Multi-package development at Fermilab with Spack

Kyle J. Knoepfel

22 October 2024

CHEP 2024

Spack adoption at Fermilab

Spack is the **s**upercomputing **pack**age manager.

- It has gained widespread traction across the HPC community and within HEP
- Spack is now part of the High-Performance Software Foundation (https://hpsf.io)

Spack adoption at Fermilab

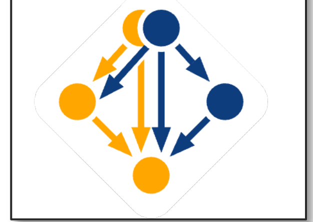
Spack is the **s**upercomputing **pack**age manager.

- It has gained widespread traction across the HPC community and within HEP
- Spack is now part of the High-Performance Software Foundation (https://hpsf.io)

Fermilab decided to switch to Spack for several reasons:

- Constraints on effort to maintain Fermilab-specific package manager
- Take advantage of technology provided by the broader computing community
- Engage and influence the broader computing community

Spack adoption at Fermilab



Spack is the **s**upercomputing **pack**age manager.

- It has gained widespread traction across the HPC community and within HEP
- Spack is now part of the High-Performance Software Foundation (https://hpsf.io)

Fermilab decided to switch to Spack for several reasons:

- Constraints on effort to maintain Fermilab-specific package manager
- Take advantage of technology provided by the broader computing community
- Engage and influence the broader computing community

Fermilab progress in the last few years:

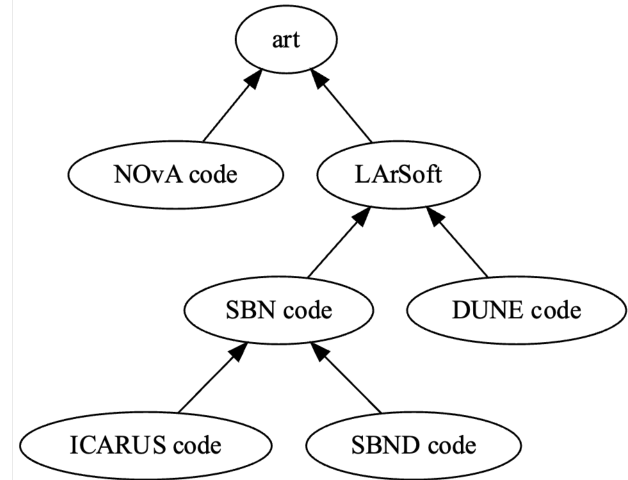
- Proof-of-principle installation of all offline code for DUNE, Mu2e, etc.
- Joined Spack's technical steering committee
- Establishing process for layered releases/environments of Fermilab-supported software
- Created solution for building CMake packages together in a Spack context

Developing multiple repositories together

Many experiments at Fermilab share software.

It is common for an experiment to develop their own code at the same time as adjusting a piece of shared software.

Simplified dependency graph



Fermilab

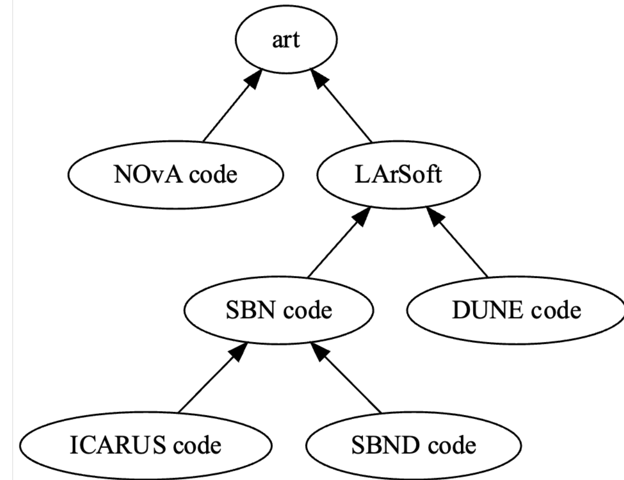
Developing multiple repositories together

Many experiments at Fermilab share software.

It is common for an experiment to develop their own code at the same time as adjusting a piece of shared software.

Most of these bodies of software are CMake packages, which can often be built together as a larger CMake project.

For the past decade, this has been done at Fermilab with the **m**ulti-**r**epository **b**uild (MRB) system.



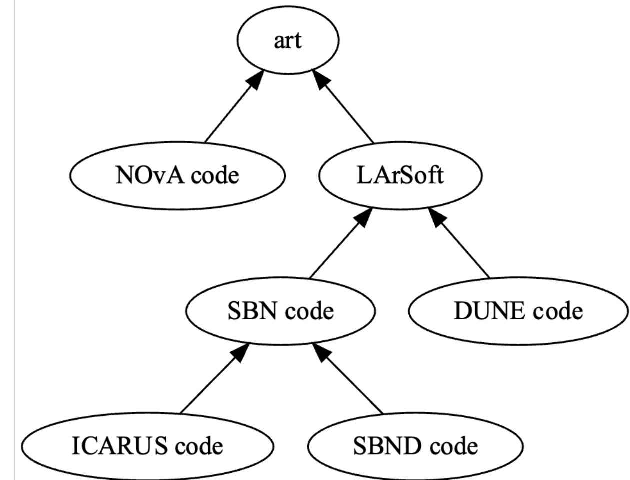
Developing multiple repositories together

Many experiments at Fermilab share software.

It is common for an experiment to develop their own code at the same time as adjusting a piece of shared software.

Most of these bodies of software are CMake packages, which can often be built together as a larger CMake project.

For the past decade, this has been done at Fermilab with the **m**ulti-**r**epository **b**uild (MRB) system.



As successful as MRB has been in building repositories in concert, it relies heavily on Fermilab's home-grown package management system (UPS).

Fermilab is pursuing a Spack-based approach for developing multiple repositories.

Code development using Spack

Fermilab has tried different approaches for replacing MRB:

1. FNAL-created `spack dev` extension

   LArSoft minimum viable product released in 2019; experiments were not ready to explore it.

Fermilab

Code development using Spack

Fermilab has tried different approaches for replacing MRB:

1. FNAL-created `spack dev` extension

   LArSoft minimum viable product released in 2019; experiments were not ready to explore it.

2. Spack-provided feature `spack develop`

   Well-integrated with Spack installations

   Supports development of any Spack package

   More Spack expertise required of users, and substantial inefficiencies in incremental builds

Code development using Spack

Fermilab has tried different approaches for replacing MRB:

1. FNAL-created `spack dev` extension

   LArSoft minimum viable product released in 2019; experiments were not ready to explore it.

2. Spack-provided feature `spack develop`

   Well-integrated with Spack installations

   Supports development of any Spack package

   More Spack expertise required of users, and substantial inefficiencies in incremental builds

3. FNAL-created `spack mpd` (MPD) extension https://github.com/FNALssi/spack-mpd

   Allows users to develop CMake packages **in concert** with Spack-provided software

   Tailored for iterative algorithm development

   Try to give a familiar feel to MRB…but *not too* familiar

Desired features of MPD

Spack interactions

- Minimize user’s required knowledge of Spack
- Take advantage of packages installed in upstream Spack instances/environments
- Directly support the installation of dependencies
  - This was not feasible with UPS
  - Must avoid rebuilding dependencies with existing installations

Usability

- Easy to setup an MPD development session
- Easy to switch between my MPD projects
  - Avoid reliance on environment variables
- Easy to list which MPD projects are available to you

Spack MPD commands

```
$ spack mpd -h
usage: spack mpd [-hV] SUBCOMMAND ...

develop multiple packages using Spack for external software

positional arguments:
  SUBCOMMAND
    build (b)             build repositories
    clear                 clear selected MPD project
    git-clone (g, clone)
                          clone git repositories
    deploy (d)            deploy developed packages
    init                  initialize MPD on this system
    install (i)           install built repositories
    list (ls)             list MPD projects
    new-project (n, newDev)
                          create MPD development area
    refresh               refresh project
    rm-project (rm)       remove MPD project
    select                select MPD project
    status                current MPD status
    test (t)              build and run tests
    zap (z)               delete everything in your build and/or install areas

optional arguments:
  -V, --version         print MPD version (0.1.0) and exit
  -h, --help            show this help message and exit
```


Fermilab

Spack MPD commands

Project commands
Commands that establish a user environment for developing a given project.

```
$ spack mpd -h
usage: spack mpd [-hV] SUBCOMMAND ...

develop multiple packages using Spack for external software

positional arguments:
  SUBCOMMAND
    build (b)           build repositories
    clear               clear selected MPD project
    git-clone (g, clone)
                        clone git repositories
    deploy (d)          deploy developed packages
    init                initialize MPD on this system
    install (i)         install built repositories
    list (ls)           list MPD projects
    new-project (n, newDev)
                        create MPD development area
    refresh             refresh project
    rm-project (rm)     remove MPD project
    select              select MPD project
    status              current MPD status
    test (t)            build and run tests
    zap (z)             delete everything in your build and/or install areas

optional arguments:
  -V, --version         print MPD version (0.1.0) and exit
  -h, --help            show this help message and exit
```


Fermilab

Spack MPD commands

Project commands
Commands that establish a user environment for developing a given project.

Development commands
Standard commands for development after a user environment has been set up for a given project.

```
$ spack mpd -h
usage: spack mpd [-hV] SUBCOMMAND ...

develop multiple packages using Spack for external software

positional arguments:
  SUBCOMMAND
    build (b)            build repositories
    clear                clear selected MPD project
    git-clone (g, clone)
                         clone git repositories
    deploy (d)           deploy developed packages
    init                 initialize MPD on this system
    install (i)          install built repositories
    list (ls)            list MPD projects
    new-project (n, newDev)
                         create MPD development area
    refresh              refresh project
    rm-project (rm)      remove MPD project
    select               select MPD project
    status               current MPD status
    test (t)             build and run tests
    zap (z)              delete everything in your build and/or install areas

optional arguments:
  -V, --version          print MPD version (0.1.0) and exit
  -h, --help             show this help message and exit
```

Spack MPD commands

Project commands
Commands that establish a user environment for developing a given project.

Development commands
Standard commands for development after a user environment has been set up for a given project.

Usability
Helper commands to let you know what you can do and what you're doing.

```
$ spack mpd -h
usage: spack mpd [-hV] SUBCOMMAND ...

develop multiple packages using Spack for external software

positional arguments:
  SUBCOMMAND
    build (b)            build repositories
    clear                clear selected MPD project
    git-clone (g, clone)
                         clone git repositories
    deploy (d)           deploy developed packages
    init                 initialize MPD on this system
    install (i)          install built repositories
    list (ls)            list MPD projects
    new-project (n, newDev)
                         create MPD development area
    refresh              refresh project
    rm-project (rm)      remove MPD project
    select               select MPD project
    status               current MPD status
    test (t)             build and run tests
    zap (z)              delete everything in your build and/or install areas

optional arguments:
  -V, --version          print MPD version (0.1.0) and exit
  -h, --help             show this help message and exit
```

Spack MPD commands

Project commands
Commands that establish a user environment for developing a given project.

Development commands
Standard commands for development after a user environment has been set up for a given project.

Usability
Helper commands to let you know what you can do and what you're doing.

Initialization
Once per Spack instance

```
$ spack mpd -h
usage: spack mpd [-hV] SUBCOMMAND ...

develop multiple packages using Spack for external software

positional arguments:
  SUBCOMMAND
    build (b)           build repositories
    clear               clear selected MPD project
    git-clone (g, clone)
                        clone git repositories
    deploy (d)          deploy developed packages
    init                initialize MPD on this system
    install (i)         install built repositories
    list (ls)           list MPD projects
    new-project (n, newDev)
                        create MPD development area
    refresh             refresh project
    rm-project (rm)     remove MPD project
    select              select MPD project
    status              current MPD status
    test (t)            build and run tests
    zap (z)             delete everything in your build and/or install areas

optional arguments:
  -V, --version         print MPD version (0.1.0) and exit
  -h, --help            show this help message and exit
```


Fermilab

Development workflow

Create new project

```
$ spack mpd new-project --name my-art-devel -T my-art-devel -E gcc-14-1 cxxstd=20 %gcc@14

==> Creating project: my-art-devel

Using build area: /scratch/knoepfel/my-art-devel/build
Using local area: /scratch/knoepfel/my-art-devel/local
Using sources area: /scratch/knoepfel/my-art-devel/srcs

==> You can clone repositories for development by invoking

  spack mpd git-clone --suite <suite name>

  (or type 'spack mpd git-clone --help' for more options)
```


Fermilab

Development workflow

Create new project

```
$ spack mpd new-project --name my-art-devel -T my-art-devel -E gcc-14-1 cxxstd=20 %gcc@14

==> Creating project: my-art-devel

Using build area: /scratch/knoepfel/my-art-devel/build
Using local area: /scratch/knoepfel/my-art-devel/local
Using sources area: /scratch/knoepfel/my-art-devel/srcs

==> You can clone repositories for development by invoking

  spack mpd git-clone --suite <suite name>

  (or type 'spack mpd git-clone --help' for more options)
```

Clone repositories

```
$ spack mpd git-clone --fork cetlib cetlib-except hep-concurrency

==> Cloning and forking:

  cetlib .................. done    (cloned, added fork knoepfel/cetlib)
  cetlib-except ........... done    (cloned, created fork knoepfel/cetlib-except)
  hep-concurrency ......... done    (cloned, created fork knoepfel/hep-concurrency)

==> You may now invoke:

  spack mpd refresh
```

Development workflow

Refresh project

Concretization *is when Spack looks for a set of package specifications that satisfy dependency requirements.*

```
$ spack mpd refresh

==> Refreshing project: my-art-devel

...

==> Concretizing project (this may take a few minutes)
==> Environment my-art-devel has been created
==> Updating view at /scratch/knoepfel/spack/var/.../my-art-devel/.spack-env/view
==> Concretization complete
```


Fermilab

Development workflow

Refresh project

Concretization *is when Spack looks for a set of package specifications that satisfy dependency requirements.*

The ***installation*** *step installs only the* ***dependencies*** *of the packages being developed.*

```
$ spack mpd refresh

==> Refreshing project: my-art-devel

...

==> Concretizing project (this may take a few minutes)
==> Environment my-art-devel has been created
==> Updating view at /scratch/knoepfel/spack/var/.../my-art-devel/.spack-env/view
==> Concretization complete

==> Ready to install MPD project my-art-devel

==> Would you like to continue with installation? [Y/n]
==> Specify number of cores to use (default is 12)
==> Installing my-art-devel
```


Fermilab

Development workflow

Refresh project

Concretization *is when Spack looks for a set of package specifications that satisfy dependency requirements.*

The ***installation*** *step installs only the* ***dependencies*** *of the packages being developed.*

```
$ spack mpd refresh

==> Refreshing project: my-art-devel

...

==> Concretizing project (this may take a few minutes)
==> Environment my-art-devel has been created
==> Updating view at /scratch/knoepfel/spack/var/.../my-art-devel/.spack-env/view
==> Concretization complete

==> Ready to install MPD project my-art-devel

==> Would you like to continue with installation? [Y/n]
==> Specify number of cores to use (default is 12)
==> Installing my-art-devel
[+] /usr (external glibc-2.34-hjl43avhawltutkgujn2ns3577kjowlq)

...

[+] /scratch/knoepfel/spack/.../intel-tbb-2021.9.0-gtkaoizm5i4m6goy7rptg7v3i5q2jrg7

==> MPD project my-art-devel has been installed.  To load it, invoke:

  spack env activate my-art-devel
```

Development workflow

Activate environment
Build

```
$ spack env activate my-art-devel
$ spack mpd build -G Ninja -j12
```

Development workflow

Activate environment
Build

```
$ spack env activate my-art-devel
$ spack mpd build -G Ninja -j12
```

CMake configuration

```
==> Configuring with command:

cmake --preset default /scratch/knoepfel/my-art-devel/srcs ...

Preset CMake variables:

  CMAKE_BUILD_TYPE:STRING="RelWithDebInfo"
  ...

-- Found TBB: /.../lib64/cmake/TBB/TBBConfig.cmake (found version "2021.9.0")
-- The C compiler identification is GNU 14.1.0
...
-- Configuring done (2.2s)
-- Generating done (0.2s)
-- Build files have been written to: /home/knoepfel/scratch/my-art-devel/build
```

Build

```
==> Building with command:

cmake --build /scratch/knoepfel/my-art-devel/build -- -j12

[0/2] Re-checking globbed directories...
[278/278] Linking CXX executable cetlib/bin/ntuple_t
```

Development workflow

Test

```
$ spack mpd test -j12

==> Testing with command:

ctest --test-dir /scratch/knoepfel/my-art-devel/build -j12

Internal ctest changing into directory: /home/knoepfel/scratch/my-art-devel/build
Test project /home/knoepfel/scratch/my-art-devel/build
        Start   1: coded_exception_test
        Start   2: demangle_t
        Start   3: exception_collector_test
        Start   4: exception_test
        Start   5: exception_category_matcher_t
        Start   6: exception_message_matcher_t
        Start   7: exception_bad_append_t
        Start   8: runThreadSafeOutputFileStream_t.sh
        Start   9: assert_only_one_thread_test
        Start  10: serial_task_queue_chain_t
        Start  11: serial_task_queue_t
        Start  12: waiting_task_list_t
  1/100 Test   #1: coded_exception_test .........................   Passed    0.01 sec
...
100/100 Test  #55: cpu_timer_test ...............................   Passed    0.55 sec

100% tests passed, 0 tests failed out of 100
```


Fermilab

Helper commands

Status

```
$ spack mpd status
==> Selected project: my-art-devel
    Environment status: active
```


Fermilab

Helper commands

Status

```
$ spack mpd status
==> Selected project: my-art-devel
    Environment status: active
```

List projects available to me

```
$ spack mpd ls

==> Existing MPD projects:

      Project name        Environment    Deployed environment
      ----------------    -----------    --------------------
      art-devel           (none)         (none)
      meld-devel          installed      (none)
   ▶  my-art-devel        active         (none)
      my-larsoft-devel    created        (none)
   ◀  test-devel          (none)         (none)
```

Selected in this shell

Selected in another shell

Fermilab

Helper commands

Status

```
$ spack mpd status
==> Selected project: my-art-devel
    Environment status: active
```

List projects available to me

```
$ spack mpd ls

==> Existing MPD projects:

     Project name         Environment    Deployed environment
     ----------------     -----------    --------------------
     art-devel            (none)         (none)
     meld-devel           installed      (none)
  ▶  my-art-devel         active         (none)
     my-larsoft-devel     created        (none)
  ◀  test-devel           (none)         (none)
```

Selected in this shell

Selected in another shell

Switch MPD Projects

```
$ spack mpd select test-devel
==> Warning: Project test-devel selected in another shell.  Use with caution.
```


Fermilab

Caveats

- **Each repository you want to develop must have a Spack recipe**

  The recipe does not need to be part of the Spack mainline repository.

  Spack has tools to help you create package recipes.

Caveats

- **Each repository you want to develop must have a Spack recipe**

  The recipe does not need to be part of the Spack mainline repository.

  Spack has tools to help you create package recipes.

- **MPD does not use Spack to build the repositories under development**

  Spack forms an environment of *dependencies* used for building the repositories.

  MPD automatically configures CMake to build the repositories together.

Caveats

- **Each repository you want to develop must have a Spack recipe**

  The recipe does not need to be part of the Spack mainline repository.

  Spack has tools to help you create package recipes.

- **MPD does not use Spack to build the repositories under development**

  Spack forms an environment of *dependencies* used for building the repositories.

  MPD automatically configures CMake to build the repositories together.

- **Minimize use of environment variables**

  Spack recipes can (and do) set environment variables during the build and run stages. But when building that code outside of Spack, those variables need to be set in a different way.

  Best to find alternatives to environment variables

  Better insulates each MPD project from each other

Conclusions

- Fermilab is switching from its custom package manager to Spack.
- Multi-package development with Spack will be achieved with the MPD extension.
- MPD is ready for beta-testing.

  Pull requests and bug reports at https://github.com/FNALssi/spack-mpd are welcome.

Thanks for your time.

Fermilab