GNU Autotools

- Autotools is for *developers* – end users does not need it
- Consists of 3 components:
  - Autoconf: Writes Makefile based on tests/choices from `configure` script
  - Automake: Make template `Makefile.in` based on developer specs in `Makefile.am`
  - Libtool: Compiles and links static/shared libraries and programs in a platform independent way.

Developer vs. End-User

- You, as a developer, give input to Autotools
- Autotools process this input, and write POSIX shell scripts
- The end users *uses* these scripts configure the sources for his or her system.

What the end user needs to know:

```
./configure
make
make install
```

Pros and Cons

- Platform independent
- Well tested (GNU tools has used Autotools for 5+ years)
- Transparent to the end user
- Easy to maintain
- De-facto standard in the Unix world
- Very flexible and configurable
- Steep learning curve (this talk will try to remedy that)

Misconceptions:

- Hard to use (simple input)
- Burdens the end-user (only needs POSIX shell, compiler, etc.)
- Incomprehensible `Makefile` (no need to understand these)
- Inflexible (can do as much as in hand-crafted, and more)

Without further ado ...

... our first simple example

```
simple> ls
Makefile.am  Simple.cxx  Simple.h bootstrap configure.ac main.cxx
```

Set package info

Set package type (foreign, gnu)

Check for C++ compiler

Make Libtool linker

Output files to write

```
at-simple> cat configure.ac
AC_INIT([Simple Autotools Example],
        [0.1], [cholm@nbi.dk],
        at-simple)
AM_INIT_AUTOMAKE([foreign])
AC_PROG_CXX
AC_PROG_LIBTOOL
AC_CONFIG_FILES(Makefile)
AC_OUTPUT
```

Write out configuration to output files and setup sources.

Specifying targets

Building libraries and programs, and linking them

```
simple> ls
Makefile.am  Simple.cxx  Simple.h bootstrap configure.ac main.cxx
```

Build shared/static library `libSimple.{a,so}`

Install in `<prefix>/lib`

Install in `<prefix>/bin`

Install in `<prefix>/include`

Sources of `libSimple`

Sources of `simple`

```
qt-simple> cat Makefile.am
lib_LTLIBRARIES          =
libSimple.la
bin_PROGRAMS             = simple
include_HEADERS          = Simple.h
libSimple_la_SOURCES     = Simple.cxx
simple_SOURCES           = main.cxx
simple_LDADD             = libSimple.la
```

Link `simple` against `libSimple`

Build program `simple`

Bootstrapping the sources

Initial setup of sources (done *once* by *developer*)

```
simple> ls
Makefile.am  Simple.cxx  Simple.h bootstrap configure.ac main.cxx
```

Adds Libtool script `ltmain.sh`, `config.guess`, `config.sub`

Puts needed macros in `aclocal.m4`

Creates template `Makefile.in`, adds scripts `install-sh`, `mkinstalldirs`, `missing`

Creates `configure` script

```
at-simple> cat bootstrap
#!/bin/sh
libtoolize --automake
aclocal
automake --add-missing --foreign
autoconf
```

Developer runs this script:

```
./bootstrap
```

Configuring the sources

Setup the sources to compile on system (done by end-user)

```
simple> ls -F
Makefile.am  aclocal.m4       config.sub@   ltmain.sh@
Makefile.in  autom4te.cache/  configure      main.cxx
Simple.cxx   bootstrap*       configure.ac  missing@
Simple.h     config.guess     install-sh@   mkinstalldirs@
```

```
at-simple> ./configure
...
configure: creating ./config.status
config.status: creating Makefile
```

Now ready to build program and library

```
at-simple> make
at-simple> make install
```

Distributing the sources

Make a gzipped tar-ball for the end-user to download – done by the *developer* fairly frequently

```
at-simple> make dist
...
at-simple> ls *.tar.gz
at-simple-0.1.tar.gz
```

This tar-ball contains all that the end-user needs.

Optionally, one can check the sanity of the distribution tar-ball:

```
at-simple> make distcheck
...
==========================================
at-simple-0.1.tar.gz is ready for distribution
==========================================
```

ROOT based Package

A package with some shared libraries and a program using ROOT

```
at-root
|-- Makefile.am
|-- acinclude.m4
|-- bootstrap
|-- configure.ac
|-- data
|    |-- Data.cxx
|    |-- Data.h
|    `-- Makefile.am
`-- dataio
     |-- DataReader.cxx
     |-- DataReader.h
     |-- DataWriter.cxx
     |-- DataWriter.h
     |-- LinkDef.h
     |-- Makefile.am
     `-- dataio.cxx
```

```
SUBDIRS          = data dataio
```

Contains macro `ROOT_PATH` from `root.m4` in ROOT distribution

```
AC_INIT([ROOT Autotools Example], [0.1],
        [cholm@nbi.dk], at-root)
AM_INIT_AUTOMAKE
AC_PROG_LIBTOOL
AC_PROG_CXX
ROOT_PATH(4.00/00,,
          [AC_MSG_ERROR([I Need ROOT])])
AC_CONFIG_FILES([Makefile data/Makefile
                 dataio/Makefile])
AC_OUTPUT
```

Checks for ROOT installation, and defines variables accordingly. Fails if ROOT isn't found or earlier version

Generating Dictionaries

Clean up

Extra preprocessor flags

From ROOT_PATH macro

```
at-root> cat data/Makefile.am
AM_CPPFLAGS                    = -I$(ROOTINCDIR)
CLEANFILES                     = DataDict.cxx DataDict.h
lib_LTLIBRARIES                = libData.la
pkginclude_HEADERS             = Data.h
libData_la_SOURCES             = Data.cxx
nodist_libData_la_SOURCES = DataDict.cxx
DataDict.cxx DataDict.h:Data.h
        $(ROOTCINT) -f DataDict.cxx -c $(AM_CPPFLAGS) $<
```

Install in
<prefix>/lib/<package>

Do not distribute (generated source)

From ROOT_PATH macro

Rule to make dictionary

Linking (external) libraries

Do not install this

To find `Data.h` header

```
at-root> cat dataio/Makefile.am
AM_CPPFLAGS                  = -I$(ROOTINCDIR) -I$(top_srcdir)/data
CLEANFILES                   = DataIODict.cxx DataIODict.h
lib_LTLIBRARIES              = libDataIO.la
bin_PROGRAMS                 = dataio
pkginclude_HEADERS           = DataWriter.h DataReader.h
noinst_HEADERS               = LinkDef.h
libDataIO_la_SOURCES         = DataWriter.cxx DataReader.cxx
nodist_libDataIO_la_SOURCES = DataIODict.cxx
libDataIO_la_LIBADD          = $(top_builddir)/data/libData.la
dataio_SOURCES               = dataio.cxx
dataio_LDADD                 = libDataIO.la $(ROOTLIBS)
dataio_LDFLAGS               = -R $(ROOTLIBDIR) -L$(ROOTLIBDIR)
DataIODict.cxx DataIODict.h:$(pkginclude_HEADERS) $(noinst_HEADERS)
        $(ROOTCINT) -f DataIODict.cxx -c $(AM_CPPFLAGS) $^
```

Incremental linkage (`libDataIO` needs `libData`)

Where to find `libData`

Runtime load path

More on Autotools

- Check for system ‘features’ - not specific OS, etc. Ease portability.
- Check for external libraries, headers, etc.
- Mix Fortran 77, C++, C, etc. transparently (Libtool does The Right Thing™).
- One-Makefile builds (i.e., no recursive `makes`, which are sometimes considered ‘evil’)
- Test-suites
- Sub-packages

```
AC_C_BIGENDIAN
AC_CHECK_TYPE(long long)
AC_FUNC_STRTOD
```

```
AC_LANG_PUSH([Fortran 77])
AC_CHECK_LIB(Pythia, PYINIT)
AC_LANG_POP([Fortran 77])
```

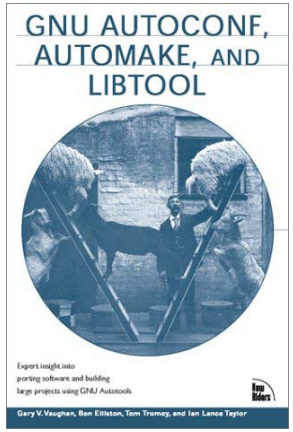
```
libFoo_la_SOURCES = Foo.cxx \
                    Bar.F Baz.c
```

```
# top Makefile.am
pkglib_LTLIBRARIES      =
...
include STEER/Makefile

# STEER/Makefile
pkglib_LTLIBRARIES += libSTEER.la
libSTEER_la_SOURCES = \
        STEER/AliRunLoader.cxx \
        ...
```

References

- Autotools web-pages (including reference manuals)
  - Autoconf: http://www.gnu.org/software/autoconf/
  - Automake: http://www.gnu.org/software/automake/
  - Libtool: http://www.gnu.org/software/libtool/
  - Info pages on system: `info <autoconf|automake|libtool>`
  - The ‘Goat Book’: “GNU Autoconf, Automake and Libtool”, freely available on-line at http://sources.redhat.com/autobook/
  - My homepage (http://cern.ch/cholm/) has a lot of packages that uses Autotools. Feel free to download and use as you like.



What creates what?