AC_OUTPUT

Check Datatype

1. Check for types for different platforms
   - AC_TYPE_UINTN_T
   - AC_TYPE_INTN_T

   Where $N = 8, 16, 32, 64$
2. How those macros work?
   - Check if stdint.h or inttypes.h have UINTN_T or UINTN_T
   - If not, define UINTN_T or UINTN_T.

Add automake functionality

1. Enabling Automake in configure.ac
   - *AC_INIT([Jupiter], [1.0], [jupiter-bugs@example.org])*
   - *AM_INIT_AUTOMAKE*
   - *AC_CONFIG_SRCDIR([src/main.c])*
2. Makefile.am for compiling sources
   - *bin_PROGRAMS = jupiter*
   - *jupiter_SOURCES = main.c*
   - *jupiter_CPPFLAGS = -I$(top_srcdir)/common*
   - *jupiter_LDADD = ../common/libjupcommon.a*

Add convenience libraries (static library)

- Makefile.am for convenience libraries
  - noinst_LIBRARIES = libjupcommon.a
  - libjupcommon_a_SOURCES = jupcommon.h print.c

Shared Libraries

- Shared libraries contain only a small table of functions that programs require.
- Shared libraries make executable file smaller.
- It is easy to update shared libraries without recompiling projects.
- Shared libraries use dynamic linking, which provides chunks of executable code that the operating system can load into a program's address space and execute.

Libtools

1. How the libtool works?
   - Libtool script is generated by configure script and ltmain.sh.
   - Libtool script then insulates the author of the build system from the nuances of building shared libraries on different platforms.
2. Add following lines in configure.ac:
   - *AC_SEARCH_LIBS([dlopen], [dl])*
   - *AC_SEARCH_LIBS([lt_dlopen], [ltdl])*

Demo1: Makefile for compiling?

- Project_Name/src/main.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char * argv[])
{
    printf("Hello from %s!\n", argv[0]);
    return 0;
}
```

Demo1: Makefile for compiling?

- Project_Name/src/Makefile

```
#TARGET1: refer to actions "HelloWorld" in TARGET2
all: HelloWorld
#TARGET2: depends on HelloWorld.o,
#         ultimately depends on TARTET3
HelloWorld: HelloWorld.o
        gcc -o HelloWorld HelloWorld.o
#TARGET3: depends on main.c
HelloWorld.o: main.c
        cc -c -o HelloWorld.o main.c
#TARGET4: delete HelloWorld and HelloWorld.o
clean:
        rm HelloWorld HelloWorld.o
#TARGET5: refer to actions "all" and "clean"
.PHONY: all clean
```

Demo2:Makefile for Creating Source Distribution Archive

- Project_Name/Makefile

```
#TARGET1: refer to src/Makefile
#         1. go into src folder
#         2. run target "all" in src/Makefile
all:
        cd src && make
#TARGET2: creates distribution, refers to TARGET3
dist: HelloWorld-1.0.tar.gz
#TARGET3: generate software package
#         ultimately depends on TARGET4
HelloWorld-1.0.tar.gz: HelloWorld-1.0
        tar -zcvf HelloWorld-1.0.tar.gz HelloWorld-1.0
        rm -rf HelloWorld-1.0
#TARGET4: create package distribution directory
HelloWorld-1.0:
        mkdir -p HelloWorld-1.0/src
        cp Makefile HelloWorld-1.0
        cp src/Makefile HelloWorld-1.0/src
        cp src/main.c HelloWorld-1.0/src
#TARGET5: check the all the functions for both Makefile and src/Makefile
distcheck: HelloWorld-1.0.tar.gz
        tar -zxvf HelloWorld-1.0.tar.gz
        cd HelloWorld-1.0 && make all
        cd HelloWorld-1.0 && make clean
        rm -rf Helloworld-1.0
        @echo "**** Package Helloworld-1.0.tar.gz is ready for distribution."
#TARGET6: refer to actions "all", "clean" and "dist"
.PHONY: all clean dist
```

Demo3: Autoconf for configuring project

- Project_Name/configure.ac

```
#pass 'HelloWorld' to variable "package" in Makefile.in
#pass '1.0' to variable "version" in Makefile.in
AC_INIT([HelloWorld],[1.0])
#generate Makefile by running the script './configure'
AC_CONFIG_FILES([Makefile src/Makefile])

AC_OUTPUT
```

Demo3: Autoconf for configuring project

```
# Configure Input
package          = @PACKAGE_NAME@
version          = @PACKAGE_VERSION@
tarname          = @PACKAGE_TARNAME@
distdir          = $(tarname)-$(version)

#TARGET1: refer to src/Makefile
#        1. go into src folder
#        2. run target "all" in src/Makefile
all clean check HelloWorld install uninstall:
        cd src && $(MAKE) $@
#TARGET2: creates distribution, refers to TARGET3
dist: $(distdir).tar.gz
#TARGET3: generate software package
#         ultimately depends on TARGET4
$(distdir).tar.gz: $(distdir)
        tar -zcvf $(distdir).tar.gz $(distdir)
        rm -rf $(distdir)
#TARGET4: create package distribution directory
$(distdir):
        mkdir -p $(distdir)/src
#       cp Makefile $(distdir)
#       cp src/Makefile $(distdir)/src
```

Demo3: Autoconf for configuring project

- autoreconf
  Generate configure script from configure.ac
- ./configure
  Generate Makefile from Makefile.in
- make dist
  Package the software project into tarball

Demo4: Autoconf for automatic Makefiles

- configure.ac

```
#Initializes the Autoconf system
#pass 'HelloWorld' to variable "package" in Makefile.in
#pass '1.0' to variable "version" in Makefile.in
AC_INIT([HelloWorld],[1.0])

#enable automake
AM_INIT_AUTOMAKE

#generate Makefile by running the script './configure'
AC_CONFIG_FILES([Makefile src/Makefile])

#check compilers
AC_PROG_CC([cc gcc])

#check data type UINT16_T
AC_TYPE_UINT16_T

#check
AC_PROG_INSTALL

AC_OUTPUT
```

Demo4: Autoconf for automatic Makefiles

- Project_name/Makefile.am

```
#specify the source file folder
SUBDIRS = src
```

- Project_name/src/Makefile.am

```
#Specify the executable file name
bin_PROGRAMS = HelloWorld
#specify the source file
HelloWorld_SOURCES = main.c
```

Demo4: Autoconf for automatic Makefiles

- touch NEWS README AUTHORS ChangeLog
  Generate empty files “NEWS”, “README”, “AUTHORS”, and “ChangeLog”
- autoreconf -i
  Generate configure script and Makefile.in from configure.ac and makefile.am
- ./configure
  Generate Makefile from Makefile.in
- make
  Compile program
- make distcheck
  Check distribution

Summary

- Design configure.ac and makefile.am according to your project.
- autoreconf -i
  - aclocal.m4
  - autom4te.cache
  - config.h.in
  - configure
  - install-sh
  - Makefile.in
- Add macros according the messages from previous step; Search libltdl library if libtool is used
- ./configure
  - It is to generate Makefile
- Package your project


Calcote, John
A practitioner's guide to GNU Autoconf, Automake, and Libtool.


GNU Autotools online