Building and Installing Software

Tim Dunn (CU Research Computing / Earth Lab)

General Workflow

The general workflow for building and installing C, C++, Fortran based applications is;

- Transfer package to the system (usually to your /projects/ dir)
- Decompress the package
- Determine required dependencies
- Load Modules
- configure
- make
- make check (if available)
- make install
- make install check (if available)

Why Build Instead of Just Installing

It’s the best fit for the system!

The question is what does ‘fit’ mean?

Summit Compile Nodes

- When building, compiling/building, installing, software on Janus or Summit you **MUST ALWAYS ssh to a compile node.** (Note: you usually start from a login node)

- For Janus
  - > ssh janus-compileX (where X = 1,2,3, or 4)

- For Summit
  - > ssh scompile

Note: You can also compile from a compute node via an interactive session but ***NEVER*** on a login node!

Transfer Package to the System

The first step is to get your desired package onto the system. The most common ways are dependent on the provider and include (but not limited to);

- wget
- git, mercury, subversion
- scp, Globus

Decompressing files

The Linux '***tar***' command can be used to decompress tarball files including those that have been further gzip compressed.

- -x, --extract, --get — extract files from an archive
- -f, --file=ARCHIVE — use archive file
- -v, --verbose — verbosely list files processed
- -z, --gzip, --gunzip, --ungzip — filter the archive through gzip

Example: To decompress a file named foo.tar.gz;

> tar –xzvf foo.tar.gz

Exercise – Untar the examples file

- Make sure your ‘BuildingIsEasy.tar.gz’ file is in your /projects/ directory
  - mv BuildingIsEasy.tar.gz /projects/<*username*>/
- tar –xzvf BuildingIsEasy.tar.gz
- cd BuildingIsEasy/
- ls

Makefiles – Your Friend

There are two main ways of building c/c++, Fortran applications.

- Manually from the command line
- Semi-automatically with ‘Makefile’ scripts

Makefiles – Your Friend

```
CC=gcc
TARGET_EXEC ?= BuildingIsEasy_1
SRC_DIR ?= src
INC_DIR ?= include
OBJ_DIR ?= obj
BIN_DIR ?= bin

SRCS := $(shell find $(SRC_DIR) -name *.c)
INC_DIRS := $(shell find $(INC_DIR) -type d)
INC_FLAGS := $(addprefix -I,$(INC_DIRS))
OBJS := $(SRCS:%=$(BIN_DIR)/%.o)
DEPS := $(OBJS:.o=.d)
CFLAGS ?= $(INC_FLAGS) -c -Wall
-include $(DEPS)

$(BIN_DIR)/$(TARGET_EXEC): $(OBJS)
        $(CC) $(OBJS) -o $@ $(LDFLAGS)

$(BIN_DIR)/%.c.o: %.c
        $(MKDIR_P) $(dir $@)
        $(CC) $(CPPFLAGS) $(CFLAGS) -c $< -o $@

.PHONY: clean

clean:
        $(RM) -r $(BIN_DIR)


MKDIR_P ?= mkdir -p
```

Exercise 2 – Your first build

- If you are not already the BuildingIsEasy/ dir cd to it

- > tar –xzf BuildingIsEasy_1.tar.gz
- > cd BuildingIsEasy_1/
- > make

To run the resultant application:

- cd bin/
- ./BuildingIsEasy_1

Exercise 3 – Build Errors

Now lets try a different flavor

- cd ../
- > tar –xzf BuildingIsEasy_2.tar.gz
- > cd BuildingIsEasy_2/
- > make

What just happened and why?

Make Clean

- make clean – a Makefile defined function to cleanup compiler created files. Used to clear the ‘gunk’ out allowing for the ability for a fresh build after making changes and/or fixing errors.

Exercise 3 – Fix 1

We can modify our make command:

- > make clean
- > make CC=gcc

Exercise 3 – Fix 2

We can modify the Makefile:

- > vi Makefile
- Hit ‘i’ to enter insert mode
- Change the first line to read ‘CC=gcc’
- Save the file and exit by typing: esc:wq!

- > make clean
- > make

Summit Compilers

Compiler Vendor	Language	Standard Compilers	openMPI Compilers	OpenMP Flags
Intel	Fortran	ifort	mpif90	-openmp
	C	icc	mpicc	-openmp
	C++	icpc	mpiCC	-openmp
GNU	Fortran	gfortran	mpfort	-fopenmp
	C	gcc	mpicc	-fopenmp
	C++	g++	mpiCC	-fopenmp
PGI	Fortran	pgfortran, pgf(77, 90, 95)	mpif90	-mp
	C	Pgcc	mpicc	-mp
	C++	pgc++	mpiCC	-mp

Other specialized compilers are also available (eg, Intel MPI, Nvidia)

Modules

- Modules help ensure that your environment is always configured properly (eg loads the proper environment paths and other required environment setups as needed)
- A module exists for each CURC installed package which is available to you.
- Our module system is hierarchal based
- ml is a shorthand way of calling module commands
  - (eg instead of ‘module load intel’ => ‘ml intel’)

Common Module Commands

- > **ml avail** – lists the available modules
- > **ml** – lists the currently loaded modules
- > **ml <*module*>** - loads the module
- > **ml unload <*module*>** - removes the requested module
- > **ml purge** – removes all loaded modules
- > **ml swap <*app1*> <*app2*>** - swaps module 1 for module 2
- > **ml help** – provides general module help
- > **ml help <*module* >** - provides a little info on the module

Exercise 3 – Fix 3

We can use the Intel compiler:

- > ml intel
- > make clean
- > make

To run the resultant application:

- cd bin/
- ./BuildingIsEasy_2

The real world - Configuring

- configure – A special script file which when ran analyzes your current environment and attempts to plug the required flags, dependencies and their paths into a customized for the system Makefile.

- Usage:
- > ./configure <optional desired params>

The final destination --prefix

- --prefix <*destination path*> This is the most popular and often most important parameter you will use. It specifies where you want to save your final installation files too. Usually you install them to your */projects/* directory.

Exercise 4 – Putting it all together

- cd ../../BuildingIsEasy_3/
- or > cd /projects/<*username*>/BuildingIsEasy/BuildingIsEasy_3/
- ./configure –prefix=/projects/<*username*>BuildingIsEasy/BIE_3/
- make
- make install

To run the resultant application:

- cd ../BIE_3/bin/
- ./BuildingIsEasy_3

Python

A Python modules resides on both Janus and Summit.

- For Janus;
  - > ml intel
  - > ml python
  - For Python 3.x > ml python/3.4.3
  - > ml ALL_PYTHON_PKGS
- For Summit;
  - > ml intel
  - > ml python
  - For Python 3.x > ml python/3.5.1

Extending Python

- Python requires the knowledge of where new packages reside and looks for them in the ***PYTHONPATH*** environment variable.
- There's a great many place's you can store this variable, for this exercise we will use '***.my.bashrc***'.

- NOTE: Ideally we would like to just store it in '.profile' as its shell independent but as of the writing of this .profile was not registering on Summit.

Exercise 5: PYTHONPATH

- > cd
- Create a new directory in your /projects/<*username*> directory called ‘MyPythonPkgs’.
  - > mkdir –p /projects/<*username*>/MyPythonPkgs
- >vi .my.bashrc #(NOTE the ‘.’ preceding ‘my’ is required and means it’s a hidden file)
- Insert the following (press ‘i’ to get into insert mode);
- export PYTHONPATH=/projects/<username>/MyPythonPkgs/lib/python
- Save the file and exit by typing: esc:wq!

Python pip

The ‘pip’ installer works wonderfully on both Janus and Summit. It adds the advantage that if you are lacking a dependency it *may attempt* to install it as well.

The trick to installing via pip is making sure its installed to a viable directory and not the one specified in the Python module you loaded (you do not have permissions to save there!)

The normal workflow is;

> pip install --install-option="--prefix=$PREFIX_PATH" package_name

Where installing a package (eg pyFoo) PREFIX_PATH is something like;

/projects/<username>/MyPythonPkgs/pyFoo

Building Python Packages

- A better approach is to build your own version using Pythons '*setup.py*' utility.

- Setup.py is the python equivalent of c/c++ and Fortrans *configure, make,* and *make install,* thus your workflow is basically the same as our earlier exercises.

- The last exercise will have you download the Tornado package (and sadly for Shelley and I who are atmospheric folks) it's a popular Python based web networking framework.
  - (http://www.tornadoweb.org/en/stable/)

Exercise 6 – Build/Install Tornado

- > ml intel
- > ml python
- > cd /projects/<*username*>/MyPythonPkgs
- > wget https://pypi.python.org/packages/1e/7c/ea047f7bbd1ff22a7f69fe55e7561040e3e54d6f31da6267ef9748321f98/tornado-4.4.2.tar.gz
- > tar xvzf tornado-4.4.2.tar.gz
- > cd tornado-4.4.2
- > python setup.py build
- > python setup.py install --home=/projects/<*username*>/MyPythonPkgs

Exercise 6 – Build/Install Tornado

Now to verify it works we can test it with

- > python
- >>> import tornado.web

C, C++, Fortran Summary

- ssh to scompile node
- Obtain package and save in /projects/
- Decompress the package with tar
- Load appropriate compiler and dependency modules
  - If additional dependencies are needed make sure you have the paths to them
- configure
- make
- make check (if available)
- make install
- make install check (if available)
- Enjoy

Python Summary

- Make sure the PYTHONPATH variable is set
- Use either;
- pip install --install-option="--prefix=$PREFIX_PATH" package_name
- or
- python setup.py build
- python setup.py install --home=/projects/<*username*>/MyPythonPkgs

- Alternatively you can install your own Anaconda Python and add your packages with ‘conda install <package name>’
  - (https://www.continuum.io/downloads)

Obtaining Help

- First and most importantly, Google is your friend and your best friend in Google Land is stackoverflow!

- When all else fails (because nothing is ever easy)

  Email us and we will be happy to help you!

rc-help@colorado.edu

Questions?