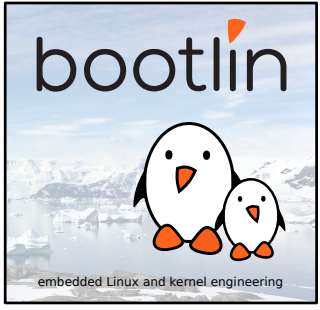
Autotools training

Latest update: August 22, 2025.

Document updates and training details:
https://bootlin.com/training/autotools

Corrections, suggestions, contributions and translations are welcome!
Send them to feedback@bootlin.com

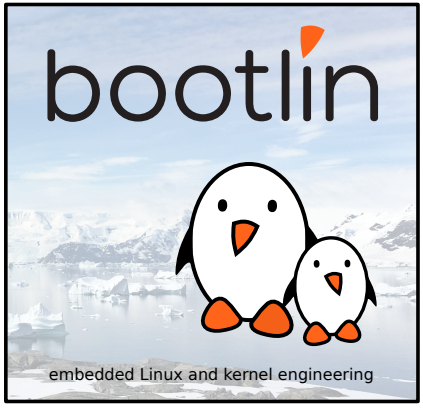


Autotools training

- These slides are the training materials for Bootlin's *Autotools* training course.
- If you are interested in following this course with an experienced Bootlin trainer, we offer:
  - **Public online sessions**, opened to individual registration. Dates announced on our site, registration directly online.
  - **Dedicated online sessions**, organized for a team of engineers from the same company at a date/time chosen by our customer.
  - **Dedicated on-site sessions**, organized for a team of engineers from the same company, we send a Bootlin trainer on-site to deliver the training.
- Details and registrations: https://bootlin.com/training/autotools
- Contact: training@bootlin.com

Icon by Eucalyp, Flaticon

About Bootlin



Bootlin introduction

- Engineering company
  - In business since 2004
  - Before 2018: *Free Electrons*
- Team based in France and Italy
- Serving **customers worldwide**
- **Highly focused and recognized expertise**
  - Embedded Linux
  - Linux kernel
  - Embedded Linux build systems
- **Strong open-source** contributor
- Activities
  - **Engineering** services
  - **Training** courses
- https://bootlin.com

bootlin

Bootlin engineering services

Bootloader / firmware development

U-Boot, Barebox, OP-TEE, TF-A, .../

Linux kernel porting and driver development

Linux BSP development, maintenance and upgrade

Embedded Linux build systems

Yocto, OpenEmbedded, Buildroot, ...

Embedded Linux integration

Boot time, real-time, security, multimedia, networking

Open-source upstreaming

Get code integrated in upstream Linux, U-Boot, Yocto, Buildroot, ...

Bootlin training courses

Embedded Linux system development On-site: 4 or 5 days Online: 7 * 4 hours	Linux kernel driver development On-site: 5 days Online: 7 * 4 hours	Yocto Project system development On-site: 3 days Online: 4 * 4 hours	Buildroot system development On-site: 3 days Online: 5 * 4 hours	Embedded Linux networking On-site: 3 days Online: 4 * 4 hours
Understanding the Linux graphics stack On-site: 2 days Online: 4 * 4 hours	Embedded Linux audio On-site: 2 days Online: 4 * 4 hours	Real-Time Linux with PREEMPT_RT On-site: 2 days Online: 3 * 4 hours	Linux debugging, tracing, profiling and performance analysis On-site: 3 days Online: 4 * 4 hours	

All our training materials are freely available
under a free documentation license (CC-BY-SA 3.0)
See https://bootlin.com/training/

Bootlin, an open-source contributor

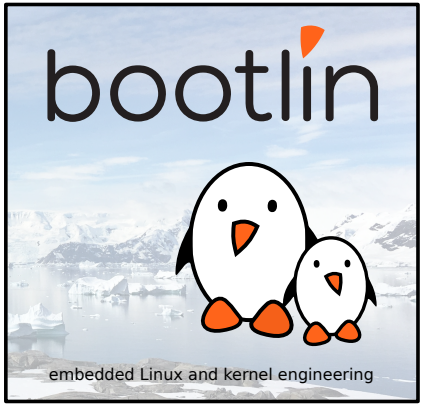
- Strong contributor to the **Linux** kernel
  - In the top 30 of companies contributing to Linux worldwide
  - Contributions in most areas related to hardware support
  - Several engineers maintainers of subsystems/platforms
  - 9000 patches contributed
  - https://bootlin.com/community/contributions/kernel-contributions/
- Contributor to **Yocto Project**
  - Maintainer of the official documentation
  - Core participant to the QA effort
- Contributor to **Buildroot**
  - Co-maintainer
  - 6000 patches contributed
- Significant contributions to U-Boot, OP-TEE, Barebox, etc.
- Fully **open-source training materials**

Bootlin on-line resources

- Website with a technical blog:
  https://bootlin.com
- Engineering services:
  https://bootlin.com/engineering
- Training services:
  https://bootlin.com/training
- LinkedIn:
  https://www.linkedin.com/company/bootlin
- Elixir - browse Linux kernel sources on-line:
  https://elixir.bootlin.com

Icon by Freepik, Flaticon

Generic course information



Training quiz and certificate

- You have been given a quiz to test your knowledge on the topics covered by the course. That's not too late to take it if you haven't done it yet!
- At the end of the course, we will submit this quiz to you again. That time, you will see the correct answers.
- It allows Bootlin to assess your progress thanks to the course. That's also a kind of challenge, to look for clues throughout the lectures and labs / demos, as all the answers are in the course!
- Another reason is that we only give training certificates to people who achieve at least a 50% score in the final quiz **and** who attended all the sessions.

Participate!

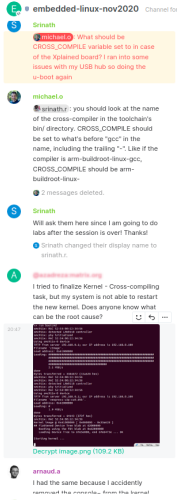
During the lectures...

- Don't hesitate to ask questions. Other people in the audience may have similar questions too.
- Don't hesitate to share your experience too, for example to compare Linux with other operating systems you know.
- Your point of view is most valuable, because it can be similar to your colleagues' and different from the trainer's.
- In on-line sessions
  - Please always keep your camera on!
  - Also make sure your name is properly filled.
  - You can also use the "Raise your hand" button when you wish to ask a question but don't want to interrupt.
- All this helps the trainer to engage with participants, see when something needs clarifying and make the session more interactive, enjoyable and useful for everyone.

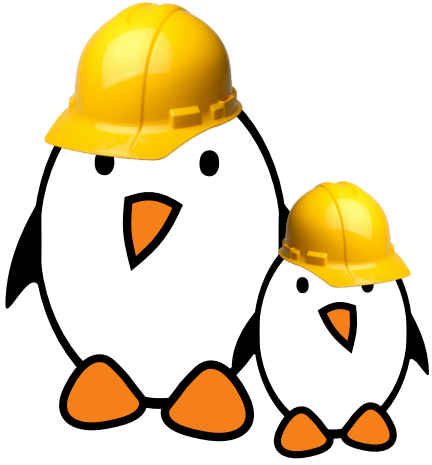
Collaborate!

As in the Free Software and Open Source community, collaboration between participants is valuable in this training session:

- Use the dedicated Matrix channel for this session to add questions.
- If your session offers practical labs, you can also report issues, share screenshots and command output there.
- Don't hesitate to share your own answers and to help others especially when the trainer is unavailable.
- The Matrix channel is also a good place to ask questions outside of training hours, and after the course is over.



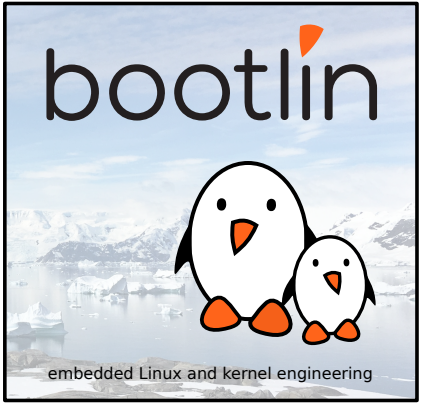
Practical lab - Training Setup



Prepare your lab environment

- Download and extract the lab archive

Autotools usage



Why do we need *autotools*?

- **Portability** accross UNIX systems, architectures, Linux distributions
  - Some C functions do not exist everywhere, or have different names or prototypes, can behave differently
  - Header files can be organized differently
  - All libraries may not be available everywhere
- **Standardized** build procedure
  - Standard options
  - Standard environment variables
  - Standard behavior

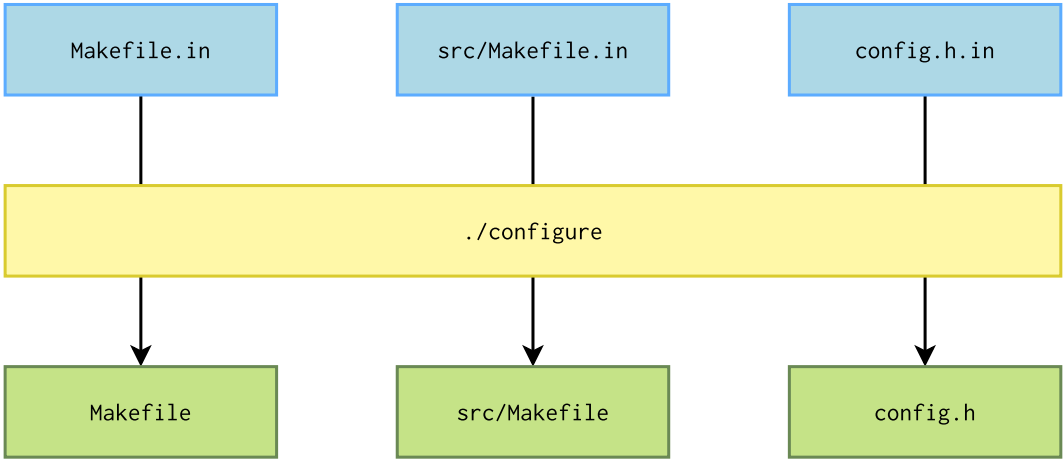
Alternatives to *autotools*

- Regular *Makefiles*
  - Not very portable
  - No configuration tests, or options
  - Hard to take into account all dependencies (e.g. dependencies on header files)
  - No standardized behavior
- CMake
  - A more modern build system
  - One language, instead of several for *autotools*
  - More straightforward to use and understand
  - Much less widely used than *autotools*, but growing in popularity
  - Also generates Makefiles, like *autotools*

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 `Makefile`s 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: *configuration names*

- A string identifying a combination of architecture, operating system, ABI and C library
- General format: `<arch>-<vendor>-<kernel>-<operating_system>`
  - `<arch>` is the type of processor, i.e. `arm`, `i686`, etc.
  - `<vendor>` is a free form string, which can be omitted
  - `<kernel>` is always `linux` when working with Linux systems, or `none` for bare metal systems
  - `<operating_system>` generally identifies the C library and ABI, i.e. `gnu`, `gnueabi`, `eabi`, `gnueabihf`, `uclibcgnueabihf`
- Also often used as the *prefix* for cross-compilation tools.
- Examples
  - `x86_64-amd-linux-gnu`
  - `powerpc-mentor-linux-gnu`
  - `armeb-linux-gnueabihf`
  - `i486-linux-musl`

System type: native compilation example

```
$ ./configure
[...]
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking for gcc... gcc
[...]
checking how to run the C preprocessor... gcc -E
[...]
```

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.

System type: cross compilation example

```
$ which arm-linux-gnueabihf-gcc
/usr/bin/arm-linux-gnueabihf-gcc
$ ./configure --host=arm-linux-gnueabihf
[...]
checking build system type... x86_64-unknown-linux-gnu
checking host system type... arm-unknown-linux-gnueabihf
checking for arm-linux-gnueabihf-gcc... arm-linux-gnueabihf-gcc
[...]
checking how to run the C preprocessor... arm-linux-gnueabihf-gcc -E
[...]
```

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

```
strace-4.9 $ ls
configure configure.ac Makefile.am system.c NEWS
AUTHORS   COPYING     file.c       ioprio.c config.h
strace-4.9 $ mkdir ../strace-build-x86 ../strace-build-arm
strace-4.9 $ cd ../strace-build-x86
strace-build-x86 $ ../strace-4.9/configure
[...]
strace-build-x86 $ make
[...]
strace-build-x86 $ cd ../strace-build-arm
strace-build-arm $ ../strace-4.9/configure --host=arm-linux-gnueabihf
[...]
strace-build-arm $ make
[...]
```

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.

```
strace-4.9 $ make DESTDIR=/tmp/test install
[...]
strace-4.9 $ find  /tmp/test/ -type f
/tmp/test/usr/local/share/man/man1/strace.1
/tmp/test/usr/local/bin/strace-log-merge
/tmp/test/usr/local/bin/strace-graph
/tmp/test/usr/local/bin/strace
```

--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.

config.log example

```
$ ./configure ...
[...]
checking for TIFFFlushData in -ltiff34... no
configure: WARNING: *** TIFF loader will not be built (TIFF library not found) ***
configure: error:
*** Checks for TIFF loader failed. You can build without it by passing
*** --without-libtiff to configure but some programs using GTK+ may
*** not work properly

$ cat config.log
[...]
configure:18177: .../usr/bin/x86_64-linux-gcc -std=gnu99 -o conftest -D_LARGEFILE_SOURCE
   -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64   -Os  -static -Wall -D_LARGEFILE_SOURCE
    -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -DG_DISABLE_SINGLE_INCLUDES  -static
    conftest.c -ltiff34 -ljpeg -lz -lm  >&5
.../host/opt/ext-toolchain/bin/../lib/gcc/x86_64-buildroot-linux-uclibc/4.8.4/../../../../
    x86_64-buildroot-linux-uclibc/bin/ld: cannot find -ltiff34
.../host/opt/ext-toolchain/bin/../lib/gcc/x86_64-buildroot-linux-uclibc/4.8.4/../../../../
   x86_64-buildroot-linux-uclibc/bin/ld: cannot find -ljpeg
collect2: error: ld returned 1 exit status
configure:18177: $? = 1
configure: failed program was:
[...]
configure:18186: result: no
configure:18199: WARNING: *** TIFF loader will not be built (TIFF library not found) ***
configure:18210: error:
*** Checks for TIFF loader failed. You can build without it by passing
*** --without-libtiff to configure but some programs using GTK+ may
*** not work properly
```

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 auxiliary 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!

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.

Distribution

- 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`!

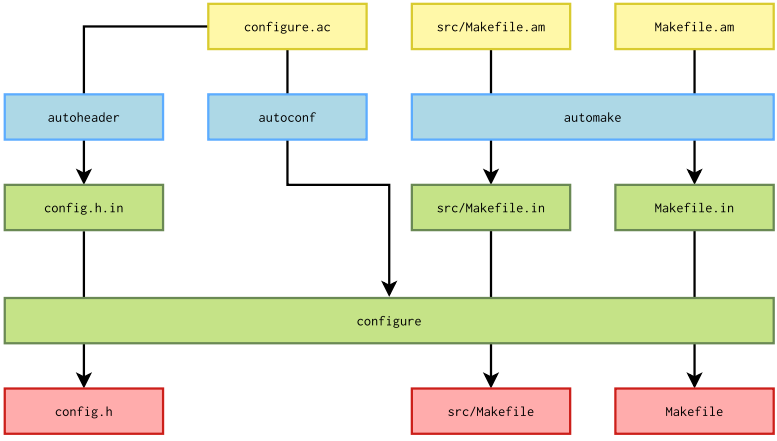
autoreconf example

```
$ find . -type f
./src/main.c
./Makefile.am
./configure.ac

$ autoreconf -i
configure.ac:4: installing './compile'
configure.ac:3: installing './install-sh'
configure.ac:3: installing './missing'
Makefile.am: installing './depcomp'

$ find . -type f
./install-sh
./src/main.c
./config.h.in
./configure
./missing
./depcomp
./aclocal.m4
./Makefile.am
./autom4te.cache/traces.0
./autom4te.cache/output.1
./autom4te.cache/output.0
./autom4te.cache/requests
./autom4te.cache/traces.1
./compile
./Makefile.in
./configure.ac
```

Overall organization



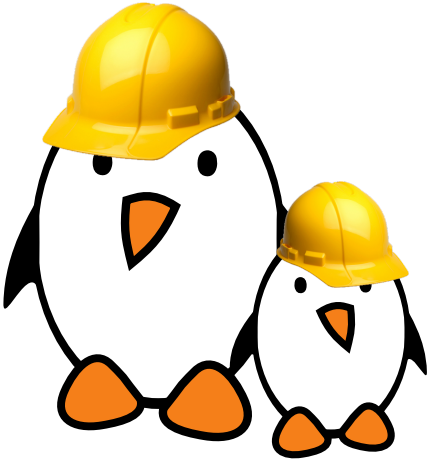
Written by the developer

Tools used by the developer generally through autoreconf

Generated by the developers, as the result of autoreconf

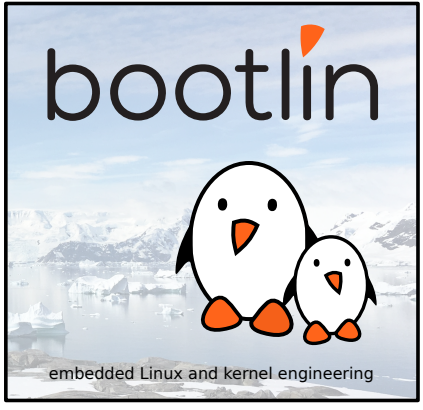
Generated by the person building the software, as the result of the configure script execution

Practical lab - Usage of existing *autotools* projects



- First build of an *autotools* package
- Out of tree build and cross-compilation
- Overriding cache variables
- Using *autoreconf*

Autotools basics



configure.ac language

- 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

```
$ cat configure.ac
AC_INIT([hello], [1.0])
AC_OUTPUT
$ ls
configure.ac
$ autoreconf -i
$ ls
autom4te.cache  configure  configure.ac
$ ./configure
configure: creating ./config.status
$ ls
autom4te.cache  config.log    config.status
configure       configure.ac
$ wc -l configure
2390 configure
```

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

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

configure.ac

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

```
$ ./configure
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
configure: creating ./config.status
```

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
```

AC_CONFIG_FILES example (1/2)

configure.ac

```
AC_INIT([hello], [1.0])
AC_PROG_CC
FOO=42
AC_SUBST([FOO])
AC_CONFIG_FILES([testfile])
AC_OUTPUT
```

testfile.in

```
abs_builddir = @abs_builddir@
CC = @CC@
FOO = @FOO@
```

AC_CONFIG_FILES example (2/2)

Executing `./configure`

```
/tmp/foo$ ./configure
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
configure: creating ./config.status
config.status: creating testfile
```

Generated `testfile`

```
abs_builddir = /tmp/foo
CC = gcc
FOO = 42
```

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

configure.ac

```
AC_INIT([hello], [1.0])
echo "The value of CC is $CC"
AC_PROG_CC
echo "The value of CC is now $CC"
FOO=42
AC_SUBST([FOO])
if test $FOO -eq 42 ; then
   echo "The value of FOO is correct!"
fi
AC_CONFIG_FILES([testfile])
AC_OUTPUT
```

Running `./configure`

```
The value of CC is
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
The value of CC is now gcc
The value of FOO is correct!
configure: creating ./config.status
config.status: creating testfile
```

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
- For these reasons, `Makefile.in` are typically not written manually, but generated by *automake* from a `Makefile.am` file

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
```

AM_INIT_AUTOMAKE

- `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 thing 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.

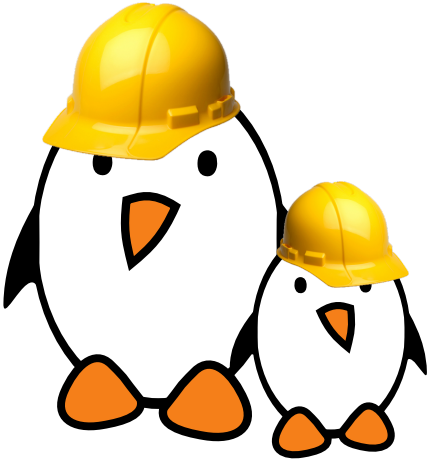
Example: building multiple programs

Makefile.am

```
bin_PROGRAMS = hello test
hello_SOURCES = main.c common.c common.h
test_SOURCES = test.c common.c common.h
```

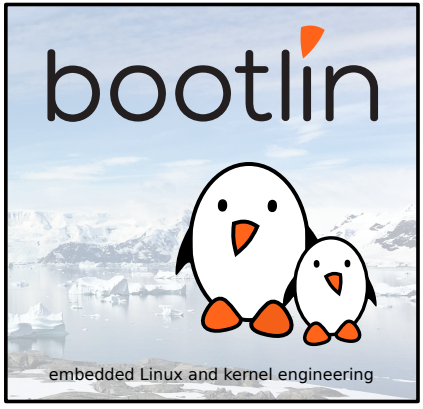
- Building two programs: `hello` and `test`
- Shared source files: `common.c` and `common.h`

Practical lab - Your first *autotools* project



- Your first `configure.ac`
- Adding and building a program
- Going further: `autoscan` and `make dist`

Autoconf advanced



Configuration header

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, vgagl) are installed */
/* #undef HAVE_LIBVGA */

/* Define to 1 if you have the <limits.h> header file. */
#define HAVE_LIMITS_H 1
```

AC_DEFINE

- 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])
```

Generated config.h

```
/* This is the foobar value */
#define FOOBAR 42
```

Checking for functions, headers, libraries, etc.

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

configure.ac

```
AC_CHECK_FUNCS([printf foobar])
echo "ac_cv_func_printf: ${ac_cv_func_printf}"
echo "ac_cv_func_foobar: ${ac_cv_func_foobar}"
AC_CONFIG_HEADER([config.h])
```

Execution of ./configure

```
$ ./configure
[...]
checking for printf... yes
checking for foobar... no
ac_cv_func_printf: yes
ac_cv_func_foobar: no
[...]
config.status: creating config.h
```

Generated config.h

```
[...]
/* Define to 1 if you have the `foobar' function. */
/* #undef HAVE_FOOBAR */

/* Define to 1 if you have the `printf' function. */
#define HAVE_PRINTF 1
[...]
```

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>`

AC_SEARCH_LIBS example

configure.ac

```
AC_SEARCH_LIBS(mvwaddstr, [ncurses cursesX curses])
```

Execution of ./configure

```
$ ./configure
[...]
checking for library containing mvwaddstr... -lncurses
[...]
$ grep ac_cv_search_mvwaddstr config.log
ac_cv_search_mvwaddstr=-lncurses
```

Compilation

```
$ make
[...]
gcc  -g -O2   -o hello main.o common.o  -lncurses
[...]
gcc  -g -O2   -o test test.o common.o  -lncurses
```

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

Custom tests

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

configure.ac

```
AC_LINK_IFELSE([AC_LANG_PROGRAM([#include <langinfo.h>],
        [char *codeset = nl_langinfo (CODESET);])],
    [glib_cv_langinfo_codeset=yes],
    [glib_cv_langinfo_codeset=no])
```

Variable in config.log

```
$ grep glib_cv_langinfo_codeset config.log
glib_cv_langinfo_codeset=yes
```

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

configure.ac

```
AC_MSG_CHECKING([for nl_langinfo])
AC_LINK_IFELSE([AC_LANG_PROGRAM([#include <langinfo.h>],
            [char *codeset = nl_langinfo (CODESET);])],
       [glib_cv_langinfo_codeset=yes],
       [glib_cv_langinfo_codeset=no])
AC_MSG_RESULT([$glib_cv_langinfo_codeset])
```

Execution of `./configure`

```
$ ./configure
[...]
checking for nl_langinfo... yes
[...]
```

External software and optional features

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

configure.ac

```
AC_ARG_ENABLE([test], AS_HELP_STRING([--enable-test], [Enable tests]),
      [echo "Action if given, val = ${enableval}"],
      [echo "Action if not given"])
echo "enable_test = ${enable_test}"
```

./configure tests

```
$ ./configure --help
[...]
Optional Features:
[...]
  --enable-test           Enable tests
$ ./configure
[...]
Action if not given
enable_test =
[...]
$ ./configure --enable-test
[...]
Action if given, val = yes
enable_test = yes
[...]
$ ./configure --disable-test
[...]
Action if given, val = no
enable_test = no
[...]
```

pkg-config

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.

PKG_CHECK_MODULES example

configure.ac

```
PKG_CHECK_MODULES(DBUS1,
     dbus-1 >= 1.2.14,
     [AC_DEFINE(HAVE_DBUS1, 1, [Define if dbus-1 is available]) have_dbus1=yes],
     have_dbus1=no)
```

Makefile.am

```
gdbus_serialization_CFLAGS = $(AM_CFLAGS) $(DBUS1_CFLAGS)
gdbus_serialization_LDADD = $(LDADD) $(DBUS1_LIBS)
```

Misc

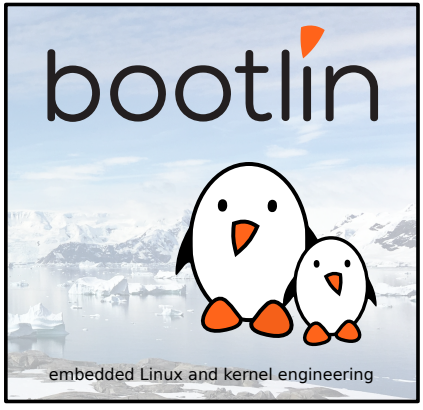
autoscan

- `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
  - https://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.*

Automake advanced



Subdirectories

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
  - *Recursive make considered harmful*, https://www.cse.iitb.ac.in/~soumen/teach/1999.2A.CS699/make.html
- **non-recursive make** is more and more commonly used in modern projects

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
```

Conditionals

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 x$debug = xtrue])
```

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 += assigment sign

Working example

```
bin_PROGRAMS = \
      bar \
      foobar

if DEBUG
bin_PROGRAMS += baz
endif
```

Conditional example

configure.ac

```
AM_CONDITIONAL(THREADS_POSIX, [test "$g_threads_impl" = "POSIX"])
AM_CONDITIONAL(THREADS_WIN32, [test "$g_threads_impl" = "WIN32"])
AM_CONDITIONAL(THREADS_NONE, [test "$g_threads_impl" = "NONE"])
```

Makefile.am

```
libglib_2_0_la_SOURCES =        \
        $(deprecated_sources)   \
        glib_probes.d           \
        garray.c                \
[...]

if THREADS_WIN32
libglib_2_0_la_SOURCES += gthread-win32.c
else
if THREADS_POSIX
libglib_2_0_la_SOURCES += gthread-posix.c
endif
endif
```

Shared libraries

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 library example (2/2)

```
$ ./configure
[...]
checking whether stripping libraries is possible... yes
checking if libtool supports shared libraries... yes
checking whether to build shared libraries... yes
checking whether to build static libraries... yes
[...]
$ make
[...]
$ make DESTDIR=/tmp/test install
[...]
$ find /tmp/test
/tmp/test/
/tmp/test/usr
/tmp/test/usr/local
/tmp/test/usr/local/include
/tmp/test/usr/local/include/myhello.h
/tmp/test/usr/local/bin
/tmp/test/usr/local/bin/hello
/tmp/test/usr/local/lib
/tmp/test/usr/local/lib/libmyhello.a
/tmp/test/usr/local/lib/libmyhello.la
/tmp/test/usr/local/lib/libmyhello.so
/tmp/test/usr/local/lib/libmyhello.so.0
/tmp/test/usr/local/lib/libmyhello.so.0.0.0
```

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`

Libtool versioning: example

Makefile.am

```
lib_LTLIBRARIES = libmyhello.la
libmyhello_la_SOURCES = lib/core.c
libmyhello_la_LDFLAGS = -version-info 3:4:2
```

Installation

```
$ make DESTDIR=/tmp/p install
[...]
$ ls -l /tmp/p/usr/local/lib
-rw-r--r-- 1 thomas thomas  6224 mai   20 15:28 libmyhello.a
-rwxr-xr-x 1 thomas thomas   963 mai   20 15:28 libmyhello.la
lrwxrwxrwx 1 thomas thomas    19 mai   20 15:28 libmyhello.so -> libmyhello.so.1.2.4
lrwxrwxrwx 1 thomas thomas    19 mai   20 15:28 libmyhello.so.1 -> libmyhello.so.1.2.4
-rwxr-xr-x 1 thomas thomas 10608 mai   20 15:28 libmyhello.so.1.2.4
```

Misc

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.la

module_test_LDADD = $(top_builddir)/gmodule/libgmodule-2.0.la $(LDADD)
module_test_LDFLAGS = $(G_MODULE_LDFLAGS)
slice_threadinit_LDADD = $(top_builddir)/gthread/libgthread-2.0.la $(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
```

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
[...]
```

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 += pcrecpparg.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 individiual 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

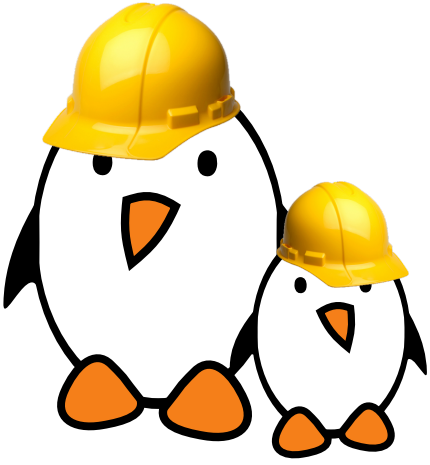
- The *auxiliary files* generated by *autotools* such as `compile`, `config.guess`, `config.sub`, `depcomp`, etc. are by default in the main directory of the source tree.
- This clutters the main directory with lots of files, which may not be very pleasant.
- `AC_CONFIG_AUX_DIR` allows to customize where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

configure.ac

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

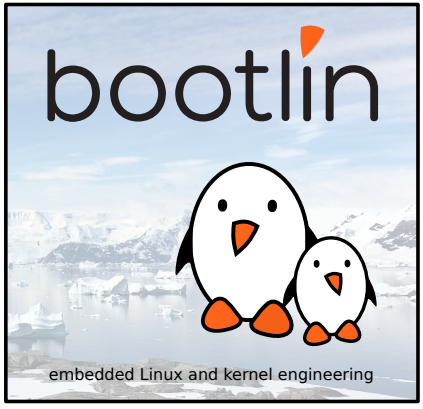
- One condition: it must be placed before the calls to `AM_INIT_AUTOMAKE` and `LT_INIT`

Practical lab - More advanced *autotools* usage



- Use `AC_ARG_ENABLE` and `config.h`
- Implement a shared library
- Switch to multiple directories
- Make the compilation of programs conditional
- Use `pkg-config`

Autotools references

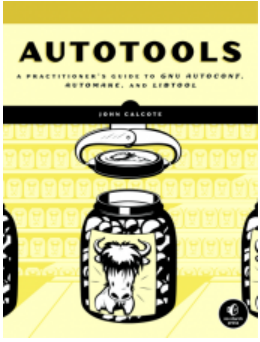


Existing code

- Lots of open-source projects are using the *autotools*
- They provide a lot of examples on how to configure and build things using the *autotools*
- However, make sure to have a critical eye when reading existing *autotools* code
  - For a lot of developers, the build system part is not their primary knowledge and interest
  - Lots of projects use deprecated constructs or truely horrible solutions
  - Don't copy/paste without thinking!

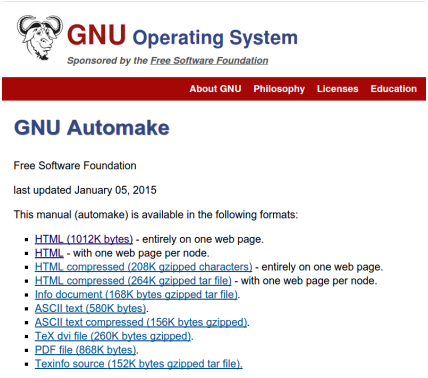
Book: *Autotools, a practitioner's guide*

- **Autotools, A Practitioner's Guide to GNU Autoconf, Automake, and Libtool**
- John Calcote
- No Starch Press
- https://www.nostarch.com/autotools.htm
- Excellent book.



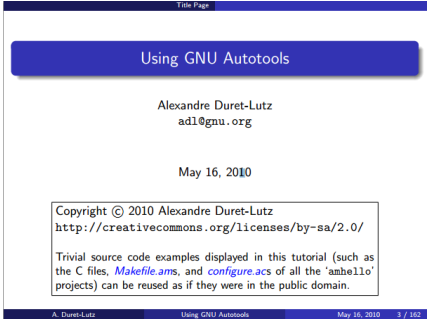
Official documentation

- The official reference documentation from GNU is also very good, once you have a good understanding of the basics.
- Autoconf
  https://www.gnu.org/software/autoconf/manual/
- Automake
  https://www.gnu.org/software/automake/manual/
- Libtool
  https://www.gnu.org/software/libtool/manual/



Tutorials

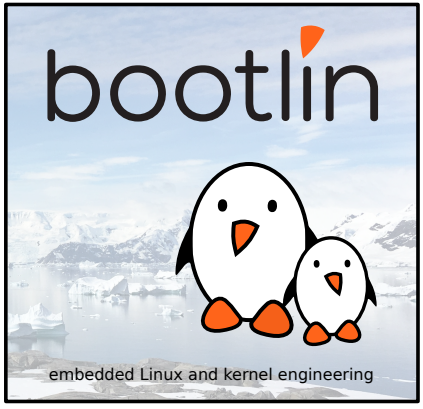
- **Autotools tutorial**, Alexandre Duret-Lutz, https://www.lrde.epita.fr/~adl/autotools.html
- **Autotools Mythbuster**, Diego Elio "Flameeyes" Pettenò, https://autotools.io/
- **Introduction to the Autotools**, David Wheeler, including a video, https://www.dwheeler.com/autotools/



Use up to date materials

- Be careful to use up-to-date material
  - For example, the well-known book *GNU Autoconf, Automake and Libtool"* by Gary Vaughan et al., published originally in 2000 is completely out of date
  - Even though *autotools* are old, they have evolved quite significantly in recent times!

Last slides



Last slide

Thank you!

And may the Source be with you

Rights to copy

Document sources: https://github.com/bootlin/training-materials/