Using Automake and Autoconf with C ++

Contents

Introduction

> *The **automake** and **autoconf** tools can be used to manage C++ projects under Unix. They should save a lot of time compared to writing **Makefile**s and **configure** scripts manually, while ensuring that your project is structured according to* GNU standards [1].
>
> *However, it is difficult for beginners to get started. Hopefully, this tutorial will provide enough information for C++ programmers who are new to Unix to create their first C++ projects, while gaining a superficial understanding of what the tools are doing.*
>
> *If you find any problem with this page, we welcome feedback or patches against the example code, which can be downloaded from the* Gitorious project page [2]. *Either send a merge request or email* murrayc@openismus.com [3].

make and configure

> *The **make** tool can be used to manage multi-file projects. **make** uses the **Makefile** file in your project folder, which lists the various compiling and linking steps, targets, and dependencies. **make** is well explained in* C-Scene: Multi-File Projects and the GNU Make Utility [4].
>
> *A **configure** script can be used to aid cross-platform compiling. A suitable **configure** script should interpret a **Makefile.in** file and then create a platform-specific **Makefile** file. It will do this after performing several tests to determine the characteristics of the platform.*
>
> *This allows a user to type `./configure` and then `make` to compile a project on her system.*

automake and autoconf

> *Obviously most well-written Makefiles and configure scripts will look very similar. In fact, GNU provide guidelines about what should be in these files. Therefore, GNU created **automake** and **autoconf** to simplify the process and*

ensure that the Makefile and configure script conform to GNU standards.

Here is a brief explanation of how these tools are used. You can see examples of the files used by these tools in the Example Files *section.*

Note: These tools use the M4 programming language. ***aclocal*** *adds* ***aclocal.m4*** *to your project directory, which defines all M4 macros which are used by the configure script.*

autoconf

autoconf *looks for a file called* ***configure.ac*** *(or* ***configure.in*** *for backward compatibility). It then runs the M4 macro processor to create the* ***configure*** *script.*

Whenever you add a macro call to ***configure.ac****,* ***aclocal*** *should be executed as well as* ***autoconf****, because* ***aclocal*** *scans* ***configure.ac*** *to find which macros it should provide. However, nowadays one can simply run the* ***autoreconf*** *utility, which takes care of those details automatically.*

Lines which every configure.ac should have

Every configure.ac should have lines like the following:

```
AC_INIT([Hello], [0.1], [bug-report@hello.example.com], [hello], [http://hello.example.com/])
AC_PREREQ([2.59])
AM_INIT_AUTOMAKE([1.10 no-define])
AC_CONFIG_HEADERS([config.h])
AC_PROG_CXX
AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

The AC_INIT macro [5] initializes autoconf with information about your project, including the project name, version number, bug-reporting address, tarball name and the project homepage.

The AM_INIT_AUTOMAKE line adds several standard checks and initializes automake.

AC_PROG_CXX checks for a C++ compiler. If your project uses C, you can check for a C compiler with AC_PROG_CC.

AC_CONFIG_FILES [6] lists the files to be generated by `configure`*. By default, each file is generated from a template file of the same name but with an* `.in` *extension appended.*

> > AC_OUTPUT [7] *finishes configure processing, and generates the output files.*
>
> ***Using a Config Header***
>
> > *The* AC_CONFIG_HEADERS([config.h]) *line indicates that you will be using a config.h file.* ***autoconf*** *will then need a* ***config.h.in*** *file, which it processes to create the* ***config.h*** *file. The generated header is included by your source code to pick up the results of the various configuration tests in the form of C preprocessor macro definitions.*
> >
> > *The template file* ***config.h.in*** *can also be generated automatically by the* ***autoheader*** *tool.* ***autoreconf*** *will invoke* ***autoheader*** *when* `AC_CONFIG_HEADERS` *is used in* ***configure.ac****. If multiple configuration headers are used,* ***autoheader*** *will always only generate the template for the first one in the list.*

automake

> ***automake*** *looks for a file called* ***Makefile.am****. It then creates a* ***Makefile.in****, based on the macros which it finds. This is later used by the* ***configure*** *script (see above).*
>
> ***GNU-style projects, or not***
>
> > *Because* ***automake*** *tries to make a GNU-style project by default, it will add a COPYING file and complain if some other necessary informative text files are missing. For the moment, you can create blank files with the following command, and fill them in later:*
> >
> > ```
> > touch NEWS README AUTHORS ChangeLog
> > ```
> > *If you do not want these GNU-style files, then you could add the 'foreign' argument to your AM_INIT_AUTOMAKE call in* ***configure.ac*** *instead:*
> >
> > ```
> > AM_INIT_AUTOMAKE([1.10 no-define foreign])
> > ```
>
> ***Telling automake about your source files***
>
> > *Use lines like the following to name your program and list its source files:*
> >
> > ```
> > bin_PROGRAMS = hello
> > hello_SOURCES = hello.h hello.cc main.cc
> > ```
> > *Note that the name of the variable* `hello_SOURCES` *is prefixed*

> > > *with the name of the target defined by the assignment to `bin_PROGRAMS`. This is a common practice with **autoconf** and **automake**. Similarly, the prefix of the `bin_PROGRAMS` variable also has a meaning: It refers to the predefined variable `bindir`, which defines the installation directory for user-visible executable files.*
> > >
> > > *Thus, the first assignment instructs **automake** to generate build rules for an executable target file called "hello", which will be installed into the directory named by `bindir` on execution of `make install`. The second assignment instructs Automake to generate rules for building the "hello" target from the three C++ source files listed.*
>
> ### *The Whole Process*
>
> > *Assuming that you have written appropriate Makefile.am and configure.ac files (there are examples below), you should be able to build your project by entering the following commands:*
> >
> > - *'**touch NEWS README AUTHORS ChangeLog**'*
> > - *'**autoreconf --force --install**' - runs aclocal, autoconf, autoheader and automake in the right order to create config.h.in, Makefile.in, configure and a number of auxiliary files*
> > - *'**./configure**' - creates Makefile from Makefile.in and config.h from config.h.in*
> > - *'**make**'*
> >
> > *Just repeat the last three steps to completely reconfigure the project. Most projects kept in version control have an **autogen.sh** script that runs everything up to the configure step. This is done to provide a standard means for generating the build files from scratch, as generated files should never be put under version control. Release tarballs ship with pre-generated **configure** and **Makefile.in** files, so that the installer does not need to have **autoconf** or **automake** installed.*

Subdirectories

> *Project files should, of course, be organised in subdirectories. Ideally, all of the source files should be in a directory called 'src' in the project directory, with all of the other files (such as Makefiles, configure scripts, and READMEs) separate at the top. Projects which have several levels of directories are called 'deep' projects. The necessary steps are listed here, but you should look at the* Example Files *section to see them in context.*

Example Files

*Here are some example **configure.ac** and **Makefile.am** files. They are sufficient to manage a C++ project which only uses the C++ Standard Library.*

The example uses non-recursive Makefile rules *[8], which makes things simpler and allows faster builds. Many existing projects use recursive make, with a Makefile.am in each subdirectory. That generates a small non-installed library in each directory, which must be linked together (in the right order) into a binary. This is one of the main problems with recursive make.*

*See the **automake** and **autoconf** manuals in the* recommended reading *section for information on the macros and variable names used in these files.*

These examples are for a 'deep' project with the following structure:

```
helloworld_cc
    autogen.sh
    configure.ac
    Makefile.am
    src
        helloworld.h
        helloworld.cc
        main.cc
        foofiles
            foo.h
            foo.cc
```

configure.ac

```
AC_INIT([Helloworld C++], [0.5], [bug-report@hello.example.com],
             [helloworld_cc], [http://hello.example.com/])
AC_PREREQ([2.59])
AM_INIT_AUTOMAKE([1.10 -Wall no-define])
AC_CONFIG_HEADERS([config.h])
AC_PROG_CXX
AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```

Makefile.am

```
AUTOMAKE_OPTIONS = subdir-objects
ACLOCAL_AMFLAGS = ${ACLOCAL_FLAGS}

bin_PROGRAMS = hello
hello_SOURCES = src/hello.h src/hello.cc src/main.cc \
                src/foo/foo.h src/foo/foo.cc
```

> ```
> dist_noinst_SCRIPTS = autogen.sh
> ```
>
> *You may download the helloworld_cc-0.5.tar.gz [9] tar archive of the simple example project, or browse through the source at Gitorious [10].*

Recommended Reading

Translations

We know of the following translations of this article:

- Serbo-Croatian [11]

1. http://www.gnu.org/software/automake/manual/standards/Makefile-Conventions.html
2. http://gitorious.org/openismus-playground/helloworld_cc/
3. mailto:murrayc@openismus.com
4. http://www.elitecoders.de/mags/cscene/CS2/CS2-10.html
5. http://www.gnu.org/software/autoconf/manual/html_node/Initializing-configure.html
6. http://www.gnu.org/software/autoconf/manual/html_node/Configuration-Files.html
7. http://www.gnu.org/software/autoconf/manual/html_node/Output.html
8. http://www.flameeyes.eu/autotools-mythbuster/automake/nonrecursive.html
9. http://www.openismus.com/documents/linux/automake/helloworld_cc-0.5.tar.gz
10. http://gitorious.org/openismus-playground/helloworld_cc/trees/master
11. http://science.webhostinggeeks.com/automake-i-autoconf
12. http://creativecommons.org/licenses/by-sa/2.0/