MinGW-w64 - for 32 and 64 bit Windows

A complete runtime environment for gcc

Brought to you by: jon_y, ktietz70, nightstrike

UsingLinuxBinaries

Authors: Anonymous

Overview

Here we will discuss installation of mingw-w64 cross-compilers for Windows 32-bit and 64-bit cross-compilers under Linux. There are two types of binaries for mingw-w64 on-line right now, the "1.0" stable and snapshot builds, and the "cutting edge" builds that use the latest available builds of gcc and other compilers. They are installed differently (for now) but are used the same way. Both run under Linux and generate Windows binary executable files of the format foo.exe. If you do your own builds from the source code, one method or the other, or both, should work for you, with minor changes that reflect the way that you do things.

The examples given here are bash scripts for Linux x86_64 that have been tested under Fedora Core 14. Differences for use under 32-bit Linux or other distributions and even different shells are left as an exercise for the reader. You need to edit these script to make the build dates match those of your downloaded tarballs, or you can use more elaborate scripts.

Installing

Installing the 1.0 Stable Releases and Daily Builds

The daily builds are available here. The binary files are of the form

> mingw-w64-1.0-bin-x86_64-linux-[yyyymmdd].tar.bz2 for generating Windows 64-bit binaries, and mingw-w32-1.0-bin-x86_64-linux-[yyyymmdd].tar.bz2 for generating Windows 32-bit binaries,

where [yyyymmdd] is the build date, as 20110222 for February 22, 2011.

To install for all users, su to root and install in /usr/mw32 and /usr/mw64; you may select other folder names than mw32 and mw64. A bash script that will accomplish this is

```
#Go to SU to install the 1.0 cross-compilers for all users
su        #SU to root; give root password to continue
cd /usr/  #Make installation directories in /usr/
#mkdir mw32   #If this is the first time, and mw32 doesn't exist yet
#mkdir mw64   #If this is the first time, and mw64 doesn't exist yet

#Install the 1.0 W32 cross-compiler
cd mw32
rm -rf ./*    #Remove previous installation, if any
tar -xvf /home/beard9g/Downloads/mingw-w32-1.0-bin_x86_64-linux_[yyyymmdd].tar.bz2
cd ..

#Install the W64 cross-compiler
cd mw64
rm -rf ./*    #Remove previous installation, if any
tar -xvf /home/beard9g/Downloads/mingw-w64-1.0-bin_x86_64-linux_[yyyymmdd].tar.bz2
```

Installing the bleeding edge builds

The bleeding edge builds will not install under root and execute by users unless you su to root; you will get permission errors. Until this is changed, you need to install these builds locally. A bash script that will accomplish this is

```
#INSTALLATION SCRIPT FOR BLEEDING EDGE COMPILERS
#Install bleeding edge compilers locally, in ~
#mkdir mw32   #If this is the first time, and mw32 doesn't exist yet
#mkdir mw64   #If this is the first time, and mw64 doesn't exist yet

#Install the W32 cross-compiler
cd mw32
rm -rf ./*    #Remove previous installation, if any
tar -xvf ../Downloads/mingw-w32-bin_x86_64-linux_20110219.tar.bz2
cd ..

#Install the W64 cross-compiler
cd mw64
rm -rf ./*    #Remove previous installation, if any
tar -xvf ../Downloads/mingw-w64-bin_x86_64-linux_20110214.tar.bz2
```

Using the cross-compilers to generate Windows 32-bit and 64-bit binary executables

All you need to do to use any of the compilers is to insert mw32/bin or mw64/bin into the PATH at the beginning. The compilers that generate Windows 32-bit scripts use a prefix of i686-w64-mingw32- (or, -w32- for Linux x86) on the usual Linux compiler binary file names, the compilers that generate the Windows 64-bit scripts use a prefix of x86_64-w64-mingw32- (or -w32-). The host options m32 and m64 are used in the example below for clarity and emphasis. Note that if no cross-compiler is used, the script reverts to a simple native compilation as installed in your Linux distribution, which may be useful for development and testing. The example below is cut down from a project that compiled a C engine that was executed as part of a package that used a Fortran 95/2003 main program. The calls to echo the version and target architectures are for keeping track of which versions are used in the builds and may be commented out if you don't want that information every build.

```
#CROSS-COMPILATION SCRIPT
#Pick just one, and only one, compiler to use; pick none to use system gcc
CCW32=0
CCW64=0
CCW32BE=0
CCW64BE=1

if [ $CCW32 = 1 ]; then
CCPATH="~/mw32/bin/:"
PREFIX="i686-w64-mingw32-"
SIZEOPT="-m32"

elif [ $CCW64=1 ]; then
CCPATH="~/mw64/bin/:"
PREFIX="x86_64-w64-mingw32-"
SIZEOPT="-m64"

elif [ $CCW32BE = 1 ]; then
CCPATH="/usr/mw32/bin/:"
PREFIX="i686-w64-mingw32-"
SIZEOPT="-m32"

elif [ $CCW64BE = 1 ]; then
CCPATH="/usr/mw64/bin/:
PREFIX="x86_64-w64-mingw32-"
SIZEOPT="-m64"

else
CCPATH=""
PREFIX=""
SIZEOPT=""
fi

#Add path to cross-compiler at beginning of PATH
PATH=$CCPATH$PATH #Works for all compiler calls within the shell of this script
#export PATH  #Will not take unless you are root but may be useful if you call other scripts from this one

#Get information on versions
$PREFIX"gcc" --version
$PREFIX"gcc" -dumpmachine
$PREFIX"gfortran" --version
$PREFIX"gfortran" -dumpmachine

#Use the compiler
$PREFIX"gcc" [options] -static $SIZEOPT -c [your C or C++ program, compiled to .o]
$PREFIX"gfortran" [options] -static [Fortran 95 programs and modules] $SIZEOPT -o [your executable binary name].exe

#PATH will revert to the system value when the script exits
```

Related

Wiki2: Home