Spack Work Items for HEP

Chris Green greenc@fnal.gov
Lynn Garren garren@fnal.gov
Patrick Gartung gartung@fnal.gov

version 0.6 (2019-12-13)

Contents

1   Introduction

While working on the Spack-based build, development, and release system for HEP, we have come across many instances big and small where Spack core or its recipes require enhancements to allow us to meet our project requirements. In some cases these are showstoppers, such as support for multiple releases and layered (central vs local) installations; in others, they improve an aspect of Spack operation without being essential, such as providing repo-specific package base classes.

This document is an attempt to be a descriptive list of these work items (the most impactful and necessary core items are listed first), along with (where appropriate) ideas for implementation, statement of importance, and current status.

2   High-Impact Core Enhancements

2.1   Support for external Spack commands

2.1.1   ~~Basic Support for external Spack commands~~

In order to support the efficient operation of Spackdev, allowing the use of Spack functions and structures internally, `spack dev` should be a spack command. For purposes of ownership and ease of maintenance, this would ideally be an external command, analogous to (*e.g.* `git flow` as an external `git` command).

3

Status PR 8612 from Massimiliano Culpo implemented basic support for external Spack commands. It is now integral to the operation of SpackDev and was merged into the FNAL fork on 2018-09-19. The PR was merged upstream on 2019-03-28 with some changes from the original PR submission as merged into the FNAL fork, which was subsequently reconciled with upstream.

2.1.2   Enhancement of Spack subcommand facilities

Spack's existing handling of subcommands should be generalized to work for external commands and at any level.

Status A git branch consisting solely of our proposed enhancement of the subcommand facilities was submitted as PR 11145. Comments were received and incorporated. Subsequently, a competing PR 11209 was submitted by the original author of PR 8612 purporting to address one of the issues addressed by PR 11145. This PR was incorporated into PR 11145. Following issues raised by reviewers, discussions resulted in the creation of PR 13635 containing the less controversial aspects of PR 11145.

2.1.3   Support for flake8 checks of external Spack commands

A simple way to check external Spack command code for rule-correctness is strongly desired.

Status While PR 8612 supports testing of external Spack commands, it appears not to support intuitive coding rule enforcement for same. An enhancement request should be discussed with the author of PR 8612.

2.1.4   Support for external commands and support code in external package repos

The ability to find external commands in external package repos. This is related to a Spack Core Enhancement described elsewhere herein.

Status The generalization of Spack's subcommand facilities will be the subject of a future PR following the eventual merge of PR 13635.

2.2 ~~The ability to layer Spack installations in a hierarchy~~

In order to be able to combine locally-built and centrally-installed packages, it becomes necessary to be able to refer to packages in a different Spack installation from the current one.

Status Originally conceived by Jim Amundson (as "Spack Chains") and entered as PR 5014, and subsequently resubmitted against a more recent version of Spack develompent by Patrick Gartung as PR 8545, this feature has been taken up by Peter Scheibel—a Spack principal— as PR 8772. This PR was merged on 2019-03-27 and a subsequent PR 11152 adding documentation was merged 2019-04-10.

2.3 A persistent relationship between a built package and the options used to build it

In order to support the use of a given Spack installation to support multiple installed software distributions over a significant period of time, it is necessary to be able to identify an installed or cached binary package such that it remains possible to use it even after updates to the recipe(s) have caused a change to the already-built package's calculated hash.

It is the nature of HEP software installations to have a relatively static central installation of software packages and distribution releases against which higher-level packages are developed by users. It must be possible to install centrally newer distribution releases while not compromising the ability of users to build and develop their own software against older ones without unnecessary rebuilds of lower level packages due to hash changes resulting from updated recipes.

Status The recent Spack Environments functionality may assist with this (as may Spack Chains, described elsewhere in this document), but this remains a thorny problem that will require discussion and exploration before a solution is implemented. The subject has been discussed with Spack principals such as Todd Gamblin and Peter Scheibel and we have been assured that this feature is compatible with the philosophy and direction of Spack, that it has other

desirous parties than us, and that it would be reasonable to expect something to look at that should do the job by the end of September 2019.

Recent news from SC-19 indicates that the new concretizer—due March 2020—is expected to satisfy this item.

2.4   Support for compiler-agnostic packages

The HEP community has several applications, distributions of which must be accompanied by substantial data. Examples are particle interaction descriptions and cross-section table. Examples outside our field might include star catalogs and other astrophysics data. Information of this nature can be quite voluminous (≥ 5~GiB). We would wish to avoid unnecessary duplication for multiple compilers and versions by having just one compiler-agnostic package containing the data.

In addition, pure C and pure interpreted-language packages such as `fftw` or `pyyaml`—or those which combine same—are similarly independent of compiler and version thereof (or perhaps require only a generic compiler runtime such as `libgcc_s.o`). A way to avoid unnecessary duplication of these packages also would be desirable, although the win here would be less in space and more in CPU time for builds.

Status   The subject has been discussed with Spack principals such as Todd Gamblin and Peter Scheibel, including at the telecon 2019-04-11, but no formal enhancement request has been made. Something providing this functionality is expected to be part of the upcoming concretization improvements, but it should be noted that merely allowing the compiler as a "normal" dependency is insufficient to provide the needed flexibility.

Recent news from SC-19 regarding the concretizer indicates that adding the compiler as a virtual dependency is expected to satisfy this item.

2.5   ~~No-source packages~~

In our existing system, we make significant use of no-source, or "umbrella" packages, whose only function is to facilitate the build or setup of a particular collection of packages.

Status [PR 11981](#) providing a `BundlePackage` class was merged 2019-08-22. Related PRs include [12536](#) (merged 2019-08-23), [12804](#) merged (2019-09-17), and [12882](#), (merged 2019-09-23).

2.6 ~~Base build location on stage rather than expanded source location~~

The definition of the CMake build area should be changed from being relative to the staged source directory to being relative to the stage directory.

We have use cases where a recipe needs to have information about the build directory in the environment set by `setup_environment()` (*e.g.* if tests need to have a directory in `PATH`). Specifically, staging happens *after* the call to `setup_environment()`, and if the stage area does not exist, then `spec.stage.source_path` returns, `None`.

Status [PR 8431](#) was submitted 2018-06-08 and was moribund for several months. This PR was eventually accepted and merged upstream on 2019-03-26.

2.7 ~~`self.stage.source_path` should be usable before it exists.~~

There are circumstances where it is desirable to *e.g.* set an environment variable in `setup_environment()` based on `self.stage.source_path`. Currently this is not possible as its existence is enforced and `setup_environment()` is executed prior to any staging of the source.

Status PRs [11528](#) (merged 2019-06-06) and [11662](#) (merged 2019-06-07) address this issue.

2.8 Masking of installed packages and dependents

A user may wish to develop packages that may already be installed (centrally, perhaps). We require a mechanism to hide these already-installed packages and their dependents from the concretization system

Status

This feature is not currently required for the Minimum Viable Product (MVP), but will be essential in the finished system. It is possible that environments or views may satisfy this requirement, although that remains to be investigated.

3 Other Spack Core Enhancements

3.1 The ability to inherit from package-repo-specific package base classes

Fermilab has a large number of experiment-related packages using a common CMake-based system called `cetmodules`. Recipes for these packages share a significant amount of boilerplate which could profitably be part of a package base class situated in the same package repo.

Status This feature has been discussed informally with Spack principals. An enhancement of the extension command system to allow non-command extensions to Spack functionality and allowing a package repo to operate as such an extension would satisfy this request.

3.2 ~~New environment manipulation functions for system path deprioritization and duplicate removal~~

In order to prevent contamination of `PATH`-like variables with system paths due to `packages.yaml` entries, several new functions provide facilities for `PATH`-like variable manipulation.

Status [PR 8476](#) was submitted 2018-06-14 and was moribund for a long time. It has been updated against upstream and discussed with Spack principals and further information has been added to the PR. After changes were made to accommodate upstream developments, it was merged on 2019-05-06.

[PR 11434](#) was submitted 2019-05-13 to allow the production of source-able text and pickled environment files via the command line (`spack build-env`

`--pickle <file>`, `spack build-env --dump <file>`). It was merged 2019-05-16.

3.3 Configurable equivalency between operating systems

Due to a change in how Scientific Linux installations identify themselves between SL7.5 and SL7.6, there needs to be a configurable equivalency between compatible operating systems *e.g.* in `config.yaml`:

```
equivalent-os:
  rhel6:
    - scientific6
    - centos6
  rhel7:
    - scientific7
    - centos7
```

Status   We have a hard-wired equivalency between `rhel7` and `scientific7` in our local code.

A concept for the solution exists (see above). Analysis is needed to determine feasibility and consequences for (*e.g.*) hash values, buildcache packages, *etc.*

This issue has been discussed with Spack principals: OS string conversion, and updating the metadata system to hadle equivalency. This issue also has relevance to CPU architecture (Haswell, *etc.*). This request needs to be re-evaluated in the context of the microarchitecture target feature, and the upcoming concretizer.

3.4 Updates to the version comparison system

Status   All the below-menioned core enhancements are at the conceptual stage only. No implementation has been attempted, but it is likely that they are perfectly feasibly implementable by a contributor rather than a core expert. This could be entered instead as an enhancement request, but the likelihood of a core developer taking it on is somewhat low. The truncated-version-ordering issue must be addressed with care as there are use cases for both behaviors, and they should be distinguishable and selectable as appropriate.

3.4.1 Support correct ordering of alpha, beta, pre, rc and patch versions

While it is not usual to need to refer to non-mainstream versions in a recipe, it is certainly not beyond imagining. Several packages have had long release candidate cycles, for instance, and a build of one might be necessary to address an observed issue prior to the final release becoming public. When non-mainstream versions are used in ranges, the version comparison needs to be aware of the relative ordering—release candidates following "pre" versions, but preceding their corresponding released versions, for example.

3.4.2 Support for half-open and closed intervals for version ranges

e.g. describing something which is valid for $7.0 \leq$ `version` < 8.0. For purely numeric versions, the usual workaround of `when=@7.0:7.999` suffices, but for something which is valid also for (*e.g.* release candidates), it becomes more desirable to be able to specify "everything before . . . " Example nomenclature:

```
# No support before ...
conflicts('cxxstd=17', when='@:<1.27.0')
# Blacklist range.
variant('bad_idea', default=True, when='@0.5.0:<1.0.0')
# Last known good.
conflicts('went_bad_after', when='@1.5.7>:')
# Ignore 2.0.0-alpha, pre, etc.
conflicts('Shaky prerelease', when='@1.8.12>:<@2.0.0'
```

3.4.3 Correct ordering between versions `X` and `X.Y`

There appears to be a problem ordering versions with different levels of specification. For instance, in `intel-tbb`, the version version of any year's releases is the unadorned year (`2019`, say), with `2019U1` being the next. These have to be encoded currently as `2019.0` and `2019.1`, because `2019` does not appear to compare less-than `2019.1`. Following discussion with Peter Scheibel, it appears that this is intentional behavior in the numeric case, and additionally applies to non-numeric versions where one is a front-anchored substring of another (*e.g.* MVP1 vs MVP1a).

3.4.4 `version()` should recognize filename extensions from `url_from_version()` including `tbz2`

Several of our recipes are of the form where the following `url_from_version()` is appropriate:

```python
def url_for_version(self, version):
    url = 'http://cdcvs.fnal.gov/cgi-bin/git_archive.cgi/'\
            'cvs/projects/{0}.v{1}.tbz2'
    return url.format(self.name, version.underscored)
```

Unfortunately, for every call to `version()`, it is necessary to specify the extension explicitly:

```python
version('2.3.0',
        sha256='4b6a29443b631957ca2a7712b5c57762'
        '0c6543e542ee9c77d246cef1e10f7324',
        extension='tbz2')
```

The system should be trained to understand `tbz2` as a valid extension when returned as a URL by `url_for_version()`.

3.5 Compact variant propagation via `depends_on()`

There is a need for propagation of variants to dependencies generally, rather than specifically. For instance, in order to pass the value of the `cxxstd` variant down a dependency tree, currently one must do:

```python
depends_on('boost cxxstd=default', when='cxxstd=default')
depends_on('boost cxxstd=98', when='cxxstd=98')
...
```

One could imagine instead a more compact syntax, something like:

```python
depends_on('boost', propagate_variants=['cxxstd', 'python'])
```

If `cxxstd` is not specified, then nothing will be propagated (if the variant exists in the dependency it will be defaulted). Otherwise, the value of `cxxstd` in the current recipe will be used.

As a further extension, using `recursive_variants` instead of `propagate_variants` would cause the specified variants to be propagated to lower-level dependen-

cies *even if intermediate dependencies did not have the variant(s) in question defined.*

Status These enhancements are at the conceptual stage only.

The new concretizer will support the specification of language standard as a virtual dependency. This should obviate the need for a `cxxstd` variant. It is possible that is desirable to propagate other variants in this way.

3.6   Improvements to the variants system

Several packages have extensive feature options, which evolve over time. Features come and go, and depend upon each other. An expanded variant specification syntax is desired to be able to specify these variants and the relationships between them. The archetype for this use case would be the `boost` recipe. Some ideas:

3.6.1   `variant()` grows a `when` clause

The `when` clause shall specify when the variant is defined. It shall be valid therein to refer to versions and to any other variants that have already been defined.

3.6.2   New function `variant.depends_on()`

This function, having a `when` clause as defined for `variant()`, above, shall specify upon what other variant(s) `self` shall depend when the `when` clause is valid. It shall return the variant object allowing for chaining, *e.g.*

```
variant('context', when='@1.61.00:').\
  depends_on('thread', 'chrono', 'system', 'date_time')
variant('fiber', when='@1.62.00:').depends_on('context').\
  depends_on('thread', 'chrono', 'system', 'date_time',
             when='@:1.68.99')
variant('signals', when='@1.29.00:1.68.99')
```

Status These enhancements are at the conceptual stage only.

3.7 Install only specified packages from YAML specifications.

Spack currently provides the ability to install packages specified by a YAML spec file in addition to those specified on the command line. Spack should provide the ability to install the *intersection* of these two classes of package specification rather than their union. This improves the ability to easily (re-)install subsets of a large and complex spec tree.

Status These enhancements are at the conceptual stage only.

3.8 Writing of the YAML spec file at the start of the build process rather than solely at install time.

To improve the development cycle time for packages that are part of complex dependency hierarchies, the YAML spec tree should be written in the same directory as `spack-build.{env,out}` to allow error-free repeats of install attempts.

Status These enhancements are at the conceptual stage only.

4 Recipe Enhancements

4.1 ~~Client-only `mysql`-ish packages~~

The existing `mariadb` package is GPL and therefore not usable by HEP community in a linking context. An LGPL client-only package provided by `mariadb` exists. In addition, we are constrained at Fermilab to avoid distributing servers wherever possible due to the obligation to follow onorous update policies.

Status PR7729 was submitted 2018-04-11 and became moribund after being nominally approved. It was eventually merged 2019-03-15. A subsequent issue 11226 was addressed and the fix merged upstream as PR 11237.

4.2 Client-only `postgresql` build

As above, server packages are problematic. A build of `postgresql` without the server component is desired.

Status PR 7728 was submitted 2018-04-11 and became enmired after questions were raised over the appropriateness of virtual packages and separate implementations rather than a (say) `~server` variant. In addition, the PR included a recipe for `py-psycopg2` which was extracted and submitted as a separate PR 9926 by a third party 2018-11-22 and merged 2018-11-24. PR 7728 was closed 2019-03-13 and a revised PR 10877 was submitted with a `client_only` option to change the way that `postgresql` is built rather than provide a separate client-only product. PR 10877 was merged 2019-03-25.

4.3 Updated `root` recipe

Our local `root` recipe had significant changes with respect to the upstream version including more versatility in configuration and use of our local forks of `postgresql-client` and `mariadb-client`. Significant changes to the upstream version were submitted by CERN as PR 8428 2018-06-08 and accepted 2019-02-08 with which our local version is incompatible.

Status After resolving a problem building more recent versions of ROOT with PR 11129, our local recipe was eventually reconciled with upstream and submitted as PR 11215 with a companion PR 11214 with necessary updates to the recipe for FTGL. PR 11214 was merged 2019-04-23, and PR 11215 was merged 2019-04-24.

4.4 Propagate C++ attributes through dependencies of C++-using packages

Several C++ recipes have a `cxxstd` variant, and depend on other C++ packages. The variant should be propagated in-recipe rather than as part of a `spack install` command, as currently.

Status A survey needs to be carried out and the appropriate changes to recipes made and propagated upstream.

The new concretizer will support the specification of language standard as a virtual dependency. This should obviate the need for a `cxxstd` variant.

5 Requirements on the New Concretizer

This section gathers requirements that could be expected to be satisfied or enabled by the new concretizer.

5.1 A persistent relationship between a built package and the options used to build it

See *A persistent relationship between a built package. . . .*

5.2 Support for compiler-agnostic packages

See *Support for compiler-agnostic packages*.

5.3 Configurable equivalency between operating systems

See *Configurable equivalency between operating systems*.

5.4 Compact variant propagation via `depends_on()`

See *Compact variant propagation via `depends_on()`*.

5.5 Improvements to the variant system

See *Improvements to the variant system*.

5.6 The ability to co-concretize packages with or without common dependencies

The API facility originally provided by PR 11158 should be preserved, and should be the default behavior when multiple top-level packages are presented to `spack install`, `spack spec`, *etc.*

5.7 The ability to build only part of a fully-specified dependency tree

The API facility currently provided by `Spec.package.do_install()` (when the `Spec` object is part of a fully-concretized tree) should be preserved, and should be extended to the command line per *Install only specified packages from YAML specifications*.

5.8 The ability to identify intermediate packages given a list

The search algorithm on a `Spec` tree as represented by get_additional(requested, specs) identifies packages in the given `Spec` tree `specs` not already named in the list `requested` that are both (possibly-indirect) dependencies of at least one package named in `requested` and dependents of at least one other so-named package. This algorithm should continue to be possible with Spack as based on the new concretizer.

6 Version History

Version 0.1 (2019-02-25)

First released version.

Version 0.2 (2019-03-19)

- Add table of contents.
- Expand the section on updates to the version comparison system with examples of closed and half-open interval version ranges, and with a new section on the ordering issue between versions `X` and `X.Y`.
- New section, *Compact variant propagation via `depends_on()`*.
- New section, *Improvements to the variants system*.
- Expand section on *Support for external Spack commands* with contribution from Ben Morgan.
- Updates to status for *Client-only `mysql`-ish packages*, and *Client-only `postgresql` build*.
- Add section number for *Client-only `postgresql` build*.

- Add version history.

Version 0.3 (2019-04-30)

- Minor re-word of *Introduction* for clarity.
- Updates to status for:
    - *Support for external Spack commands* (and fix a typo in the title).
    - *The ability to layer Spack installations. . . .*
    - *A persistent relationship between a built package. . . .*
    - *Support for compiler-agnostic packages*.
    - *Base build location on stage. . . .*
    - *New environment manipulation functions. . . .*
    - *Correct ordering between versions `X` and `X.Y`* (and expand description).
    - *Client-only `mysql`-ish packages*.
    - *Client-only `postgresql` build*.
    - *Updated `root` recipe*.
- Add status section for *Improvements to the variants system*.
- Consolidate status updates into a single coherent statement of status where appropriate.
- New section, *`self.stage.source_path` should be usable before it exists*.
- Improve code snippet sections for line overruns and Python-style formatting.

Version 0.4 (2019-05-16)

- New section *Masking of installed packages and dependents*.
- Updates to status for:
    - *New environment manipulation functions. . . .*
    - *No-source packages*.
    - *Configurable equivalency between operating systems*.
- Strikethrough notation is used in item titles to indicate resolution.

Version 0.5 (2019-06-11)

- New section *Install only specified packages from YAML specifications*.

- New section *Writing of the YAML spec file at the start of the build process. . . .*
- Update to status for `self.stage.source_path` *should be usable before it exists*.

Version 0.6 (2019-12-13)

- Add markdown variables controlling document characteristics: class, font size, language, section numbering, table of contents and depth thereof.
- Demote status sections to level 4 (no numbering), split status sections up to be more specific when appropriate.
- Updates to status for:
  - *Enhancement of Spack subcommand facilities*.
  - *Support for compiler-agnostic packages*.
  - *No-source packages* (resolved).
  - `self.stage.source_path` *should be usable before it exists* (resolved).
  - *New environment manipulation functions. . .* (resolved).
  - *The ability to inherit from package-repo-specific package base classes*.
  - *Support for external commands and support code in external package repos*.
  - *A persistent relationship between a built package. . . .*
  - *Configurable equivalency between operating systems*.
  - *Compact variant propagation via* `depends_on()`.
  - *Propagate C++ attributes through dependencies of C++-using packages*.
- New section *Requirements on the New Concretizer* gathering requirements related to the concretizer.