Search or go to...
Project
openfoam
Manage
Plan
Issues
Issue boards
Milestones
Wiki
Code
Deploy
Analyze

building

Last edited by **Mark Olesen** 11 months ago

Linux, Unix-like systems

Version	Build
v2406	Build
v2312	Build
v2306	Build
v2212	Build
v2206	Build
v2106	Build
v2012	Build
v2006	Build
v1912	Build
v1906	Build
older	*obsolete*

Packaging systems

System	Links	Status	Notes
spack	package openfoam	Actively maintained by OpenCFD	notes
EasyBuild	package OpenFOAM	Maintained independently, with input from OpenCFD	
debian, RPM	*See precompiled*	Actively maintained by OpenCFD	

spack

The installation of OpenFOAM with spack will generally require the latest (development version) of spack. If this is available, you can install OpenFOAM in various configurations and dependencies, but typically can simply install directly:

```
$ spack install openfoam
```

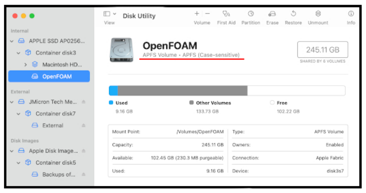
easybuild

The installation of OpenFOAM with easybuild will generally require the latest (development version) of easybuild.

Darwin (Mac-OS)

The support for Darwin is complete, but less well tested than Linux.

- Compilation uses the *system* **clang** compiler.
- The Darwin build (and operation) requires a ***case-sensitive file system***.
  (For older systems, this can be created as a disk image and mounted)



Used 9.16 GB · Other Volumes 133.73 GB · Free 102.22 GB

Mount Point:	/Volumes/OpenFOAM	Type:	APFS Volume
Capacity:	245.11 GB	Owners:	Enabled
Available:	102.45 GB (230.3 MB purgeable)	Connection:	Apple Fabric
Used:	9.16 GB	Device:	disk3s7

Known issues

CGAL

- ThirdParty CGAL will normally need to be compiled *without* mpfr/gmp. This should be done manually prior to building OpenFOAM or other ThirdParty. For example,

  ```
  cd $WM_THIRD_PARTY_DIR
  ./makeCGAL gmp-none mpfr-none
  ```

  The `wmake/rules/darwin64Clang/cgal` file avoids references to gmp/mpfr libraries.

Windows (cross-compiled)

Windows 64bit binaries can be generated on 64bit Linux by ***cross-compilation***.

Tuning

Different configurations

Sometimes it is useful to switch between entire sets of configuration preferences without re-editing the files each time. This is the purpose of the `FOAM_CONFIG_ETC` variable. It specifies an absolute path, or a path relative to the project directory where various configuration files can be found. These are selected in preference to the normal shipped configuration files.

This allows swapping in a set of different preferences without modifying the regular settings. See cross-compilation for an example of its use.

Different compiler versions

By default, OpenFOAM handles newer/older non-system compilers as *ThirdParty* installations and uses the combination of `WM_COMPILER` and `WM_COMPILER_TYPE` to select them. In some cases, however, it is more convenient to install prebuilt compiler binaries as *system* compilers (e.g., using deb or rpm packages). These compilers are typically distinguished by an additional version suffix (eg, `gcc-11`, `clang-13`, `clang-13.0` etc).

The `WM_COMPILE_CONTROL` environment can be used to add the additional resolution necessary. For example,

```
export WM_COMPILER=Gcc
export WM_COMPILE_CONTROL="version=11"
```

This will add the suffix `-11` to the regular compiler names. Note, that is normally good practice to add some compiler version information into the build name as well. For example,

```
export WM_COMPILER=Clang130
export WM_COMPILE_CONTROL="version=13.0"
```

Be certain to verify that the rules have actually been set as expected:

```
wmake -show-cxx
wmake -show-path-cxx
```

If this change represents your standard default compiler definition, then place the information into the `etc/prefs.sh` file (see the `etc/bashrc` file for some details) and re-source your OpenFOAM environment. If you would like to selectively enable this compiler definition, a common means is to place the same definition information into a user configuration file (for example, `~/.OpenFOAM/clang130`) and then specify that configuration when sourcing your OpenFOAM environment. For example,

Pages 80
Search pages
building
cross compile mingw
coding
git workflow
patterns
dictionary
HashTable
memory
parallel
patterns
precision
registry
selectors
strings
scripts
scripts
style
filenames
style
configuring
Home
icons
info
modules
visualization
packaging
debian
Locations
README
sourceforge
README
suse
Locations
page access code
page build code
page feature requests
precompiled
apptainer
debian
docker
docker old
redhat
suse
windows
running
openfoam selector
shell session
Submitting issues
tuning
upgrade
upgrade
v3 Developer Upgrade G...
v3 User Upgrade Guide
v1606 Developer Upgrad...
v1606 User Upgrade Gui...
v1612 Developer Upgrad...
v1612 User Upgrade Guide
v1612 utility postProcess
v1706 Developer Upgrad...
v1706 User Upgrade Guide
v1712 Developer Upgrad...
v1712 User Upgrade Guide
v1806 Developer Upgrad...
v1806 User Upgrade Gui...
v1812 Developer Upgrad...
v1812 User Upgrade Guide
v1906 Developer Upgrad...
v1906 User Upgrade Gui...
v1912 Developer Upgrad...

```
source /path/to/OpenFOAM-version/etc/bashrc  clang130
```

The `bashrc` will locate and use the configuration file, after which the compiler will be properly selected. Again, to verify everything has actually been set properly:

```
wmake -show-cxx
wmake -show-path-cxx
```

Processor-specific handling

Processor-specific builds are typically handled by creating a new compilation option. For example, to create Broadwell-specific options:

```
    $ cd wmake/rules/linux64Gcc
    $ cp c++Opt c++OptBdw
```

edit this file and then use WM_COMPILE_OPTION=OptBdw in the `prefs.sh` before re-sourcing the OpenFOAM environment.

Since OpenFOAM is purely C++ code, there is no need to apply special processor-specific optimizations for C code (the regular `-O2` optimization is fine) since these components only appear as part of the wmake build toolchain itself.

Comments