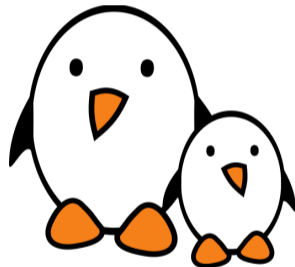




GNU Autotools: a tutorial

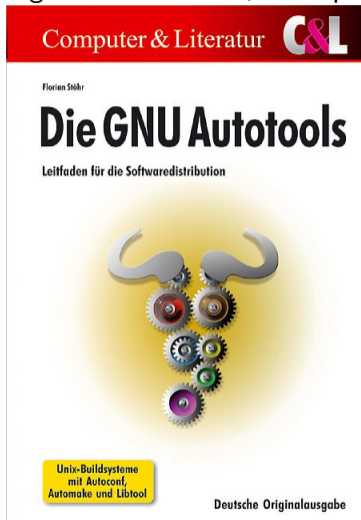


- ▶ CTO and Embedded Linux engineer at **Free Electrons**
 - ▶ Embedded Linux specialists.
 - ▶ Development, consulting and training.
 - ▶ <http://free-electrons.com>
- ▶ Contributions
 - ▶ **Kernel support for the Marvell Armada** ARM SoCs from Marvell
 - ▶ Major contributor to **Buildroot**, an open-source, simple and fast embedded Linux build system
- ▶ Living in **Toulouse**, south west of France

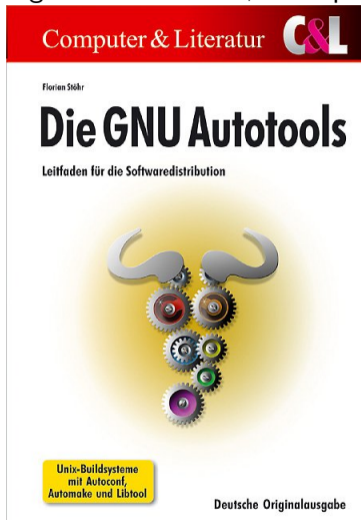


When talking about *autotools*, most people think:

When talking about *autotools*, most people think:



When talking about *autotools*, most people think:



But this is a German book, really about the autotools!



Autotools, why?

- ▶ Yes, the *autotools* are old
- ▶ Yes, they have their pain points
- ▶ Yes, people hate them
- ▶ Due to this, people tend to roll-their-own, and roll-their-own build systems tend to be even worse than the *autotools*
- ▶ But
 - ▶ They bring a number of very useful benefits
 - ▶ They are not that complicated when you take the time to get back to the basics



Autotools: benefits

- ▶ Standardized build procedure and behavior: users know how to build things that use the *autotools*
 - ▶ Good for human users, but also for build systems
- ▶ Proper handling for diverted installation
 - ▶ I.e. build with `prefix=/usr`, but divert the installation to another directory. Needed for cross-compilation.
- ▶ Built-in support for out-of-tree build
- ▶ Built-in handling of dependencies on header files
- ▶ Support for cross-compilation aspects
- ▶ Somewhat esoteric, but standardized languages used
 - ▶ Learn once, use for many projects
 - ▶ New contributors are more likely to know the *autotools* than your own custom thing
- ▶ Of course, there are alternatives, *CMake* being the most interesting and widely used.



Disclaimer

- ▶ I am not an autotools expert
- ▶ I don't know the internals of autotools, only their usage
- ▶ This tutorial will only cover the basics aspects
 - ▶ Sufficient to understand the autoconf/automake documentation
 - ▶ Sufficient to understand most existing build systems
- ▶ Won't cover many advanced aspects



Autotools tutorial: agenda

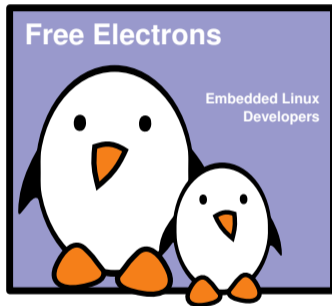
1. User point of view
2. *autoconf* basics
3. *automake* basics
4. *autoconf* advanced
5. *automake* advanced



User point of view

Thomas Petazzoni

© Copyright 2004-2016, Free Electrons.
Creative Commons BY-SA 3.0 license.
Corrections, suggestions, contributions and translations are welcome!



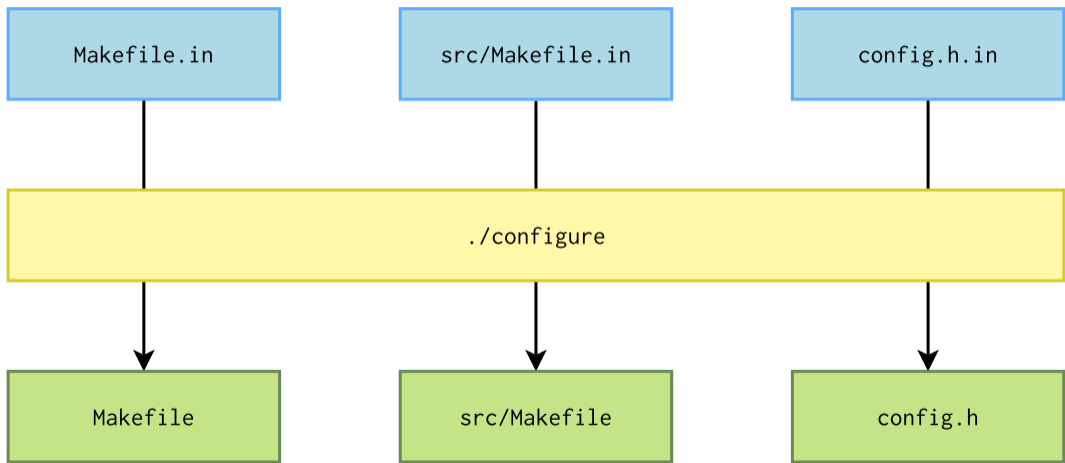


Using *autotools* based packages

- ▶ The basic steps to build an *autotools* based software component are:
 - 1. Configuration**
`./configure`
Will look at the available build environment, verify required dependencies, generate Makefiles and a `config.h`
 - 2. Compilation**
`make`
Actually builds the software component, using the generated Makefiles.
 - 3. Installation**
`make install`
Installs what has been built.



What is configure doing?





Standard Makefile targets

- ▶ `all`, builds everything. The default target.
- ▶ `install`, installs everything that should be installed.
- ▶ `install-strip`, same as `install`, but then strips debugging symbols
- ▶ `uninstall`
- ▶ `clean`, remove what was built
- ▶ `distclean`, same as `clean`, but also removes the generated *autotools* files
- ▶ `check`, run the test suite
- ▶ `installcheck`, check the installation
- ▶ `dist`, create a tarball



Standard filesystem hierarchy

- ▶ `prefix`, defaults to `/usr/local`
 - ▶ `exec-prefix`, defaults to `prefix`
 - ▶ `bindir`, for programs, defaults to `exec-prefix/bin`
 - ▶ `libdir`, for libraries, defaults to `exec-prefix/lib`
- ▶ `includedir`, for headers, defaults to `prefix/include`
- ▶ `datarootdir`, defaults to `prefix/share`
 - ▶ `datadir`, defaults to `datarootdir`
 - ▶ `mandir`, for man pages, defaults to `datarootdir/man`
 - ▶ `infodir`, for info documents, defaults to `datarootdir/info`
- ▶ `sysconfdir`, for configuration files, defaults to `prefix/etc`
- ▶ `--<option>` available for each of them
 - ▶ E.g: `./configure --prefix=~ /sys/`



Standard configuration variables

- ▶ CC, C compiler command
- ▶ CFLAGS, C compiler flags
- ▶ CXX, C++ compiler command
- ▶ CXXFLAGS, C++ compiler flags
- ▶ LDFLAGS, linker flags
- ▶ CPPFLAGS, C/C++ preprocessor flags
- ▶ and many more, see `./configure --help`
- ▶ E.g: `./configure CC=arm-linux-gcc`



System types: build, host, target

- ▶ *autotools* identify three **system types**:
 - ▶ **build**, which is the system where the build takes place
 - ▶ **host**, which is the system where the execution of the compiled code will take place
 - ▶ **target**, which is the system for which the program will generate code. This is only used for compilers, assemblers, linkers, etc.
- ▶ Corresponding `--build`, `--host` and `--target` *configure* options.
 - ▶ They are all automatically *guessed* to the current machine by default
 - ▶ `--build`, generally does not need to be changed
 - ▶ `--host`, must be overridden to do cross-compilation
 - ▶ `--target`, needs to be overridden if needed (to generate a cross-compiler, for example)
- ▶ Arguments to these options are *configuration names*, also called *system tuples*



System type: native compilation example

Demo
(based on the kmod source code)



Cross-compilation

- ▶ By default, *autotools* will guess the **host** machine as being the current machine
- ▶ To cross-compile, it must be overridden by passing the `--host` option with the appropriate *configuration name*
- ▶ By default, *autotools* will try to use the cross-compilation tools that use the *configuration name* as their prefix.
- ▶ If not, the variables `CC`, `CXX`, `LD`, `AR`, etc. can be used to point to the cross-compilation tools.



Out of tree build

- ▶ *autotools* support out of tree compilation by default
- ▶ Consists in doing the build in a directory separate from the source directory
- ▶ Allows to:
 - ▶ Build different configurations without having to rebuild from scratch each time.
 - ▶ Not clutter the source directory with build related files
- ▶ To use out of tree compilation, simply run the configure script from another empty directory
 - ▶ This directory will become the build directory



Out of tree build: example

Demo



Diverted installation with DESTDIR

- ▶ By default, `make install` installs to the directories given in `--prefix` and related options.
- ▶ In some situations, it is useful to *divert* the installation to another directory
 - ▶ Cross-compilation, where the build machine is not the machine where applications will be executed.
 - ▶ Packaging, where the installation needs to be done in a temporary directory.
- ▶ Achieved using the `DESTDIR` variable.

Demo!



--prefix or DESTDIR ?

- ▶ `--prefix` and `DESTDIR` are often misunderstood
- ▶ `--prefix` is the location where the programs/libraries will be placed when executed on the *host machine*
- ▶ `DESTDIR` is a way of temporarily diverting the installation to a different location.
- ▶ For example, if you use `--prefix=/home/<foo>/sys/usr`, then binaries/libraries will look for icons in `/home/<foo>/sys/usr/share/icons`
 - ▶ Good for native installation in `/home/<foo>/sys`
 - ▶ **Bad** for cross-compilation where the binaries will ultimately be in `/usr`



--prefix or DESTDIR use cases

- ▶ Native compilation, install system-wide in `/usr`

```
$$ ./configure --prefix=/usr
$$ make
$$ sudo make install
```

- ▶ Native compilation, install in a user-specific directory:

```
$ ./configure --prefix=/home/<foo>/sys/
$ make
$ make install
```

- ▶ Cross-compilation, install in `/usr`, diverted to a temporary directory where the system for the target is built

```
$$ ./configure --prefix=/usr
$$ make
$$ make DESTDIR=/home/<foo>/target-rootfs/ install
```



Analyzing issues

- ▶ `autoconf` keeps a log of all the tests it runs in a file called `config.log`
- ▶ Very useful for analysis of `autoconf` issues
- ▶ It contains several sections: *Platform*, *Core tests*, *Running config.status*, *Cache variables*, *Output variables*, *confdefs.h*
- ▶ The end of the *Core tests* section is usually the most interesting part
 - ▶ This is where you would get more details about the reason of the *configure* script failure
- ▶ At the beginning of `config.log` you can also see the `./configure` line that was used, with all options and environment variables.

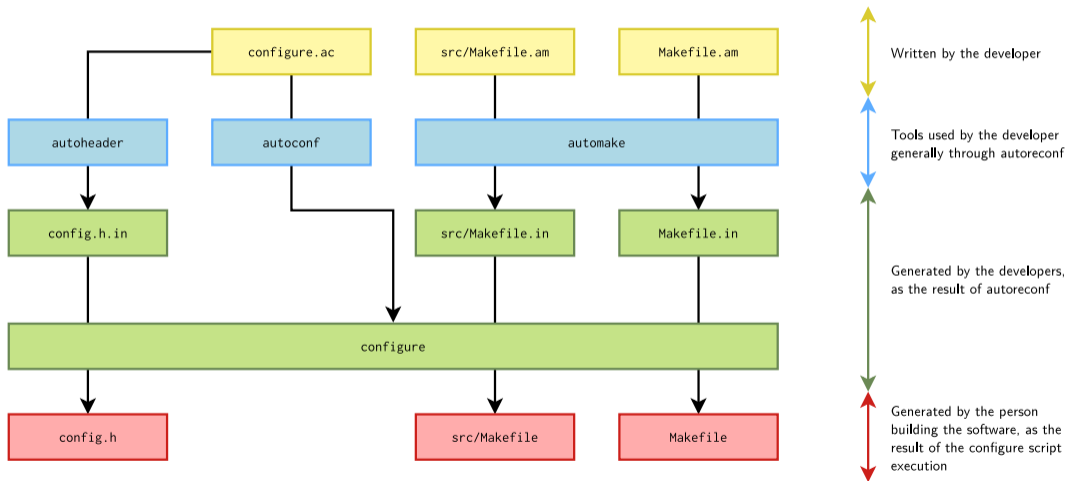


autotools: *autoconf* and *automake*

- ▶ The `configure` script is a shell script generated from `configure.ac` by a program called `autoconf`
 - ▶ `configure.ac` used to be named `configure.in` but this name is now deprecated
 - ▶ Written in shell script, augmented with numerous `m4` macros
- ▶ The `Makefile.in` are generated from `Makefile.am` files by a program called `automake`
 - ▶ Uses special `make` variables that are expanded in standard `make` constructs
- ▶ Some auxilliary tools like `autoheader` or `aclocal` are also used
 - ▶ `autoheader` is responsible for generating the *configuration header* template, `config.h.in`
- ▶ Generated files (`configure`, `Makefile.in`, `Makefile`) should not be modified.
 - ▶ Reading them is also very difficult. Read the real source instead!



Overall organization





Cache variables

- ▶ Each test done by a `configure.ac` script is associated with a *cache variable*
- ▶ The list of such variables and their values is visible in `config.log`:

```
## ----- ##  
## Cache variables. ##  
## ----- ##  
ac_cv_build=x86_64-unknown-linux-gnu  
ac_cv_c_compiler_gnu=yes  
[...]  
ac_cv_path_SED=/bin/sed
```

- ▶ If the autodetected value is not correct for some reason, you can override any of these variables in the environment:

```
$ ac_cv_path_SED=/path/to/sed ./configure
```

- ▶ This is sometimes useful when cross-compiling, since some tests are not always cross-compilation friendly.



- ▶ In general:
 - ▶ When a software is published as a *tarball*, the `configure` script and `Makefile.in` files are already generated and part of the tarball.
 - ▶ When a software is published through *version control system*, only the real sources `configure.ac` and `Makefile.am` are available.
- ▶ There are some exceptions (like tarballs not having pre-generated `configure/Makefile.in`)
- ▶ Do not version control generated files!



Regenerating *autotools* files: autoreconf

- ▶ To generate all the files used by *autotools*, you could call `automake`, `autoconf`, `aclocal`, `autoheader`, etc. manually.
 - ▶ But it is not very easy and efficient.
- ▶ A tool called `autoreconf` automates this process
 - ▶ Useful option: `-i` or `--install`, to ask `autoreconf` to copy missing auxiliary files
- ▶ Always use `autoreconf`!



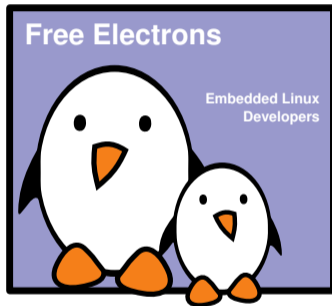
autoconf basics

Thomas Petazzoni

© Copyright 2004-2016, Free Electrons.

Creative Commons BY-SA 3.0 license.

Corrections, suggestions, contributions and translations are welcome!





- ▶ Really a shell script
- ▶ Processed through the `m4` preprocessor
- ▶ Shell script augmented with special constructs for portability:
 - ▶ `AS_IF` instead of shell `if ... then .. fi`
 - ▶ `AS_CASE` instead of shell `case ... esac`
 - ▶ etc.
- ▶ *autoconf* provides a large set of *m4* macros to perform most of the usual tests
- ▶ Make sure to quote macro arguments with `[]`



Minimal configure.ac

configure.ac

```
AC_INIT([hello], [1.0])  
AC_OUTPUT
```

▶ AC_INIT

- ▶ Every configure script must call `AC_INIT` before doing anything else that produces output.
- ▶ Process any command-line arguments and perform initialization and verification.
- ▶ Prototype:
`AC_INIT (package, version, [bug-report], [tarname], [url])`

▶ AC_OUTPUT

- ▶ Every `configure.ac`, should finish by calling `AC_OUTPUT`.
- ▶ Generates and runs `config.status`, which in turn creates the makefiles and any other files resulting from configuration.



Minimal configure.ac example

Demo 01



Additional basic macros

- ▶ `AC_PREREQ`
 - ▶ Verifies that a recent enough version of *autoconf* is used
 - ▶ `AC_PREREQ([2.68])`
- ▶ `AC_CONFIG_SRCDIR`
 - ▶ Gives the path to one source file in your project
 - ▶ Allows *autoconf* to check that it is really where it should be
 - ▶ `AC_CONFIG_SRCDIR([hello.c])`
- ▶ `AC_CONFIG_AUX_DIR`
 - ▶ Tells *autoconf* to put the auxiliary build tools it requires in a different directory, rather than the one of `configure.ac`
 - ▶ Useful to keep cleaner build directory



Demo 02



Checking for basic programs

- ▶ `AC_PROG_CC`, makes sure a C compiler is available
- ▶ `AC_PROG_CXX`, makes sure a C++ compiler is available
- ▶ `AC_PROG_AWK`, `AC_PROG_GREP`, `AC_PROG_LEX`, `AC_PROG_YACC`, etc.



Checking for basic programs: example

Demo 03



AC_CONFIG_FILES

- ▶ `AC_CONFIG_FILES (file..., [cmds], [init-cmds])`
- ▶ Make `AC_OUTPUT` create each file by copying an input `file` (by default `file.in`), substituting the *output variable values*.
- ▶ Typically used to turn the Makefile templates `Makefile.in` files into final `Makefile`.
- ▶ Example:
`AC_CONFIG_FILES([Makefile src/Makefile])`
- ▶ `cmds` and `init-cmds` are rarely used, see the *autoconf* documentation for details.



Output variables

- ▶ *autoconf* will replace `@variable@` constructs by the appropriate values in files listed in `AC_CONFIG_FILES`
- ▶ Long list of standard variables replaced by *autoconf*
- ▶ Additional shell variables declared in `configure.ac` can be replaced using `AC_SUBST`
- ▶ The following three examples are equivalent:

```
AC_SUBST([FOO], [42])
```

```
FOO=42  
AC_SUBST([FOO])
```

```
AC_SUBST([FOO])  
FOO=42
```



Demo 04



configure.ac: a shell script

- ▶ It is possible to include normal shell constructs in `configure.ac`
- ▶ Beware to not use *bashisms*: use only POSIX compatible constructs
- ▶ Most configure scripts use directly shell constructs, but `AS_ECHO`, `AS_IF`, etc. are available.

Demo 05 and 05b



Writing Makefile.in?

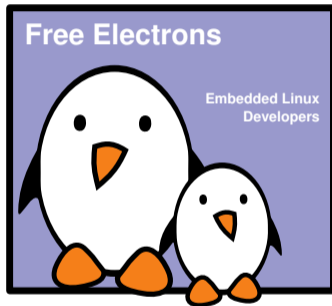
- ▶ At this point, we have seen the very basics of *autoconf* to perform the configuration side of our software
- ▶ We could use `AC_CONFIG_FILES` to generate `Makefile` from `Makefile.in`
- ▶ However, writing a `Makefile.in` properly is not easy, especially if you want to:
 - ▶ be portable
 - ▶ automatically handle dependencies
 - ▶ support conditional compilation, out-of-tree build, diverted installation, cross-compilation, etc.
- ▶ For these reasons, `Makefile.in` are typically not written manually, but generated by *automake* from a `Makefile.am` file



automake basics

Thomas Petazzoni

© Copyright 2004-2016, Free Electrons.
Creative Commons BY-SA 3.0 license.
Corrections, suggestions, contributions and translations are welcome!





Makefile.am language

- ▶ Really just a `Makefile`
 - ▶ You can include regular *make* code
- ▶ Augmented with *automake* specific constructs that are expanded into regular *make* code
- ▶ For most situations, the *automake* constructs are sufficient to express what needs to be built



Makefile.am minimal example

- ▶ The minimal example of `Makefile.am` to build just one C file into a program is only two lines:

Makefile.am

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

- ▶ Will compile `main.c` to `main.o`
- ▶ And link `hello.o` into the `hello` executable
- ▶ Which will be installed in `$prefix/bin`



Enabling *automake* in `configure.ac`

- ▶ To enable *automake* usage in `configure.ac`, you need:
 - ▶ A call to `AM_INIT_AUTOMAKE`
 - ▶ Generate the `Makefile` using `AC_CONFIG_FILES`
- ▶ *automake* will generate the `Makefile.in` at *autoreconf* time, and *configure* will generate the final `Makefile`

`configure.ac`

```
AC_INIT([hello], [1.0])
AM_INIT_AUTOMAKE([foreign 1.13])
AC_PROG_CC
AC_CONFIG_FILES([Makefile])
AC_OUTPUT
```



Demo 06



- ▶ `AM_INIT_AUTOMAKE([OPTIONS])`
- ▶ Interesting options:
 - ▶ `foreign`, tells *automake* to not require all the GNU Coding Style files such as `NEWS`, `README`, `AUTHORS`, etc.
 - ▶ `dist-bzip2`, `dist-xz`, etc. tell *automake* which tarball format should be generated by `make dist`
 - ▶ `subdir-objects` tells *automake* that the objects are placed into the subdirectory of the build directory corresponding to the subdirectory of the source file
 - ▶ `version`, e.g `1.14.1`, tells the minimal *automake* version that is expected



Makefile.am syntax

- ▶ An *automake* parsable `Makefile.am` is composed of **product list variables**:

```
bin_PROGRAMS = hello
```

- ▶ And **product source variables**:

```
hello_SOURCES = main.c
```



Product list variables

```
[modifier-list]prefix_PRIMARY = product1 product2 ...
```

- ▶ `prefix` is the installation prefix, i.e. where it should be installed
 - ▶ All `*dir` variables from *autoconf* can be used, without their `dir` suffix: use `bin` for `bindir`
 - ▶ E.g.: `bindir`, `libdir`, `includedir`, `datadir`, etc.
- ▶ `PRIMARY` describes what type of things should be built:
 - ▶ `PROGRAMS`, for executables
 - ▶ `LIBRARIES`, `LTLIBRARIES`, for libraries
 - ▶ `HEADERS`, for publicly installed header files
 - ▶ `DATA`, arbitrary data files
 - ▶ `PYTHON`, `JAVA`, `SCRIPTS`
 - ▶ `MANS`, `TEXINFOS`, for documentation
- ▶ After the `=` sign, list of products to be generated



Product source variables

```
[modifier-list]product_SOURCES = file1 file2 ...
```

- ▶ The `product` is the normalized name of the product, as listed in a *product list variable*
 - ▶ The normalization consists in replacing special characters such as `.` or `+` by `_`. For example, `libfoo+.a` in a *product list variable* gives the `libfoo__a_SOURCES` product source variable.
- ▶ `_SOURCES` is always used, it's not like a configurable *primary*.
 - ▶ Contains the list of files containing the source code for the product to be built.
 - ▶ Both source files *and* header files should be listed.



More complicated automake example

Demo 07



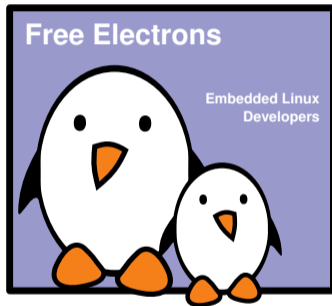
autoconf advanced

Thomas Petazzoni

© Copyright 2004-2016, Free Electrons.

Creative Commons BY-SA 3.0 license.

Corrections, suggestions, contributions and translations are welcome!





Configuration header

- ▶ Very often, C/C++ code needs to know the result of certain tests done by the `configure` script.
- ▶ A template C header file can be automatically generated by `autoheader`, generally named `config.h.in`
- ▶ The final header file is generated by `configure`, generally named `config.h`
- ▶ Declared using `AC_CONFIG_HEADERS`

`configure.ac` `extract`

```
AC_CONFIG_HEADERS([config.h])
```

Example `config.h`

```
/* Define if the complete vga libraries (vga, vga1) are installed */  
/* #undef HAVE_LIBVGA */  
  
/* Define to 1 if you have the <limits.h> header file. */  
#define HAVE_LIMITS_H 1
```



- ▶ `AC_DEFINE` allows to create C definitions in the *configuration header*
- ▶ `AC_DEFINE (variable, value, description)`

`configure.ac`

```
AC_DEFINE([FOOBAR], [42], [This is the foobar value])
```

Demo 08



Checking for functions

- ▶ You may need to check if certain functions are available and/or meet certain characteristics
- ▶ Family of `AC_FUNC_*` macros
 - ▶ `AC_FUNC_FORK`, `AC_FUNC_GETLOADAVG`, `AC_FUNC_MALLOC`, etc.
 - ▶ See *autoconf* manual for details
- ▶ `AC_CHECK_FUNC[S]` to check for generic functions
 - ▶ `AC_CHECK_FUNC` (`function`, [`action-if-found`], [`action-if-not-found`])
 - ▶ `AC_CHECK_FUNCS` (`function...`, [`action-if-found`], [`action-if-not-found`])
 - ▶ Results available
 - ▶ `ac_cv_func_<function>` variable in `configure.ac`
 - ▶ `HAVE_<FUNCTION>` defines in *configuration headers*



AC_CHECK_FUNCS() example

Demo 09



Checking for headers

- ▶ Much like `AC_FUNC_*` and `AC_CHECK_FUNC[S]`, but for headers
- ▶ Variety of `AC_HEADER_*` macros
 - ▶ Check the autoconf manual for details
- ▶ `AC_CHECK_HEADER[S]` for generic headers checking
 - ▶ `AC_CHECK_HEADER` (`header-file`, [`action-if-found`], [`action-if-not-found`], [`includes`])
 - ▶ `AC_CHECK_HEADERS` (`header-file...`, [`action-if-found`], [`action-if-not-found`], [`includes`])
 - ▶ Results available in:
 - ▶ `ac_cv_header_<header-file>` variable in `configure.ac`
 - ▶ `HAVE_<HEADER>_H` define in `config.h`



AC_CHECK_HEADERS example

```
configure.ac
```

```
[...]  
AC_CHECK_HEADERS([spawn.h],  
    [echo "Header spawn.h was found"; has_spawn=yes],  
    [echo "Header spawn.h was not found"])  
echo ${has_spawn}  
[...]
```

Execution of ./configure

```
$ ./configure  
[...]  
checking for spawn.h... yes  
Header spawn.h was found  
yes  
[...]
```



Checking for libraries

```
AC_SEARCH_LIBS (function, search-libs,  
               [action-if-found], [action-if-not-found],  
               [other-libraries])
```

- ▶ Search for a library defining `function`, by linking a simple program calling `function`
- ▶ Tries first with no library, and then with the different libraries in `search-libs`, one after the other.
- ▶ If a library is found, `-llibrary` is prepended to the `LIBS` variable, so programs will be linked against it. `action-if-found` is executed.
- ▶ If not, `action-if-not-found` is executed
- ▶ `other-libraries` allows to pass additional `-l<foo>` arguments that may be needed for the link test to succeed.
- ▶ Result in `ac_cv_search_<function>`



Demo 10



Other checks

- ▶ **Programs** with `AC_CHECK_PROGS`
 - ▶ `AC_CHECK_PROGS(PERL, [perl5 perl])`
- ▶ **Declarations** with `AC_CHECK_DECLS`
- ▶ **Structure members** with `AC_CHECK_MEMBERS`
- ▶ **Types** with `AC_CHECK_TYPES`
 - ▶ `AC_CHECK_TYPES(int8_t)`
- ▶ See the *autoconf* manual for details



Writing new tests

- ▶ You can create your own tests by pre-processing, compiling or linking small test programs:
 - ▶ Pre-processing test
`AC_PREPROC_IFELSE (input, [action-if-true], [action-if-false])`
 - ▶ Compiling test
`AC_COMPILE_IFELSE (input, [action-if-true], [action-if-false])`
 - ▶ Link test
`AC_LINK_IFELSE (input, [action-if-true], [action-if-false])`
- ▶ Input should be formatted with `AC_LANG_SOURCE` or `AC_LANG_PROGRAM`
- ▶ Runtime tests can also be created
 - ▶ Beware, by nature, they cannot work for cross-compilation!
 - ▶ `AC_RUN_IFELSE`



Writing new tests: AC_LINK_IFELSE

Demo 11



Printing messages

- ▶ When creating new tests, you may want to show messages, warnings, errors, etc.
- ▶ `AC_MSG_CHECKING (feature-description)`
 - ▶ Notify the user that configure is checking for a particular feature.
- ▶ `AC_MSG_RESULT (result-description)`
 - ▶ Notify the user of the results of a check
- ▶ `AC_MSG_NOTICE (message)`
 - ▶ Deliver the *message* to the user.
- ▶ `AC_MSG_ERROR (error-description, [exit-status = $?/1])`
 - ▶ Notify the user of an error that prevents configure from completing.
- ▶ `AC_MSG_WARN (problem-description)`
 - ▶ Notify the configure user of a possible problem.



Printing messages: example

Demo 11 continued



Cache variables

- ▶ Each test done by *autoconf* is normally associated to a **cache variable**.
 - ▶ Allows to speed-up the configure step by passing a cache file with pre-defined values.
 - ▶ Allows to override the results of tests if they are not correct for some reason
- ▶ `AC_CACHE_VAL(cache-id, commands-to-set-it)`, runs *commands* if *cache-id* is not already set. *commands* must set the *cache-id* variable and have no side-effect.
- ▶ `AC_CACHE_CHECK(message, cache-id, commands)`, wrapper around `AC_CACHE_VAL` to print the message.



Demo 11 further continued



Using external software

- ▶ When a package uses external software, `--with-<package>=<arg>` and `--without-<package>` options are generally offered to control usage of the external software.
- ▶ Implemented using the `AC_ARG_WITH` macro.

```
AC_ARG_WITH (package, help-string,  
            [action-if-given], [action-if-not-given])
```

- ▶ `package` gives the name of the option
- ▶ `help-string` is the help text, visible in `./configure --help`
- ▶ `action-if-given` is executed when the option is used, either positively (`--with`) or negatively (`--without`)
- ▶ `action-if-not-given` is executed when the option is not used
- ▶ `<arg>` available as `$withval` inside `action-if-given`, `$with_<package>` outside.



Package options

- ▶ When a package offers optional features, `--enable-<feature>` and `--disable-<feature>` options are generally offered to control the optional feature.
- ▶ Implemented using the `AC_ARG_ENABLE` macro.

```
AC_ARG_ENABLE (feature, help-string,  
              [action-if-given], [action-if-not-given])
```

- ▶ Usage very similar to the one of `AC_ARG_WITH`
- ▶ Value available as `$enableval` inside *action-if-given*, `$enable_<feature>` outside.



Formatting the help string

- ▶ To help formatting the help string, *autoconf* provides the `AS_HELP_STRING` macro
- ▶ Allows to properly align the different options in the `./configure --help` output

```
AS_HELP_STRING (left-hand-side, right-hand-side,  
               [indent-column = '26'], [wrap-column = '79'])
```



AC_ARG_ENABLE example

Demo 12



Using pkg-config with autoconf

- ▶ To find libraries, a much better solution than `AC_SEARCH_LIBS` is to use **pkg-config**
- ▶ pkg-config is a database of small text files, using the `.pc` extension, describing how to use a given library
 - ▶ installed in `usr/lib/pkgconfig` on most systems
 - ▶ installed by most modern libraries
- ▶ The `pkg-config` command line tool allows to query this database for the compiler and linker flags needed to use a given library.
- ▶ The `PKG_CHECK_MODULES` *autoconf* macro allows to query the pkg-config database.



The PKG_CHECK_MODULES macro

- ▶ Syntax:

```
PKG_CHECK_MODULES(prefix, list-of-modules,  
                  action-if-found, action-if-not-found)
```

- ▶ `prefix` will be used to create the `<prefix>_CFLAGS` and `<prefix>_LIBS` variables
 - ▶ Contain the pre-processor and linker flags to use the libraries listed in `list-of-modules`
 - ▶ Are already `AC_SUBST`ed, so can be used directly in `Makefile.am`
- ▶ `list-of-modules` is one or several `pkg-config` libraries
 - ▶ Can contain version specifiers, such as `foo >= 3 bar baz <= 4`
- ▶ Will exit with a failure if one of the dependencies is missing.



Demo 13

- ▶ `autoscan` is a program provided together with `autoconf`
- ▶ Scans the source tree in the current directory (or the one passed as argument)
- ▶ From that, `autoscan`:
 - ▶ Searches the source files for common portability problems
 - ▶ Checks for incompleteness of the `configure.ac` file, if any
 - ▶ Generates `configure.scan`, which can be used as a preliminary `configure.ac`



Additional m4 macros

- ▶ The core `autoconf` macros are installed in `/usr/share/autoconf/autoconf/`
- ▶ Additional macros can be installed by other packages in `/usr/share/aclocal`
 - ▶ Examples: `pkg.m4` (for `pkg-config`), `gpg-error.m4`, `iconv.m4`, etc.
- ▶ The **GNU Autoconf Archive** is a collection of more than 500 macros for `autoconf`
 - ▶ <http://www.gnu.org/software/autoconf-archive/>
 - ▶ Example: `AX_C_LONG_LONG`, *Provides a test for the existence of the long long int type and defines `HAVE_LONG_LONG` if it is found.*



Demo 14



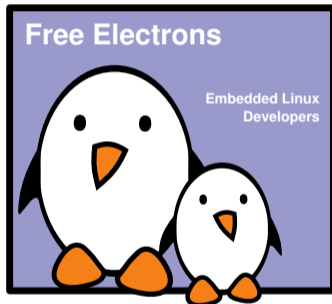
automake advanced

Thomas Petazzoni

© Copyright 2004-2016, Free Electrons.

Creative Commons BY-SA 3.0 license.

Corrections, suggestions, contributions and translations are welcome!





Subdirectories

- ▶ A project is often organized with multiple directories
- ▶ `automake` offers two options to support this:
 - ▶ **recursive make**, where a sub-call to `make` is made for sub-directories, and each directory has its own `Makefile.am`
 - ▶ **non-recursive make**, where there is a single `Makefile.am`, building everything
- ▶ **recursive make** used to be the norm, but has significant drawbacks
 - ▶ Performance for parallel building
 - ▶ *Recursive make considered harmful*,
<http://aegis.sourceforge.net/auug97.pdf>
- ▶ **non-recursive make** is more and more commonly used in modern projects
 - ▶ If the `Makefile.am` grows too large, one can use `include` to split it.



Recursive make

- ▶ The `SUBDIRS` variable in a `Makefile.am` indicates the sub-directories that contain other `Makefile.am`

`configure.ac`

```
AC_CONFIG_FILES([Makefile src/Makefile])
```

`Makefile.am`

```
SUBDIRS = src
```

`src/Makefile.am`

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



Non-recursive make

- ▶ The `AM_INIT_AUTOMAKE` macro accepts a `subdir-objects` argument
- ▶ If specified, allows a `Makefile.am` to reference code in another directory

`configure.ac`

```
AM_INIT_AUTOMAKE([subdir-objects])
AC_CONFIG_FILES([Makefile])
```

`Makefile.am`

```
bin_PROGRAMS = hello
hello_SOURCES = src/main.c
```



Demo 15 and 16



automake conditionals

- ▶ In order to use a conditional in a `Makefile.am`, it must be defined in the `configure.ac` script.
- ▶ Done using the `AM_CONDITIONAL(conditional, condition)` macro

`configure.ac`

```
AM_CONDITIONAL([DEBUG], [test "${debug}" = "true"])
```

`Makefile.am`

```
if DEBUG
...
else
...
endif
```



Usage of *automake* conditionals

You cannot use conditionals inside a variable definition

Non-working example

```
bin_PROGRAMS = \  
  bar \  
if DEBUG  
  baz \  
endif  
  foobar
```

You should instead use an intermediate variable

Working example

```
if DEBUG  
DEBUG_PROGS = baz  
endif  
  
bin_PROGRAMS = \  
  bar \  
  $(DEBUG_PROGS) \  
  foobar
```

Or the += assignment sign

Working example

```
bin_PROGRAMS = \  
  bar \  
  foobar  
  
if DEBUG  
bin_PROGRAMS += baz  
endif
```



Conditional example

Demo 17



Building shared libraries

- ▶ Building shared libraries is very different between Unix variants
- ▶ A specific tool, called `libtool`, was created to abstract away the differences between platforms.
- ▶ Concept called *libtool libraries*, using the `.la` suffix
- ▶ A libtool library can designate a static library, a shared library, or both.
 - ▶ `--{enable,disable}-{static,shared}` to select
- ▶ Libtool libraries declared using the `LTLIBRARIES` primary in a `Makefile.am`
- ▶ Typically used in conjunction with the `HEADERS` primary to install public headers.
- ▶ `configure.ac` must call the `LT_PREREQ` and `LT_INIT` macros



Libtool library example

configure.ac

```
[...]  
LT_PREREQ([2.4])  
LT_INIT  
[...]
```

Makefile.am

```
bin_PROGRAMS = hello  
hello_SOURCES = src/main.c  
  
lib_LTLIBRARIES = libmyhello.la  
libmyhello_la_SOURCES = lib/core.c  
include_HEADERS = lib/myhello.h
```




Libtool versioning

- ▶ Needed to support changes in the library interface
- ▶ Each system handles library versioning differently
- ▶ `libtool` does not use the traditional `<major>.<minor>.<revision>`
- ▶ It uses a more abstract representation, converted differently depending on the system on which you're building.
- ▶ `libtool` representation is `<current>:<revision>:<age>`
 - ▶ `current` is the interface number, incremented whenever the public interface changes
 - ▶ `revision` is incremented whenever the library source code is changed
 - ▶ `age` is incremented when new functions are added, reset to 0 when functions are removed
- ▶ Defined using `-version-info <current>:<revision>:<age>` in `<product>_LDFLAGS`



Demo 18



Global *automake* variables

- ▶ Variables that you can define in `Makefile.am`
 - ▶ Apply to the current `Makefile.am`
 - ▶ Affect all products described in the current `Makefile.am`
- ▶ `AM_CPPFLAGS`, default pre-processor flags
- ▶ `AM_CFLAGS`, default compiler flags
- ▶ `AM_LDFLAGS`, default linker flags
- ▶ `LDADD`, libraries not detected by *configure* that we should link with
- ▶ Do not set `CPPFLAGS`, `CFLAGS` and `LDFLAGS`, so that they can be passed in the environment by users

Example

```
LDADD = $(top_builddir)/glib/libglib-2.0.la
AM_CPPFLAGS = $(gmodule_INCLUDES) $(GLIB_DEBUG_FLAGS)
AM_CFLAGS = -g
```



Per product variables

- ▶ `<product>_SOURCES`, list of source files
- ▶ `<product>_LDADD`, libraries to link with
- ▶ `<product>_CPPFLAGS`, pre-processor flags, overrides `AM_CPPFLAGS`
- ▶ `<product>_CFLAGS`, compiler flags, overrides `AM_CFLAGS`
- ▶ `<product>_LDFLAGS`, linker flags, overrides `AM_LDFLAGS`

Example

```
LDADD = $(top_builddir)/glib/libglib-2.0.1a

module_test_LDADD = $(top_builddir)/gmodule/libgmodule-2.0.1a $(LDADD)
module_test_LDFLAGS = $(G_MODULE_LDFLAGS)
slice_threadinit_LDADD = $(top_builddir)/gthread/libgthread-2.0.1a $(LDADD)
```



Useful variables

- ▶ Autoconf provides several variables that can be useful in your `Makefile.am`:
 - ▶ `top_srcdir`, the relative path to the top of the source tree
 - ▶ `srcdir`, the relative path to the directory that contains the current `Makefile`
 - ▶ `top_builddir`, the relative path to the top of the build tree
 - ▶ `builddir`, the current directory
 - ▶ `abs_top_srcdir`, `abs_srcdir`, `abs_top_builddir`, `abs_builddir`, absolute variants of the previous variables
- ▶ Example usage: library code in `lib/`, header files in `include/`:

`lib/Makefile.am`

```
lib_LTLIBRARIES = libhello.la
libhello_la_SOURCES = ...
libhello_la_CPPFLAGS = -I$(top_srcdir)/include
```

Demo 19



Silent rules

- ▶ By default, *automake* generate Makefiles that displays the full compilation commands
- ▶ Using the `AM_SILENT_RULES`, you can get a slimmer build output
- ▶ By default, the output remains verbose, but can be silenced by passing the `V=0` variable.
- ▶ If `AM_SILENT_RULES([yes])` is used, the output is quiet by default, and verbose if `V=1` is passed.

```
$ make
CC      lib/core.lo
CCLD    libmyhello.la
CC      src/main.o
CCLD    hello

$ make V=1
[...]
libtool: link: (cd ".libs" && rm -f "libmyhello.so.0" && ln -s "libmyhello.so.0.0.0" ...)
libtool: link: (cd ".libs" && rm -f "libmyhello.so" && ln -s "libmyhello.so.0.0.0" ...)
libtool: link: ar cru .libs/libmyhello.a lib/core.o
libtool: link: ranlib .libs/libmyhello.a
[...]
```

Demo 20



make dist

- ▶ `make dist` generates a tarball to release the software
- ▶ All files listed in `_SOURCES` variables are automatically included, as well as the necessary *autotools* files
- ▶ Additional files can be added to the distribution using the `EXTRA_DIST` variable in `Makefile.am`:

Makefile.am

```
# These files are used in the preparation of a release
EXTRA_DIST += \
  PrepareRelease \
  CheckMan \
  CleanTxt \
  [...]
```

- ▶ Distribution can also be controlled using the `dist` and `nodist` *automake* product modifiers:

Makefile.am

```
nodist_include_HEADERS += pcrepparg.h
dist_doc_DATA = doc/pcre.txt
```



Macro directory

- ▶ By default, all the third-party *autoconf* macros get copied into the (very large) `aclocal.m4` file.
- ▶ It is possible to get some of the third-party macros copied to individual files in a separate directory, which is nicer.
- ▶ Directory declared using `AC_CONFIG_MACRO_DIR`, generally named `m4` by convention:

`configure.ac`

```
AC_CONFIG_MACRO_DIR([m4])
```

- ▶ The `ACLOCAL_AMFLAGS` in `Makefile.am` should also be adjusted:

`Makefile.am`

```
ACLOCAL_AMFLAGS = -I m4
```

- ▶ For now, mainly used by `libtool` for its own *m4* macros.



Auxiliary directory

- ▶ The *auxiliary files* generated by *autotools* such as `compile`, `config.guess`, `config.sub`, `depcomp`, etc. are by default in the main directory of the source tree.
- ▶ This clutters the main directory with lots of files, which may not be very pleasant.
- ▶ `AC_CONFIG_AUX_DIR` allows to customize where these files are generated:

`configure.ac`

```
AC_CONFIG_AUX_DIR([build-aux])
```

- ▶ One condition: it must be placed before the calls to `AM_INIT_AUTOMAKE` and `LT_INIT`



Demo 21

Questions?

Thomas Petazzoni

`thomas.petazzoni@free-electrons.com`

Slides under CC-BY-SA 3.0

`http://free-electrons.com/pub/conferences/2016/elc/petazzoni-autotools-tutorial/`

Demos: `https://github.com/tpetazzoni/autotools-demo`