# 1 LabBook

I keep looking up the same commands over and over again, so here I'll denote some useful ones.

# 2 Linux Commands (Ubuntu)

## 2.1 Find a library on the system

There seems to be multiple ways to do this, and sometimes one command works over another, not sure why.

Requires installing apt-file

ldd - print shared object dependencies. Very useful for debugging missing shared libraries.

Can also try grep with ls -R

Then there is find

## 2.2 Building on linux

## 2.3 Linking static libraries into shared

When passing arguments to the linker you need to ensure you use the '-Wl,–whole-archive' and '-Wl,–no-whole-archive' option. Wrap these around static libraries that you are tyring to pull into a shared library.

This is necessary to tell the linker to pull all the functions from the library into the shared library you are building. Otherwise, only some will be pulled in and you will get a linker error.

It seems there is also another way here

Use -l: instead of -l. For example -l:libXYZ.a to link with libXYZ.a. Notice the lib written out, as opposed to -lXYZ which would auto expand to libXYZ.

Note, these commands can be embedded into a CMake script by passing to 'TARGET_LINK_LIBRARIES'

## 2.4 Inspective broken builds

Useful commands (linux)

List all the shared object libraries that libx depends on

List the symbols in a library, along with their status (found, undefined etc.)

Use the -D option to inspect dynamic symbols only

Pipe output of nm into grep to search for specific function

You can examine the Rpath on Linux thus:

# 3 CMake

Note: cmake is now on pip version 3.17. pip install cmake.

### 3.1 Copy or install a file

Copy during configuration stage
    Copy at install time
    amsmath

### 3.2 Docker

Turn off all containers
    using a filter
    A great resource on explaining the docker basics
    amsmath hyperref

## 4 What is the difference between msys and mingw?

Shamelesly stolen from
    MinGW doesn't provide a linux-like environment, that is MSYS(2) and/or Cygwin

    Cygwin is an attempt to create a complete UNIX/POSIX environment on Windows. MinGW is a C/C++ compiler suite which allows you to create Windows executables - you only need the normal MSVC runtimes, which are part of any normal Microsoft Windows installation.

    MinGW provides headers and libraries so that GCC (a compiler suite, not just a "unix/linux compiler") can be built and used against the Windows C runtime.

    MSYS is a fork of Cygwin (msys.dll is a fork of cygwin.dll) cygwyn gcc + cygwin environment defaults to producing binaries linked to the (GPL) cygwin dll (or cygwin1.dll???) mingw + msys defaults to producing binaries linked to the platform C lib.

    MinGW: It does not have a Unix emulation layer like Cygwin, but as a result your application needs to specifically be programmed to be able to run in Windows,

    MinGW forked from version 1.3.3 of Cygwin

    Unlike Cygwin, MinGW does not require a compatibility layer DLL and thus programs do not need to be distributed with source code.

    This means, other than Cygwin, MinGW does not attempt to offer a complete POSIX layer on top of Windows, but on the other hand it does not require you to link with a special compatibility library.

    Cygwin comes with the MingW libaries and headers and you can compile without linking to the cygwin1.dll by using -mno-cygwin flag with gcc. I greatly prefer this to using plain MingW and MSYS. ( This does not work any more with cygwin 1.7.6. gcc: The -mno-cygwin flag has been removed; use a mingw-targeted cross-compiler. )

    MSYS is a collection of GNU utilities such as bash, make, gawk and grep to allow building of applications and programs which depend on traditionally UNIX tools to be present. It is intended to supplement MinGW and the deficiencies of the cmd shell.

    An example would be building a library that uses the autotools build system. Users will typically run "./configure" then "make" to build it. The configure

shell script requires a shell script interpreter which is not present on Windows systems, but provided by MSYS.

A common misunderstanding is MSYS is "UNIX on Windows", MSYS by itself does not contain a compiler or a C library, therefore does not give the ability to magically port UNIX programs over to Windows nor does it provide any UNIX specific functionality like case-sensitive filenames. Users looking for such functionality should look to Cygwin or Microsoft's Interix instead.

MSYS2 uses Pacman (of Arch Linux) to manage its packages and comes with three different package repositories: - msys2: Containing MSYS2-dependent software - mingw64: Containing 64-bit native Windows software (compiled with mingw-w64 $x86_64 toolchain) - mingw32 : Containing 32-bit native Windows software (compiled with mingw-w64 i686 toolchain)

Cygwin provides a runtime library called cygwin1.dll that provides the POSIX compatibility layer where necessary. The MSYS2 variant of this library is called msys-2.0.dll and includes the following changes to support using native Windows programs: 1) Automatic path mangling of command line arguments and environment variables to Windows form on the fly.

MSYS is a fork of an old Cygwin version with a number of tweaks aimed at improved Windows integration, whereby the automatic POSIX path translation when invoking native Windows programs is arguably the most significant.

amsmath

## 4.1 Git commands

### 4.1.1 Submodule

Update submodules

amsmath

# 5 Cross platform CMake

https://gitlab.kitware.com/cmake/community/-/wikis/doc/tutorials/How-To-Write-Platform-Checks