Spack and SpackDev Build System

James Amundson
LArSoft Tools and Technology Workshop 2017
2017-06-20
What are you doing and why?

• Background
  – Spack and SpackDev are not one-for-one replacements for existing tools

• Spack
• SpackDev
• Status/issues
• Tutorial
Background from the world outside of Fermilab

- Consider a set of toy packages with dependencies

- **corge** package
  - provides *corgegator* executable
    - uses *libcorge*
    - uses *libquux* (from *quux* package)
      - uses *libgarply* (from *garply* package)
System packages

- Install as a system package (Scientific Linux [Ubuntu])
- Standard tools: distribution specific: rpm [dkpg], yum [apt]
- packages: corge, quux and garply (rpm, [dpkg])
- dependency management + package retrieval: yum [apt]
  - `yum install corge` (also installs quux and garply)
- user environment:
  - `corgenator` in `/usr/bin` (in default PATH)
  - Libraries in `/usr/lib` (in default system library path)
  - Nothing more to do; just type `corgenator`
Linux distributions do not have tools for this, in general

Standard tools: trained monkeys

**staging**: download *corge* source package (trained monkey)

**dependency management**: determine dependencies and install (recursive trained monkey)

**configuration**: point *corge* at dependencies
  - e.g., *cmake* or */configure* (trained monkey)

**compile**

**install** in non-system directory

**user environment**:
  - need to add *corge* executable directory to **PATH**
  - need to add *corge* library directory to **LD_LIBRARY_PATH**
  - likewise for dependencies (trained monkey should get started…)

---

6/20/17  James Amundson I Spack and SpackDev
Monkey hate package management

• In case you haven’t figured it out, you are the monkey in this scenario
Fermilab alternative to trained monkeys

- **pullProducts + ups + cetbuildtools + mrb**
- **package retrieval:** pullProducts
- **packaging + dependency management + user environment:** ups
- **configuration + build:** cetbuildtools
  - cetbuildtools depends on ups
- **staging:** mrb

- This is only a rough picture of the roles played by various tools
Why change?

• Many complaints about the build system
  – We won’t get into that here
• Some other issues
  – pullProducts
    • very centralized (users cannot easily set up new distributions)
  – ups
    • Fermilab specific
    • Hard to google (curse you, United Parcel Service)
    • Fermilab has to maintain it
    • Few packages available
    • Non-trivial to create new packages
    • Difficult user interface
      – familiar, though
        • Stockholm syndrome?
    • Leads to very complicated user environment
    • OSX no longer fully supports "LD_LIBRARY_PATH" (i.e., DYLD_LIBRARY_PATH)
    • RPATH is an alternative to "LD_LIBRARY_PATH”
      – eliminates dependency on user environment
        • simplification cannot be overestimated
Spack

- Spack is a package manager designed to handle multiple versions and variants
  - https://spack.io/
  - https://github.com/LLNL/spack
- Spack has an active community of mostly non-HEP, but mostly scientific, developers
Spack is well documented
  – Spack is now on Slack
Spack already contains many packages
  – spack list | wc
    1470  1470  13938 (now: 1550)
Spack has a friendly user interface
  – spack --help
  – spack list --help
Spack packages are easy to create and understand
  – try
    spack edit eigen
Spack gives us RPATH for “free”
Spack features

- Spack has a rigorous model for multiple versions, compilers and variants
  - Values consistency over reuse
  - More on this later
- Spack allows the user to specify which system (or other) packages to use instead of Spack-compiled versions
  - Details go in ~/.spack/<platform>/packages.yaml
    - Can also specify preferences for, e.g., compilers

packages:
  all:
    compiler: [clang@8.1.0-apple, gcc@7.1.0]
  cmake:
    paths:
      cmake@3.8.2: /usr/local/bin/cmake
    buildable: False
More Spack features

• Environment handling is configurable
  – Default is “environment modules”
    • old, Tcl-based
  – Lmod
    • newer, more rigorous, Lua-based
  – Adding ups is an option

• Spack internally uses compiler wrappers to add automatic support for RPATH

• Environment handling is *much simpler* because of the extensive use of RPATH
Local Spack features

- `spack buildcache`
  - Fetches and installs pre-compiled binaries
  - Performs relocations utilizing `patchelf` (Linux) or `install_name_tools` (OSX)
  - Contributions from Benedikt Hegner, Patrick Gartung, JFA
- SpackDev support
  - Minor behind-the-scenes additions
    - Mostly to export information
- Automatic system package discovery for `packages.yaml`
  - *Not yet implemented*
- A stable Spack branch
  - Our needs for stability differ from others in the community
  - No long-term divergence
Spack versions and configurations

```
# Install a particular version by appending @
$ spack install mpileaks@1.1.2

# Specify a compiler (and its version), with %
$ spack install mpileaks@1.1.2 %gcc@4.7.3

# Add special compile-time options by name
$ spack install mpileaks@1.1.2 %gcc@4.7.3 debug=True

# Add special boolean compile-time options with +
$ spack install mpileaks@1.1.2 %gcc@4.7.3 +debug

# Add compiler flags using the conventional names
$ spack install mpileaks@1.1.2 %gcc@4.7.3 cppflags="-O3 -floop-block"

# Cross-compile for a different architecture with arch=
$ spack install mpileaks@1.1.2 arch=bgqos_0
```

Dependencies can be customized

```
# Install mpileaks and link it with specific versions of libelf and libdwarf
$ spack install mpileaks@1.1.2 %gcc@4.7.3 +debug ^libelf@0.8.12 ^libdwarf@20130729+debug
```
Spack dependencies

- `spack spec zlib`

```
mac> spack spec zlib
Input spec
---------------------------------
zlib

Normalized
---------------------------------
zlib

Concretized
---------------------------------
zlib@1.2.11%clang@3.8.1+pic+shared arch=darwin-sierra-x86_64
```

```
|mac> spack spec zlib%gcc
Input spec
---------------------------------
zlib%gcc

Normalized
---------------------------------
zlib%gcc

Concretized
---------------------------------
zlib@1.2.11%gcc@7.1.0+pic+shared arch=darwin-sierra-x86_64
```
Spack spec with dependencies

```shell
mac> spack spec corge
Input spec
----------------------------------------
corge

Normalized
----------------------------------------
corge
  ^cmake@3.0:
  ^quux
    ^garply

Concretized
----------------------------------------
corge@2.0.0%clang@3.8.1 arch=darwin-sierra-x86_64
  ^cmake@3.8.1%clang@3.8.1~doc+ncurses+openssl+ownlibs~qt arch=darwin-sierra-x86_64
    ^ncurses@6.0%clang@3.8.1~symlinks arch=darwin-sierra-x86_64
      ^pkg-config@0.29.2%clang@3.8.1+internal_glib arch=darwin-sierra-x86_64
    ^openssl@1.0.2k%clang@3.8.1 arch=darwin-sierra-x86_64
      ^zlib@1.2.11%clang@3.8.1+pic+shared arch=darwin-sierra-x86_64
  ^quux@2.0.0%clang@3.8.1 arch=darwin-sierra-x86_64
    ^garply@2.0.0%clang@3.8.1 arch=darwin-sierra-x86_64
```
Spack handles packaging, dependency management, package retrieval and package installation
  – It has hooks to user environment tools

SpackDev handles developing packages with dependencies
  – Uses Spack for packaging and dependency management
  – Builds packages just like Spack does
    • configuration
    • RPATH handling
  – SpackDev sets up a build area, then gets out of the way
    • Build with make and/or ninja
    • No environment variables (no ”setup”)
    • Transparent
      – Spack functionality provided by readable shell scripts
• SpackDev is not very complicated

```
mac> spackdev --help
```

**usage:** $spackdev [-h] SUBCOMMAND ...

**positional arguments:**

<table>
<thead>
<tr>
<th>SUBCOMMAND</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getdeps</td>
<td>install missing dependencies of packages in a SpackDev area</td>
</tr>
<tr>
<td>info</td>
<td>describe a $spackdev area</td>
</tr>
<tr>
<td>init</td>
<td>initialize a spackdev area</td>
</tr>
<tr>
<td>stage</td>
<td>stage packages in a spackdev area</td>
</tr>
</tbody>
</table>

**optional arguments:**

- `-h, --help` show this help message and exit

• SpackDev handles dependency installation, staging, configuration and build area creation
  – not quite in that order
SpackDev, still cont.

- SpackDev will build intermediate dependencies

```
|mac> spackdev init --no-stage --no-dependencies corge garply
==> requested packages: corge garply
==> additional inter-dependent packages: quux
==> creating wrapper scripts
==> creating build area
```
Status

• SpackDev is behind schedule
  – Planned to give a full demo here
  – Only going to explore spack functionality
• Delays are because of me

Learn from our mistakes

• Project managers
  – Do not rely on your department head to accomplish a long-term project
• Department heads
  – Do not commit yourself to a long-term development project
• Restructuring SpackDev development to remove me as a stumbling block
What needs to be done

• Binary package distribution (buildcache) is in place, but needs refinement
  – Cannot query what is available
  – Likely to get wrong variant -> binary package not found

• Automated support for system packages is necessary
  – We have a plan, but not an implementation
  – Without it “spack install lmod” will install 36 dependent packages, including perl, tar (!) and git.

• Full support for building art and LArSoft stacks without UPS is underway
  – Needs completion and testing

• Need to refine the user experience
Spackdev-bootstrap

- [https://github.com/amundson/spackdev-bootstrap](https://github.com/amundson/spackdev-bootstrap)

```bash
  git clone https://github.com/amundson/spackdev-bootstrap.git
  cd spackdev-bootstrap
  ./bootstrap-spackdev
```

- Checks out Spack
- Checks out SpackDev
- Creates setup script
  - Adds spack to path
  - Adds spack shell function
    - optional for spack
  - Adds spackdev to path
Exploring Spack

- Spack commands are like git commands
  
  - `spack command [arguments]`

- Everything accepts a help argument
  
  - `spack -help`
  
  - `spack list -help`

- Some spack commands (try with `--help` first)
  
  - `spack find`
  
  - `spack list`
  
  - `spack compiler list`
  
  - `spack edit <package>`

- The Spack documentation site contains a full tutorial
  
Things to try

- `spack list`
- `spack find`
- `spack install zlib`
- `spack find` (after installing zlib)
- `mkdir foo; cd foo; spackdev init --no-deps garply`
- `mkdir bar; cd bar; spackdev init --no-deps --no-stage garply corge`
- `spack install corge`
  - Will build cmake and dependents – expect to wait over 10 minutes