Installation of GCC and boost

In which directory should I install software for linux ?

- `/bin` essential program / command for starting and running the linux system (i.e. linux commands)
- `/sbin` essential system admin tools for starting and running the linux system
- `/usr/bin` user-installed program/ command, via package manager such as `apt` and `yum`
- `/usr/sbin` user-installed system admin tools, via package manager such as `apt` and `yum`
- `/usr/local/bin` user-compiled program / command, which is built from source after downloading
- `/usr/local/sbin` user-compiled system admin tools, which is built from source after downloading

Installation of Linux software in general

Installation of linux software that shipped with linux distro can be done by :

```
>> sudo apt install software                    // for ubuntu
>> sudo yum install software                    // for redhat
```

Installation of linux software that requires manual compilation from source code *(name of script may be different, see readme)* :

```
>> wget https://abc.org/path/path/software.tar.gz   // download zipped source
>> tar -zxvf software.tar.gz                        // unzip source
>> cd software
>> configure                                        // run config script provided
>> make                                             // run make to compile source
>> sudo make install                                // deploy artifacts to /usr/bin or /usr/local/bin
                                                    //                     /usr/lib or /usr/local/lib etc
```

Or using GIT, when available :

```
>> git clone https://abc.org/git/software.git
>> git branch -a
>> git checkout remote/origin/release/target-version
>> configure
>> make
>> sudo make install
```

Sometimes if `git clone` fails with error message `unable to access ... server certificate verification failed`, then we need to install `ca-certificates` :

```
>> sudo apt-get update
>> sudo apt-get install apt-transport-https ca-certificates -y
>> sudo update-ca-certificates
```

Mixing different version gcc and different C++ standards (11, 14, 17 20)

Given same version gcc compiler, it is safe to link objects compiled with different C++ standard together into the same artifact :

- object compiled with `gcc-12` and `-std=c++11`
- object compiled with `gcc-12` and `-std=c++14`
- object compiled with `gcc-12` and `-std=c++17` ► safe to link them together

Given different version gcc compilers, it is not necessarily safe to link objects compiled with same C++ standard :

- object compiled with `gcc-5` and `-std=c++11`
- object compiled with `gcc-8` and `-std=c++11` ► safe to link them together, as `-std=c++11` is stable in `gcc-5&8`

However when the standard is still evolving in one of the compiler :

- object compiled with `gcc-7` and `-std=c++17`
- object compiled with `gcc-8` and `-std=c++17` ► not safe to link, as `-std=c++17` is unstable in `gcc-7`, still evolving

The idea can be generalized to different version compilers with different standard :

- object A compiled with `gcc-5` and `-std=c++11`
- object B compiled with `gcc-7` and `-std=c++14`
- object C compiled with `gcc-8` and `-std=c++17` ► safe to link them together, as each standard is stable in the compiler

Installation of GCC

In Daiwa Capital Market, I need to upgrade GCC from `gcc-8.4` to `gcc12.2`, the latter does not come with Ubuntu 18, thus it cannot be installed with `apt-get`. Please go to GCC official site, check its GIT URL to the latest version of source code.

Step 1 Download source code, `git clone` takes about 20 minutes and it creates a folder named `gcc-source`.

```
>> git clone https://gcc.gnu.org/git/gcc.git gcc-source
>> git branch -a
>> git checkout remotes/origin/releases/gcc-12
>> git checkout -b my_develop
```

Step 2 Try to build GCC for the first time *(it probably fails, while notifies you some missing prerequisites)*. Please do not compile GCC in the original folder `gcc-source`, as told by GCC official site, we need to create another folder, says `gcc-build` next to it.

```
>> mkdir gcc-build
>> cd gcc-build
>> ./../gcc-source/configure --prefix=$HOME/install/gcc-12 --enable-languages=c,c++
```

Option `--prefix` determines the destination of artifacts. if it is omitted, artifacts will be deployed to default locations :

- `/usr/local/bin` for executable `gcc`
- `/usr/local/lib` for library `libstdc++.so`

The first run of configure usually fails, it will list some missing prerequisites, such as :

- MPC GNU Multiple-precision C library
- MPFR GNU Multiple-precision floating-point rounding library
- GMP GNU Multiple Precision Arithmetic Library
- PPL Parma Polyhedra Library (memory optimizations)
- ELF Executable and Linkable Format library

We have to `apt-get install` each of the missing prerequisites. However the above are the their package names, in order to find out, we need `ap-cache`, and filter the long list of output with `grep dev`, as `dev` is a common keyword.

```
>> apt-cache pkgnames                // list all package names
>> apt-cache search MPC  | grep dev   // search package names with MPC,  we have : libmpc-dev
>> apt-cache search MPFR | grep dev   // search package names with MPFR, we have : libmpfrc++-dev
>> apt-cache search GMP  | grep dev   // search package names with GMP,  we have : libgmp-dev
```

Once we know the package names, we can install them :

```
>> sudo apt-get install libmpc-dev
>> sudo apt-get install libmpfrc++-dev
>> sudo apt-get install libgmp-dev
>> sudo apt-get install gcc-multilib
```

Step 3 Try to build GCC for the second time. This time, we use default destination for artifacts. Besides, it may fail again, informing you to disable `multilib`, please do so.

```
>> ./../gcc-source/configure --enable-languages=c,c++                  // it fails, requiring to disable multilib
>> ./../gcc-source/configure --enable-languages=c,c++ --disable-multilib // it works
>> make -j12
>> sudo make install
Oll Korrect

>> gcc --version
still old version, why?
```

Step 4 Locate the artifacts. Here is the list :

```
// *** Compilers ***
/usr/local/bin/x86_64-linux-gnu-c++*
/usr/local/bin/x86_64-linux-gnu-g++*
/usr/local/bin/x86_64-linux-gnu-gcc*
/usr/local/bin/x86_64-linux-gnu-gcc-12.2.1*

// *** Standard C++ header-only libraries ***
/usr/local/include/c++/12.2.1                          // which keeps : <vector> <optional> <atomic>

// *** Standard C++ libraries ***
/usr/local/lib64                                       // which keeps : libstdc++.so.6.0.30 libatomic.so

// *** GNU C libraries ***
/usr/local/lib/gcc/x86_64-linux-gnu/12.2.1             // which keeps : libgcc.a libgcov.a
/usr/local/lib/gcc/x86_64-linux-gnu/12.2.1/include     // which keeps : <avx*.h> <cpuid.h> <*intrin.h>
```

Here are the substeps we need to point to these artifacts :

- set environment variable `PATH` to compiler path so that `gcc` can be located when we invoke `gcc`
- in `Makefile`, add include path `-I` to standard C++ header path so that `gcc` can locate `<vector>`
- in `Makefile`, add library path `-L` to standard C++ library path so that `gcc` can locate `libstdc++.a`
- set environment variable `LD_LIBRARY_PATH` to standard C++ library path so that application can locate `libstdc++.so`
- set symbolic link in `/usr/bin` to desired version `gcc`

```
>> export PATH=/usr/local/bin:$PATH
>> export LD_LIBRARY_PATH=/usr/local/lib64:$LD_LIBRARY_PATH
>> env                                                 // list all environment variables
>> env | grep PATH                                     // list location of gcc
>> env | grep LD_LIBRARY_PATH                          // list location of libstdc++.so
```

We can also export environment variables in `~/.profile`, so that they are loaded when terminal is instantiated.

```
>> cd /usr/bin
>> ll *gcc*
>> sudo ln -s /usr/local/bin/x86_64-linux-gnu-g++-12.2.1    g++-12
>> sudo ln -s /usr/local/bin/x86_64-linux-gnu-gcc-12.2.1    gcc-12
>> sudo ln -s /usr/local/bin/x86_64-linux-gnu-gcc-ar        gcc-ar-12
>> sudo ln -s /usr/local/bin/x86_64-linux-gnu-gcc-nm        gcc-nm-12
>> sudo ln -s /usr/local/bin/x86_64-linux-gnu-gcc-ranlib    gcc-ranlib-12
>> sudo ln -sf g++-12 g++
>> sudo ln -sf gcc-12 gcc
>> gcc --version
```

Use `ln -s path symbol` for creating new symbolic link, use `ln -sf path symbol` for forced-assigning existing link with new value.

Running test application

Suppose we can compile a test program, now we run it :

```
>> cd ~/dev
>> ./build/debug/Test
```

However we get errors in runtme :

```
./build/debug/Test: /usr/lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3.4.29' not found (required by ./build/debug/Test)
./build/debug/Test: /usr/lib/x86_64-linux-gnu/libstdc++.so.6: version `GLIBCXX_3.4.26' not found (required by ./build/debug/Test)
```

which means it cannot locate the correct `libstdc++.so` in runtime. More information about my situation :

- my old `libstdc++.so for gcc-8.4.0` locates in `/usr/lib/x86_64-linux-gnu/libstdc++.so.6.0.25`
- my new `libstdc++.so for gcc-12.2.1` locates in `/usr/local/lib64/libstdc++.so.6.0.30`

Hence `Test` is still loading the old version shared library. It can be solved by `export LD_LIBRARY_PATH` as shown in previous page.

Installation of CMake
We need to upgrade cmake from :

- existing cmake 3.10
- to new cmake 3.25

Step 1 Download source code. Look for latest version in cmake official site. We use `wget` instead of `git clone` this time :

```
>> cd ~
>> wget https://github.com/Kitware/CMake/releases/download/v3.25.0-rc4/cmake-3.25.0-rc4.tar.gz
>> tar -zxvf cmake-3.25.0-rc4.tar.gz
>> cd cmake-3.25.0-rc4
>> nvim README.rst              // good practice to readme before go ahead
```

Step 2 Compile source code using the 3 step approach. Like building GCC, it may fail and prompt you to install prerequisites.

```
>> ./bootstrap                  // fails and prompt to install prerequisite OPENSSL
>> sudo apt install libssl-dev
>> ./bootstrap
>> make                         // compilation
>> sudo make install            // deployment
>> cmake -version               // verify
>> which cmake                  // verify, inside /usr/bin
```

What are the differences between option `-std=c++20` and `-std=gnu++20`?

- `-std=c++20` and
- `-std=gnu++20`

Installation of Boost
We need to upgrade boost from :

- existing boost 1.64
- to new boost 1.80

Step 1 Download source code. Look for latest version in boost official site. We use `wget` instead of `git clone` this time :

```
>> cd ~
>> wget -O boost_1_80_0.tar.gz https://sourceforge.net/projects/boost/files/boost/1.80.0/boost_1_80_0.tar.gz/download
>> tar -zxvf boost_1_80_0.tar.gz
>> cd boost_1_80_0
```

Step 2 Compile source code using the 2 step approach (`make` and `make install` are merged as `b2`). Again, need prerequisites.

```
>> sudo apt-get update
>> sudo apt-get install build-essential g++ \
                        python-dev autotools-dev \
                        libicu-dev libbz2-dev libboost-all-dev // stuffs that boost depends on
>> ./bootstrap.sh --prefix=/usr/local
>> sudo ./b2 --with=all -j $num_cores install
```

where `num_cores` can be found by running the following command :

```
>> num_cores=`cat /proc/cpuinfo | grep "cpu cores" | uniq | awk '{print $NF}'`
>> echo $num_cores
```

Lets see where the artifacts are :

```
>> cd /usr/local/lib
>> ll
-rwxr-xr-x 1 root root    13888 Nov 21 10:34 libboost_atomic.a*
lrwxrwxrwx 1 root root       25 Nov 21 10:40 libboost_atomic.so -> libboost_atomic.so.1.80.0*
-rwxr-xr-x 1 root root    18592 Nov 21 10:40 libboost_atomic.so.1.80.0*
-rwxr-xr-x 1 root root   155062 Nov 21 10:34 libboost_chrono.a*
lrwxrwxrwx 1 root root       25 Nov 21 10:40 libboost_chrono.so -> libboost_chrono.so.1.80.0*
-rwxr-xr-x 1 root root    41512 Nov 21 10:40 libboost_chrono.so.1.80.0*
-rwxr-xr-x 1 root root   163970 Nov 21 10:34 libboost_container.a*
lrwxrwxrwx 1 root root       28 Nov 21 10:40 libboost_container.so -> libboost_container.so.1.80.0*
-rwxr-xr-x 1 root root   103816 Nov 21 10:40 libboost_container.so.1.80.0*
```

Miscellaneous – installation of make

In DCME, there is a symbolic link `gmake` which points to `make`, and I make mistake by switching destination path and symbol when I run command `ln -sf`, which ends up deleting executable `make`, and I cannot compile C++ file anymore. The solution is to reintall :

```
>> sudo apt install make            // it refused to install make, as it finds make in apt list
>> make --version                   // it fails to locate it, as make is deleted
```

then I was forced to install `make` by :

```
>> sudo apt --reinstall install make
>> make --version                   // it works finally
```

Miscellaneous – special folders in Ubuntu

In WSL / Ubuntu, we have :

```
/usr/bin                           // symlink to all versions gcc             export to PATH
/usr/local/bin                     // physical location of gcc-12.2.1         export to PATH
/usr/local/include/c++/12.2.1      // header for std c++ (header only lib)    add to -I Makefile
/usr/local/include/boost           // header for boost 1.80.0                 add to -I Makefile
/usr/local/lib64                   // libstdc++.so / libstdc++.a              add to -L Makefile, export to LD_...
/usr/local/lib                     // libboost_thread.so / libboost_thread.a  add to -L Makefile, export to LD_...
```

Reference

Can we mix artifacts built with different versions gcc? With same or different C++ standard (11/14/17/20)?

- search Stack overflow : *46746878* *(please see next page)*

How to install (build from source) gcc-12 on Ubuntu?

- search Stack overflow : *70835585* *(please see next page)*
- search Youtube : *Mike Shah Building GCC 11 (and beyond) from Source*

How to install (build from source) gcc-12 on Redhat?

- search Medium.com : *Darrenjs building-gcc-from-source*

How to handle runtime GLIBCXX error?

- search : *Omair Majid, What is this GLIBCXX error?*

gcc-12 is not available in ubuntu 20.04, so we need to compile it from source code, here are the steps which I borrowed from this video:

- **Step 1**: clone gcc source code and checkout gcc-12 branch

```
$ git clone https://gcc.gnu.org/git/gcc.git gcc-source
$ cd gcc-source/
$ git branch -a
$ git checkout remotes/origin/releases/gcc-12
```

- **Step 2**: make another build dir

Note this is important as running `./configure` from within the source directory is not supported as documented here.

```
$ mkdir ../gcc-12-build
$ cd ../gcc-12-build/
$ ./../gcc-source/configure --prefix=$HOME/install/gcc-12 --enable-languages=c,c++
```

- **Step 3**: installing GCC prequisites and run configure again

The missing libraries will be shown in above `./confgiure` output, search and install them one by one.

```
$ apt-cache search MPFR
$ sudo apt-get install libmpfrc++-dev
$ apt-cache search MPC | grep dev
$ sudo apt-get install libmpc-dev
$ apt-cache search GMP | grep dev
$ sudo apt-get install libgmp-dev
$ sudo apt-get install gcc-multilib
$ ./../gcc-source/configure --prefix=$HOME/install/gcc-12 --enable-languages=c,c++
```

An alternartive is to run the download_prerequisites script.

```
$ cd ../
$ cd gcc-source/
$ ./contrib/download_prerequisites
$ ./../gcc-source/configure --prefix=$HOME/install/gcc-12 --enable-languages=c,c++
```

- **Step 4**: compile gcc-12

```
$ make -j16
```

Still flex is missing:

```
$ sudo apt-get install flex
$ ./../gcc-source/configure --prefix=$HOME/install/gcc-12 --enable-languages=c,c++
$ make -j16
$ make install
```

Another way is to use Ubuntu 22.04 where gcc-12 is available. In Ubuntu 22.04, gcc-12 can be installed with `apt`:

```
$ sudo apt install gcc-12
```