Build Tools

https://projects.coin-or.org/BuildTools

Stefan Vigerske
COIN-OR Workshop · October 15, 2018
BuildTools 0.5 – 0.8
COIN-OR BuildTools project

- a set of scripts to utilize Autotools (GNU Build System) to simplify building of COIN-OR projects on various platforms
- initial development (2006–2011) by Andreas Wächter
- using Autoconf 2.59 (2003), Automake 1.9.6 (2005), and Libtool 1.5.22 (2005)
- replaced hand-crafted Makefiles
- used by CoinUtils, Osi, Clp, Cgl, Cbc, Symphony, Ipopt, Bonmin, Couenne, OS, ...

Changeset 1 for trunk

- Timestamp: May 5, 2008 1:07:35 AM (12 years ago)
- Author: andreasw
- Message: first version
- Location: trunk
- Files: 4 added
  - Makemain.inc
  - coin.m4
  - run_autotools

Changeset 107

- Timestamp: Sep 18, 2006 7:19:10 PM (12 years ago)
- Author: andreasw
- Message: Creating new release 0.5.1 from stable/0.5
- File: 1 copied
  - releases/0.5.1 (copied from stable/0.5)
COIN-OR BuildTools project

- a set of scripts to utilize **Autotools (GNU Build System)** to simplify building of COIN-OR projects on various platforms
- initial development (2006–2011) by **Andreas Wächter**
- using Autoconf 2.59 (2003), Automake 1.9.6 (2005), and Libtool 1.5.22 (2005)
- replaced hand-crafted Makefiles
- used by CoinUtils, Osi, Clp, Cgl, Cbc, Symphony, Ipopt, Bonmin, Couenne, OS, ...
- further development by **Lou Hafer, Ted Ralphs, me**

Apr 30, 2006 – Oct 13, 2018
Autoconf generates a configure script, which checks for:

- compilers, e.g., for C, C++, Fortran
- compiler capabilities, e.g., OpenMP
- header files, e.g., unistd.h, cstdint
- libraries, e.g., libz, libgmp, libblas
- (library) functions, e.g., snprintf, _snprintf
- types, e.g., intptr_t, long long
- Fortran naming convention

and collects the results in config.status

- detected dependencies (HAVE_LIBZ, HAVE_SNPRINTF)
- compiler and linker flags (CFLAGS, LDFLAGS, LIBS)
- which files to process (Makefile.in, config.h.in)

and creates the libtool helper script

- compilation and linking of single object files, libraries, executables, incl. dependency tracking (.lo, .la)
Autotools machinery: automake

Automake reads a *Makefile.am*, which specifies

- libraries and executables to be build
- source files to be compiled for each lib/exe
- header and data files to be installed
- additional build rules in Makefile-syntax

and generates a GNU Makefile template (*Makefile.in*)

- rules to build all libraries and executables (by use of *libtool*)
- rules to install and uninstall headers, binaries, data

For COIN-OR BuildTools based projects, `configure.ac` and `Makefile.am` are handwritten. Then the project manager runs a script that runs:

```bash
cat /usr/share/aclocal/libtool.m4 BuildTools/coin.m4 \   > acinclude.m4
aclocal   # configure.ac + acinclude.m4 → aclocal.m4
autoheader # configure.ac + acinclude.m4 → config.h.in
automake   # Makefile.am + config.h.in → Makefile.in
autoconf    # configure.ac + aclocal.m4 → configure
```

`config.h.in`, `Makefile.in`, `configure`, and some helper scripts are redistributed.
<table>
<thead>
<tr>
<th>File</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coin-functions</td>
<td>21K</td>
<td>utility functions for other scripts</td>
</tr>
<tr>
<td>coin.m4</td>
<td>152K</td>
<td>common macros to use in configure.ac</td>
</tr>
<tr>
<td>commit_new_release</td>
<td>7.5K</td>
<td>script to commit a new release</td>
</tr>
<tr>
<td>commit_new_stable</td>
<td>7.3K</td>
<td>script to commit a new stable branch</td>
</tr>
<tr>
<td>compile_f2c/</td>
<td>4.0K</td>
<td>script to use f2c as Fortran 77 compiler</td>
</tr>
<tr>
<td>config.guess</td>
<td>44K</td>
<td>configure helper script (comes with autoconf)</td>
</tr>
<tr>
<td>config.sub</td>
<td>32K</td>
<td>configure helper script (comes with autoconf)</td>
</tr>
<tr>
<td>depcomp</td>
<td>16K</td>
<td>build helper script (comes with autoconf)</td>
</tr>
<tr>
<td>get.dependencies.sh</td>
<td>18K</td>
<td>build helper script (should not be here)</td>
</tr>
<tr>
<td>headers/</td>
<td>4.0K</td>
<td>config.h files for non-autotools-builds</td>
</tr>
<tr>
<td>install-sh</td>
<td>9.1K</td>
<td>install helper script (comes with autoconf)</td>
</tr>
<tr>
<td>ltmain.sh</td>
<td>192K</td>
<td>build helper script (comes with autoconf)</td>
</tr>
<tr>
<td>Makemain.inc</td>
<td>4.9K</td>
<td>common Makefile rules to use in Makefile.am</td>
</tr>
<tr>
<td>missing</td>
<td>11K</td>
<td>build helper script (comes with autoconf)</td>
</tr>
<tr>
<td>MSVisualStudio/</td>
<td>4.0K</td>
<td>common MSVS project file inputs</td>
</tr>
<tr>
<td>prepare_new_release</td>
<td>26K</td>
<td>script to create a new release</td>
</tr>
<tr>
<td>prepare_new_stable</td>
<td>22K</td>
<td>script to create a new stable branch</td>
</tr>
<tr>
<td>run_autotools</td>
<td>14K</td>
<td>script to rebuild configure, Makefile.in, etc.</td>
</tr>
<tr>
<td>setExternals</td>
<td>5.0K</td>
<td>script to update svn:externals of projects</td>
</tr>
<tr>
<td>share/</td>
<td>4.0K</td>
<td>template config.site file</td>
</tr>
</tbody>
</table>
COIN-OR project structure: base directory

E.g., `svn co https://projects.coin-or.org/svn/Clp/trunk`:

<table>
<thead>
<tr>
<th>Directory</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clp/</td>
<td></td>
</tr>
<tr>
<td>BuildTools/</td>
<td><a href="https://projects.coin-or.org/svn/BuildTools/stable/0.8">https://projects.coin-or.org/svn/BuildTools/stable/0.8</a></td>
</tr>
<tr>
<td>CoinUtils/</td>
<td><a href="https://projects.coin-or.org/svn/CoinUtils/trunk/CoinUtils">https://projects.coin-or.org/svn/CoinUtils/trunk/CoinUtils</a></td>
</tr>
<tr>
<td>Osi/</td>
<td><a href="https://projects.coin-or.org/svn/Osi/trunk/Osi">https://projects.coin-or.org/svn/Osi/trunk/Osi</a></td>
</tr>
<tr>
<td>Data/Netlib/</td>
<td><a href="https://projects.coin-or.org/svn/Data/Netlib/stable/1.2">https://projects.coin-or.org/svn/Data/Netlib/stable/1.2</a></td>
</tr>
<tr>
<td>Data/Sample/</td>
<td><a href="https://projects.coin-or.org/svn/Data/Sample/stable/1.2">https://projects.coin-or.org/svn/Data/Sample/stable/1.2</a></td>
</tr>
<tr>
<td>ThirdParty/ASL</td>
<td><a href="https://projects.coin-or.org/svn/BuildTools/ThirdParty/ASL/stable/1.3">https://projects.coin-or.org/svn/BuildTools/ThirdParty/ASL/stable/1.3</a></td>
</tr>
<tr>
<td>ThirdParty/Blas</td>
<td><a href="https://projects.coin-or.org/svn/BuildTools/ThirdParty/Blas/stable/1.4">https://projects.coin-or.org/svn/BuildTools/ThirdParty/Blas/stable/1.4</a></td>
</tr>
<tr>
<td>ThirdParty/Lapack</td>
<td><a href="https://projects.coin-or.org/svn/BuildTools/ThirdParty/Lapack/stable/1.5">https://projects.coin-or.org/svn/BuildTools/ThirdParty/Lapack/stable/1.5</a></td>
</tr>
<tr>
<td>ThirdParty/Metis</td>
<td><a href="https://projects.coin-or.org/svn/BuildTools/ThirdParty/Metis/stable/1.3">https://projects.coin-or.org/svn/BuildTools/ThirdParty/Metis/stable/1.3</a></td>
</tr>
<tr>
<td>ThirdParty/Mumps</td>
<td><a href="https://projects.coin-or.org/svn/BuildTools/ThirdParty/Mumps/stable/1.3">https://projects.coin-or.org/svn/BuildTools/ThirdParty/Mumps/stable/1.3</a></td>
</tr>
<tr>
<td>ThirdParty/Glpk</td>
<td><a href="https://projects.coin-or.org/svn/BuildTools/ThirdParty/Glpk/stable/1.10">https://projects.coin-or.org/svn/BuildTools/ThirdParty/Glpk/stable/1.10</a></td>
</tr>
<tr>
<td>configure</td>
<td></td>
</tr>
<tr>
<td>configure.ac</td>
<td></td>
</tr>
<tr>
<td>Makefile.am</td>
<td></td>
</tr>
<tr>
<td>Makefile.in</td>
<td></td>
</tr>
<tr>
<td>Dependencies</td>
<td></td>
</tr>
<tr>
<td>INSTALL</td>
<td></td>
</tr>
<tr>
<td>LICENSE</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Base directory configure.ac (here: Clp)

AC_PREREQ(2.59)
AC_INIT([Clp],[trunk],[coin-clp@lists.coin-or.org])
AC_COPYRIGHT([Copyright 2006 International Business Machines and others. ...])
AC_REVISION(0.9)
AC_CONFIG_SRCDIR(configure.ac)  # file to check whether package is present
AC_PREFIX_DEFAULT(['pwd'])      # change default for --prefix
AC_COIN_CREATE_LIBTOOL         # create libtool, reused in subdirectories

AC_COIN_MAIN_PACKAGEDIR(ASL,   ThirdParty, [solvers/asl.h])
AC_COIN_MAIN_PACKAGEDIR(Blas,  ThirdParty, [daxpy.f])
AC_COIN_MAIN_PACKAGEDIR(Lapack,ThirdParty, [LAPACK/SRC/dlarf.f])
AC_COIN_MAIN_PACKAGEDIR(Metis, ThirdParty, [metis-4.0/Lib/balance.c])
AC_COIN_MAIN_PACKAGEDIR(Mumps, ThirdParty, [MUMPS/src/dmumps_part1.F])
AC_COIN_MAIN_PACKAGEDIR(Glpk,  ThirdParty, [glpk/src/glplpx01.c])
AC_COIN_MAIN_PACKAGEDIR(Sample, Data)
AC_COIN_MAIN_PACKAGEDIR(Netlib, Data)
AC_COIN_MAIN_PACKAGEDIR(CoinUtils)
AC_COIN_MAIN_PACKAGEDIR(Osi)
AC_COIN_MAIN_PACKAGEDIR(Clp)

AC_COIN_DOXYGEN(CoinUtils Osi)

AC_CONFIG_FILES([Makefile])      # file to be created by config.status
AC_CONFIG_FILES([doxydoc/doxygen.conf])  # file to be created by config.status

AC_COIN_FINALIZE
AUTOMAKE_OPTIONS = foreign   # disable GNU standard checks
EXTRA_DIST = doxydoc/doxygen.conf
SUBDIRS = $(subdirs)   # $(subdirs) is set by configure to list of subdirectories to recurse into

test: all
   cd Clp; $(MAKE) test
unitTest: test
tests:
   for dir in $(subdirs); do if test -r $$dir/test/Makefile; then (cd $$dir; $(MAKE) test) fi; done
unitTests: tests
doxydoc:
   for dir in $(subdirs); do ... done;
clean-doxydoc:
   ( cd doxydoc ; rm -rf html *.log *.tag )
install-doxydoc: doxydoc
   ...
uninstall-doxydoc:
   ...
clean-local: clean-doxydoc
# install-data-local: install-doxydoc
uninstall-local: uninstall-doxydoc

.PHONY: test unitTest tests unitTests doxydoc

DISTCLEANFILES = coin_subdirs.txt   # extra files to be cleaned by make distclean
include BuildTools/Makemain.inc
COIN-OR project structure: project directory

E.g., svn co https://projects.coin-or.org/svn/Clp/trunk:

Clp/AUTHORS
Clp/clp.pc.in
Clp/clp-uninstalled.pc.in
Clp/configure
Clp/configure.ac
Clp/doxydoc/
Clp/examples/
Clp/LICENSE
Clp/Makefile.am
Clp/Makefile.in
Clp/MSVisualStudio/
Clp/README
Clp/src/
Clp/test/
  configure
  configure.ac
Makefile.am
Makefile.in
BuildTools/
...
Project directory configure.ac (here: Clp/configure.ac)

AC_PREREQ(2.59)
AC_INIT([Clp],[trunk],[clp@list.coin-or.org])
AC_COPYRIGHT([Copyright 2006 International Business Machines and others. ...])
AC_CONFIG_SRCDIR(src/ClpSimplex.cpp)
AC_CANONICAL_BUILD # Get the system type
AC_COIN_PROJECTDIR_INIT(Clp) # Initialize some COIN-OR specific vars, e.g., version number
AC_COIN_DEBUG_COMPILE(Clp) # Check if user wants to produce debugging code
AC_COIN_PROG_CXX # Check for C++ compiler
AC_COIN_INIT_AUTO_TOOLS # Initialize automake and libtool the COIN-OR way

AC_COIN_CHECK_PACKAGE(CoinUtils, [coinutils], [ClpLib])
AC_COIN_CHECK_PACKAGE(Osi, [osi], [OsiClpLib])
AC_COIN_CHECK_PACKAGE(OsiTests, [osi-unittests])
AC_COIN_CHECK_PACKAGE(Sample, [coindatasample])
AC_COIN_CHECK_PACKAGE(Netlib, [coindatanetlib])
AC_COIN_CHECK_USER_LIBRARY(AMD, AMD, [amd.h], [amd_defaults], [ClpLib]) # AMD from UFL
coin_save_LIBS="$LIBS"
LIBS="$LIBS $AMD_LIBS"
AC_COIN_CHECK_USER_LIBRARY(CHOLMOD, CHOLMOD, [cholmod.h], [cholmod_start], [ClpLib]) # CHOLMOD from UFL
LIBS="$coin_save_LIBS"
AC_COIN_CHECK_PACKAGE(ASL, [coinasl], [ClpLib])
AC_COIN_CHECK_GNU_READLINE(ClpLib)

AC_COIN_CHECK_CXX_CHEADER(math)
AC_COIN_CHECK_CXX_CHEADER(float)
AC_COIN_CHECK_CXX_CHEADER(ieeefp)
...

11/30
AC_COIN_VPATH_LINK(examples/hello.mps) # In case this is a VPATH configuration (build directory different
from source directory), we need to make sure that the
input files for the examples are available in the build directory.
AC_COIN_VPATH_LINK(examples/input.130)
AC_COIN_VPATH_LINK(examples/g.tiny)
AC_COIN_VPATH_LINK(examples/gparm.tiny)

AC_COIN_DOXYGEN(CoinUtils Osi)

# Files to be created by config.status (except for the config header files)
AC_CONFIG_FILES([Makefile
  examples/Makefile
  src/Makefile
  src/OsiClp/Makefile
  test/Makefile
  clp.pc
  clp-uninstalled.pc])
AC_CONFIG_FILES([doxydoc/doxygen.conf])
AC_CONFIG_HEADER([src/config.h src/config_clp.h])
AC_COIN_FINALIZE
libLTLIBRARIES = `libClp.la libClpSolver.la` # libs to be compiled, to be installed in 'lib' dir

libClp_la_SOURCES = ClpConfig.h ClpCholeskyBase.cpp ClpCholeskyBase.hpp ... ClpPEDualRowSteepest.cpp
if COIN_HAS_AMD
libClp_la_SOURCES += ClpCholeskyUfl.cpp ClpCholeskyUfl.hpp # if AMD available, compile ClpCholeskyUfl.cpp
endif
libClpSolver_la_SOURCES = ClpSolver.cpp CbcOrClpParam.cpp CbcOrClpParam.hpp ... unitTest.cpp

libClp_la_LIBADD = `$(CLPLIB_LIBS)`
libClpSolver_la_LIBADD = `$(CLPLIB_LIBS)` libClp.la

bin_PROGRAMS = clp # executable to be compiled, to be installed in 'bin'
clp_SOURCES = ClpMain.cpp
clp_LDADD = libClpSolver.la libClp.la `$(CLPLIB_LIBS)` `$(ASL_LIBS)`
clp_DEPENDENCIES = libClpSolver.la libClp.la `$(CLPLIB_DEPENDENCIES)` # same as _LDADD, but without -L, -l

AM_CPPFLAGS = `$(CLPLIB_CFLAGS)` -DCOIN_HAS_CLP
if COIN_HAS_AMD
AM_CPPFLAGS += -I`$(CYGPATH_W)` `$(AMDINCDIR)`
endif

includecoindir = `$(includedir)/coin` # directory where to install header files
includecoin_HEADERS = Clp_C_Interface.h ClpCholeskyBase.hpp ... ClpPEDualRowSteepest.hpp

install-exec-local:
   `$(install_sh_DATA)` config_clp.h `$(DESTDIR)`$(includecoindir)/ClpConfig.h
uninstall-local:
   rm -f `$(DESTDIR)`$(includecoindir)/ClpConfig.h
Building COIN-OR project: running configure

> ./configure -C
configure: creating cache config.cache
checking build system type... x86_64-unknown-linux-gnu
checking whether we want to compile in debug mode... no
checking for gcc... gcc
...
checking for sys/types.h... yes
checking for sys/stat.h... yes
checking for stdlib.h... yes
checking for string.h... yes
...
checking whether source of project ASL is available and should be compiled... no, source file ./ThirdParty/ASL/solvers/asl.h not available
checking whether source of project Blas is available and should be compiled... no, source file ./ThirdParty/Blas/daxpy.f not available
checking whether source of project Lapack is available and should be compiled... no, source file ./ThirdParty/Lapack/LAPACK/SRC/dlarf.f not available
checking whether source of project Metis is available and should be compiled... no, source file ./ThirdParty/Metis/metis-4.0/Lib/balance.c not available
checking whether source of project Mumps is available and should be compiled... no, source file ./ThirdParty/Mumps/MUMPS/src/dmumps_part1.F not available
checking whether source of project Glpk is available and should be compiled... no, source file ./ThirdParty/Glpk/glpk/src/glplpx01.c not available
checking whether source of project Sample is available and should be compiled... yes, source in Data/Sample
checking whether source of project Netlib is available and should be compiled... yes, source in Data/Netlib
checking whether source of project CoinUtils is available and should be compiled... yes, source in CoinUtils
checking whether source of project Osi is available and should be compiled... yes, source in Osi
checking whether source of project Clp is available and should be compiled... yes, source in Clp
...
updating cache config.cache
configure: creating ./config.status
config.status: creating Makefile
config.status: creating doxydoc/doxygen.conf
config.status: executing depfiles commands
configure: configuring in Data/Sample
configure: running /bin/sh './configure' --prefix=/home/stefan/work/coin/Clp-trunk '-C' --cache-file=../../config.cache --srcdir=. conf
configure: loading cache ../..//config.cache
...
Building COIN-OR project: running configure (cont.)

... configure: configuring in Clp
configure: running /bin/sh './configure' --prefix=/home/stefan/work/coin/Clp-trunk '-C' --cache-file=../config.cache --srcdir=.
configure: loading cache ../config.cache
checking build system type... (cached) x86_64-unknown-linux-gnu
... configure: Using libtool script in directory ..
checking if library version is set... no
checking for pkg-config... (cached) pkg-config
checking pkg-config is at least version 0.16.0... yes
checking for COIN-OR package CoinUtils... yes: trunk
checking for COIN-OR package Osi... yes: trunk
checking for COIN-OR package OsiTests... yes: trunk
checking for COIN-OR package Sample... yes: 1.2
checking for COIN-OR package Netlib... yes: 1.2
checking if user provides library for AMD... no
checking if user provides library for CHOLMOD... no
checking for COIN-OR package Glpk... not given: Package 'coinglpk', required by 'virtual:world', not found
checking for COIN-OR package Mumps... not given: Package 'coinmumps', required by 'virtual:world', not found
checking for COIN-OR package ASL... not given: Package 'coinsasl', required by 'virtual:world', not found
...
configure: creating ./config.status
config.status: creating Makefile
config.status: creating examples/Makefile
config.status: creating src/Makefile
...
config.status: executing depfiles commands
configure: In case of trouble, first consult the troubleshooting page at https://projects.coin-or.org/BuildTools/wiki/user-troubleshooting
configure: Configuration of Clp successful
configure: In case of trouble, first consult the troubleshooting page at https://projects.coin-or.org/BuildTools/wiki/user-troubleshooting
configure: Main configuration of Clp successful
Building COIN-OR project: running make

> make
Making all in Data/Sample
make[1]: Entering directory 'Data/Sample'
make[1]: Nothing to be done for 'all'.
make[1]: Leaving directory 'Data/Sample'
Making all in CoinUtils
make[1]: Entering directory 'CoinUtils'
Making all in src
make[2]: Entering directory 'CoinUtils/src'
make all-am
make[3]: Entering directory 'CoinUtils/src'
if /bin/sh ../../libtool --tag=CXX --mode=compile g++ -DHAVE_CONFIG_H -I. -I‘echo .‘ -O3 -pipe -DNDEBUG -Wparentheses 
-DCOINUTILS_BUILD -MT CoinAlloc.lo -MD -MP -MF "../.deps/CoinAlloc.Tpo" -c -o CoinAlloc.lo CoinAlloc.cpp; 
then mv -f "../.deps/CoinAlloc.Tpo" "../.deps/CoinAlloc.Plo"; else rm -f "../.deps/CoinAlloc.Tpo"; exit 1; fi
mkdir .libs
-Wconversion -Wno-unknown-pragmas -Wno-long-long -DCOINUTILS_BUILD -MT CoinAlloc.lo -MD -MP -MF ../.deps/CoinAlloc.Tpo 
-c CoinAlloc.cpp -fPIC -DPIC -o .libs/CoinAlloc.o

... 
/bin/sh ../../libtool --tag=CXX --mode=link g++ -O3 -pipe ... -o libCoinUtils.la -rpath lib -no-undefined 
 CoinBuild.lo ... CoinWarmStartPrimalDual.lo -lbz2 -lz -llapack -lblas -lm
make -shared ... .libs/CoinAlloc.o .libs/CoinBuild.o ... .libs/CoinWarmStartPrimalDual.o -lbz2 -lz -llapack -lblas ... 
-Wl,-soname -Wl,libCoinUtils.so.0.0.0 -o .libs/libCoinUtils.so.0.0.0
(cd .libs & & rm -f libCoinUtils.so.0.0.0 & & ln -s libCoinUtils.so.0.0.0.0 libCoinUtils.so.0)
(cd .libs & & rm -f libCoinUtils.so & & ln -s libCoinUtils.so.0.0.0.0 libCoinUtils.so)
creating libCoinUtils.la
(cd .libs & & rm -f libCoinUtils.la & & ln -s ..../libCoinUtils.la libCoinUtils.la)
make[3]: Leaving directory 'CoinUtils/src'
... 
Making all in Clp
... 
g++ -DHAVE_CONFIG_H -I. -ICoinUtils/src ... -c ClpCholeskyBase.cpp -fPIC -DPIC -o .libs/ClpCholeskyBase.o
BuildTools trunk
Monolithic build of a project including all its dependencies from source isn’t preferred.

- Conflicts when installing in a common location, e.g., /usr/local.
- Better build and install one project at a time.
- Extra complexity due to possibility to build against both installed and not-yet-installed versions of a dependency.
Problems with current stable (0.8)

Monolithic build of a project including all its dependencies from source isn’t preferred.

- Conflicts when installing in a common location, e.g., /usr/local.
- Better build and install one project at a time.
- Extra complexity due to possibility to build against both installed and not-yet-installed versions of a dependency.

Autotools from 2005 is “slightly” outdated.

- Various BuildTools patches to libtool, etc., make it hard to upgrade.
- We miss out on Autotools improvements from the last decade.
Problems with current stable (0.8)

Monolithic build of a project including all its dependencies from source isn’t preferred.

- Conflicts when installing in a common location, e.g., /usr/local.
- Better build and install one project at a time.
- Extra complexity due to possibility to build against both installed and not-yet-installed versions of a dependency.

Autotools from 2005 is “slightly” outdated.

- Various BuildTools patches to libtool, etc., make it hard to upgrade.
- We miss out on Autotools improvements from the last decade.

Building on Windows with MS/Intel compilers requires a lot of “hacks”.

- libtool patches to adapt to different compiler and linker usage
- Slashes vs Backslashes, e.g., from coin.m4:
  ```
  sed -e 's/\([^/]\)*\(/$(CYGPATH_W) \
  | sed -e "s/\\\\\\\\\\|/\\|g"'\2.lib /g'
  ```
- Building DLLs is hardly possible at all.
Main changes and goals:

- Use current autotools versions and make future upgrades easier.
Main changes and goals:

- Use current autotools versions and make future upgrades easier.
- Elimination of base directories.
- Helper script to build and install dependencies.
- Rely on pkg-config to locate dependencies and identify compiler/linker flags.
- Clearer identification which compiler and linker flags are needed for which library or executable within a package.
Main changes and goals:

- Use current autotools versions and make future upgrades easier.
- Elimination of base directories.
- Helper script to build and install dependencies.
- Rely on pkg-config to locate dependencies and identify compiler/linker flags.
- Clearer identification which compiler and linker flags are needed for which library or executable within a package.
- Rely on helper-scripts (e.g., compile, ccc1) to support MS/Intel compilers on Windows. Avoid patching up libtool.
- Build DLLs on Windows.
New Autotools versions

BuildTools/trunk currently expects

- Autoconf 2.69 (2012, latest)
- Automake 1.15 (2015); latest would be 1.16.1 (2018)
- Libtool 2.4.6 (2015, latest)

Script `install_autotools.sh` downloads and installs these in $HOME/local2.
Usage: get.dependencies.sh <command> --option1 --option2

Commands:

**fetch**: Checkout source code for all dependencies
- **options**: --svn (checkout from SVN)
- --git (checkout from git)
- --skip='proj1 proj2' skip listed projects
- --no-third-party don’t download third party source (getter-scripts)

**build**: Configure, build, test (optional), and pre-install all projects
- **options**: --xxx=yyy (will be passed through to configure)
- --monolithic do ’old style’ monolithic build
- --parallel-jobs=n build in parallel with maximum ’n’ jobs
- --build-dir=/dir/to/build/in do a VPATH build (default: $PWD/build)
- --test run unit test of main project before install
- --test-all run unit tests of all projects before install
- --verbosity=i set verbosity level (1-4)
- --reconfigure re-run configure

**install**: Install all projects in location specified by prefix
- **options**: --prefix=/dir/to/install (where to install, default: $PWD/build)

**uninstall**: Uninstall all projects

General options:
- --debug: Turn on debugging output
**pkg-config / pkgconf**

- Checks whether certain packages (in required version) are available on system.
- Packages need to provide a `.pc` file, e.g., `coinutils.pc`:
  
  ```
  prefix=/path/to/CoinUtils/build
  exec_prefix=$prefix
  libdir=$exec_prefix/lib
  includedir=$prefix/include/coin-or
  ```

  **Name:** CoinUtils  
  **Description:** COIN-OR Utilities  
  **URL:** https://projects.coin-or.org/CoinUtils  
  **Version:** trunk  
  **Libs:** `-L$libdir -lCoinUtils`  
  **Libs.private:** `-L/path/to/CoinUtils/build/lib -lcoinglpk -llapack -lblas -lreadline -lbz2 -lz`  
  **Cflags:** `-I$includedir`  
  **Requires.private:** coinglpk lapack blas

- Typically reside in `/usr/lib/pkgconfig`, `/usr/lib64/pkgconfig`, etc.
pkg-config (.pc) files can be queried for compiler and linker flags:

> pkg-config --exists coinutils

> pkg-config --cflags coinutils
   -I/path/to/CoinUtils/build/include/coin-or
   -I/path/to/CoinUtils/build/include/coin-or/ThirdParty

> pkg-config --libs coinutils
   -L/path/to/CoinUtils/build/lib -lCoinUtils

> pkg-config --libs coinutils --static
   -L/path/to/CoinUtils/build/lib -lCoinUtils -L/path/to/CoinUtils/build/lib
   -lcoinglpk -lblas -lreadline -lbz2 -lz -L/path/to/CoinUtils/build/lib
   -lcoinglpk -llapack -lblas

> pkg-config --variable=datadir coindatasample.pc
   /path/to/CoinUtils/build/share/coin-or-sample
configure macro checking for pkg-config package

\texttt{AC_COIN_CHK_PKG([prim], [client packages], [.pc file name], [default action], [cmdopts])}

- determine availability of primary package prim
- assemble compiler flags, linker flags, and data directories
- augment client\_CFLAGS, client\_LFLAGS, client\_PCFILES for each client, i.e., library or executable to be build
- define automake conditional and preprocessor define COIN\_HAS\_PRIM
- no test that package actually works
configure macro checking for pkg-config package

AC_COIN_CHK_PKG(prim, client packages, .pc file name, default action, cmdopts)

- determine availability of primary package prim
- assemble compiler flags, linker flags, and data directories
- augment client_CFLAGS, client_LFLAGS, client_PCFILES for each client, i.e., library or executable to be build
- define automake conditional and preprocessor define COIN_HAS_PRIM
- no test that package actually works

Example: AC_COIN_CHK_PKG(CoinUtils, [ClpLib])

- checks availability of coinutils.pc
- appends to CLPLIB_CFLAGS, CLPLIB_LFLAGS, CLPLIB_PCFILES
- defines COIN_HAS_COINUTILS (automake conditional, C preprocessor define)
configure macro checking for non-pkg-config packages

\texttt{AC_COIN_CHK_LIB([prim], [client packages], [lflgs], [cflgs], [dflgs],
[func], [other libraries], [dfltaction], [cmdopts])}

- determine availability and usability of package \texttt{prim}
- assemble \texttt{compiler flags, linker flags, and data directories} (given \texttt{cflgs, lflgs, dflgs}, or overwritten by user (\texttt{--with-prim-lflags}, etc.))
- augment \texttt{client\_CFLAGS\_NOPC} and \texttt{client\_LFLAGS\_NOPC} for each client, i.e., library or executable to be build
- define \texttt{automake conditional} and \texttt{preprocessor define} \texttt{COIN\_HAS\_PRIM}
- check that \texttt{func} is available with given linker flags (calls linker)
configure macro checking for non-pkg-config packages

AC_COINCHK_LIB([prim], [client packages], [lflgs], [cflgs], [dflgs],
[func], [other libraries], [dftlaction], [cmdopts])

- determine availability and usability of package prim
- assemble compiler flags, linker flags, and data directories (given cflgs, lflgs, dflgs, or overwritten by user (--with-prim-lflags, etc.))
- augment client_CFLAGS_NOPC and client_LFLAGS_NOPC for each client, i.e., library or executable to be build
- define automake conditional and preprocessor define COIN_HAS_PRIM
- check that func is available with given linker flags (calls linker)

Example: AC_COINCHK_LIB(AMD, [ClpLib], [-lamd],
[-I/usr/include/suitesparse], [], [amd_defaults])

- checks whether function amd_defaults is available with -lamd
- appends -lamd to CLPLIB_LFLAGS_NOPC
- appends -I/usr/include/suitesparse to CLPLIB_CFLAGS_NOPC
- defines COIN_HAS_AMD (automake conditional, C preprocessor define)
Automake brings script `compile`:

Usage: compile [--help] [--version] PROGRAM [ARGS]

Wrapper for compilers which do not understand `-c -o`.
Remove `-o dest.o` from ARGS, run PROGRAM with the remaining arguments, and rename the output as expected.
Automake brings script compile:

Usage: compile [--help] [--version] PROGRAM [ARGS]

Wrapper for compilers which do not understand '-c -o'.
Remove '-o dest.o' from ARGS, run PROGRAM with the remaining arguments, and rename the output as expected.

Example:
libtool: compile:  compile cl -DHAVE_CONFIG_H -I. -I../src -DNDEBUG
                -c ../src/CoinPackedVector.cpp -o CoinPackedVector.obj
Microsoft (R) C/C++-Optimierungscompiler Version 19.00.24213.1 fuer x64
Copyright (C) Microsoft Corporation. Alle Rechte vorbehalten.

CoinPackedVector.cpp
C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\INCLUDE\xlocale(341):
  warning C4530: C++-Handler verwendet, aber Entladensemantik ist nicht aktiviert.
  Geben Sie /EHsc an.
C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\INCLUDE\limits(210):
  warning C4577: "noexcept" wird ohne angegebenen Ausnahmebehandlungsmodus verwendet.
  Die Beendigung bei einer Ausnahme ist nicht sichergestellt. Geben Sie "/EHsc" an.
https://github.com/swig/cccl:


cccl is a wrapper around Microsoft’s cl.exe compiler. It translates parameters given by [OPTIONS] that Unix cc understands into parameters that cl understands.
Alternative: cccl

https://github.com/swig/cccl:


ccl is a wrapper around Microsoft’s cl.exe compiler. It translates parameters given by [OPTIONS] that Unix cc understands into parameters that cl understands.

Example:


cl "-nologo" "-DHAVE_CONFIG_H" "-I." "-I../src" "-DNDEBUG" "-c"
    "../src/CoinPackedVector.cpp" "-FoCoinPackedVector.obj"
CoinPackedVector.cpp
C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\INCLUDE\xlocale(341):
    warning C4530: C++-Handler verwendet, aber Entladesemantik ist nicht aktiviert. Geben Sie /EHsc an.
C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\INCLUDE\limits(210):
# Static vs. Shared Libraries

<table>
<thead>
<tr>
<th>Static Libs</th>
<th>Unix</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>.a</td>
<td>.a</td>
<td>.a</td>
</tr>
<tr>
<td>.lib</td>
<td>.lib</td>
<td>.lib</td>
</tr>
<tr>
<td>a collection (archive) of <strong>object files</strong></td>
<td>a collection of <strong>object files</strong></td>
<td>a collection of <strong>object files</strong></td>
</tr>
<tr>
<td>all functions and data are exposed (unless static)</td>
<td>all functions and data are exposed (unless static)</td>
<td>all functions and data are exposed (unless static)</td>
</tr>
<tr>
<td>no record of dependencies on other libraries</td>
<td>no record of dependencies on other libraries</td>
<td>no record of dependencies on other libraries</td>
</tr>
</tbody>
</table>
### Static vs. Shared Libraries

<table>
<thead>
<tr>
<th></th>
<th>Unix</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Static Libs</strong></td>
<td>.a</td>
<td>.lib</td>
</tr>
<tr>
<td></td>
<td>a collection (archive) of <strong>object files</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>all functions and data are exposed (unless static)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>no record of dependencies on other libraries</td>
<td></td>
</tr>
<tr>
<td><strong>Shared Libs</strong></td>
<td>.so</td>
<td>.dll (+.lib for linking)</td>
</tr>
<tr>
<td></td>
<td>a collection of <strong>functions and data</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>everything exposed by default</td>
<td>nothing exposed by default</td>
</tr>
<tr>
<td></td>
<td>records dependencies on other libraries</td>
<td></td>
</tr>
</tbody>
</table>
Shared Libraries on Windows with Libtool

- Functions to be exported by a DLL on Windows should be marked with the attribute _declspec(dllexport) in the source code (and _declspec(dllimport) when importing).
- Alternatively, a .def file with a list of all functions to be exported can be specified.
• Functions to be exported by a DLL on Windows should be marked with the attribute `declspec(dllexport)` in the source code (and `declspec(dllimport)` when importing).

• Alternatively, a `.def` file with a list of all functions to be exported can be specified.

• For C (and Fortran?) libraries, libtool can automatically expose all functions in a DLL by parsing output of `nm`. This is sufficient to build, e.g., a Glpk DLL.

• For C++, libtool does not do this. Thus, `declspec(dllexport/dllimport)` should be added to headers files of COIN-OR projects. (See also `AC_LIBTOOL_WIN32.DLL`.)

• Might then also change the GCC default to not expose all functions by default.
Shared Libraries on Windows with Libtool

- Functions to be exported by a DLL on Windows should be marked with the attribute `__declspec(dllexport)` in the source code (and `__declspec(dllimport)` when importing).
- Alternatively, a `.def` file with a list of all functions to be exported can be specified.
- For C (and Fortran?) libraries, libtool can automatically expose all functions in a DLL by parsing output of `nm`. This is sufficient to build, e.g., a Glpk DLL.
- For C++, libtool does not do this. Thus, `__declspec(dllexport/dllimport)` should be added to headers files of COIN-OR projects. (See also `AC_LIBTOOL_WIN32_DLL`.)
- Might then also change the GCC default to not expose all functions by default.

Example (CPLEX `cpxconst.h`):

```c
#ifndef BUILD_CPXLIB
#   if defined(_WIN32)
#       define CPXLIBAPI __declspec(dllexport)
#   elif defined(__GNUC__) && defined(__linux__)
#       define CPXLIBAPI __attribute__((visibility("default")))
#   else /* !defined(_WIN32) */
#       define CPXLIBAPI __declspec(dllexport)
#   endif

... ...
#endif
```
Projects with autotools-update branch

Development happens in

- BuildTools/trunk

and branch autotools-update of

- BuildTools/ThirdParty/Blas
- BuildTools/ThirdParty/Glpk
- BuildTools/ThirdParty/Lapack
- CoinUtils
- Osi
- Clp
- Cgl
- Cbc
Hands-On Session (after lunch): possible things to do

Try out projects that already work with BuildTools/trunk:

- try fetch & build of autotools-update branches, try get.dependencies.sh
- report problems, preferably with patch to fix ([http://projects.coin-or.org/BuildTools/newticket](http://projects.coin-or.org/BuildTools/newticket))
- Linux and Mac OS X should work, Windows static libraries could also work
- Makefiles of examples might still need revision
Hands-On Session (after lunch): possible things to do

Try out projects that already work with BuildTools/trunk:

- try fetch & build of autotools-update branches, try get.dependencies.sh
- report problems, preferably with patch to fix (http://projects.coin-or.org/BuildTools/newticket)
- Linux and Mac OS X should work, Windows static libraries could also work
- Makefiles of examples might still need revision

Migrate more projects to BuildTools/trunk:

- create branch autotools-update
- clear out project base directory
- update configure.ac, Makefile.am
- setup or update Travis and Appveyor files
Hands-On Session (after lunch): possible things to do

Try out projects that already work with BuildTools/trunk:

- try fetch & build of autotools-update branches, try get.dependencies.sh
- report problems, preferably with patch to fix ([http://projects.coin-or.org/BuildTools/newticket](http://projects.coin-or.org/BuildTools/newticket))
- Linux and Mac OS X should work, Windows static libraries could also work
- Makefiles of examples might still need revision

Migrate more projects to BuildTools/trunk:

- create branch autotools-update
- clear out project base directory
- update configure.ac, Makefile.am
- setup or update Travis and Appveyor files

Improve BuildTools/trunk:

- Windows: compile vs cccl; missing compiler flags (-EHsc, -Zi)
- make API explicit: __declspec(dllexport/dllimport), __attribute__((visibility(...)))
- Windows: proper DLL builds
- check setup of .pc files, static vs shared (.private)
- support systems without pkg-config, redistrib. own pkgconf ([https://github.com/pkgconf/pkgconf](https://github.com/pkgconf/pkgconf))?
- update to current automake?, try out new Python compilation support
End.