Developing under Spack: Latest changes to MPD

Kyle J. Knoepfel

10 December 2024

LArSoft coordination meeting

Code development using Spack

We have tried different approaches for replacing MRB:

1. FNAL-created `spack dev`

   LArSoft minimum viable product released in 2019; little response from experiments.

2. Spack-provided feature `spack develop`

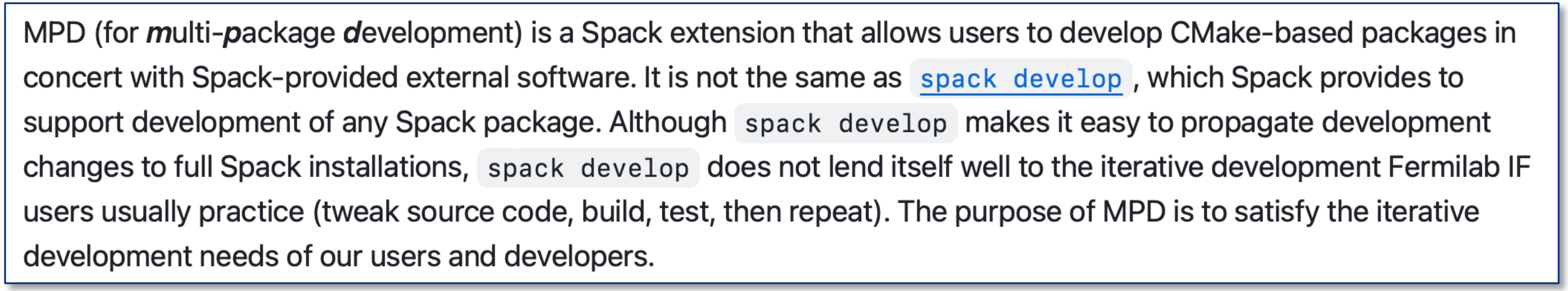
   More Spack expertise required of users, and substantial inefficiencies in incremental builds.

3. Using Spack environments to only provide access to external software

   The work presented here.

Spack MPD

https://github.com/FNALssi/spack-mpd



MPD (for ***m***ulti-***p***ackage ***d***evelopment) is a Spack extension that allows users to develop CMake-based packages in concert with Spack-provided external software. It is not the same as `spack develop`, which Spack provides to support development of any Spack package. Although `spack develop` makes it easy to propagate development changes to full Spack installations, `spack develop` does not lend itself well to the iterative development Fermilab IF users usually practice (tweak source code, build, test, then repeat). The purpose of MPD is to satisfy the iterative development needs of our users and developers.

- Try to give a familiar feel to MRB but retain only those things most commonly used.
- **MPD is now a beta-quality product—you are welcome to try it**. We are happy to accept bug reports and pull requests for documentation and for implementation.
- Developers of SciSoft-dependent packages should use the FNAL-provided bootstrap script, which automatically installs MPD as part of your Spack installation.

Desired features

Spack interactions

- Minimize user’s required knowledge of Spack
- Take advantage of packages installed in upstream Spack 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 session
- Easy to switch between my MPD projects

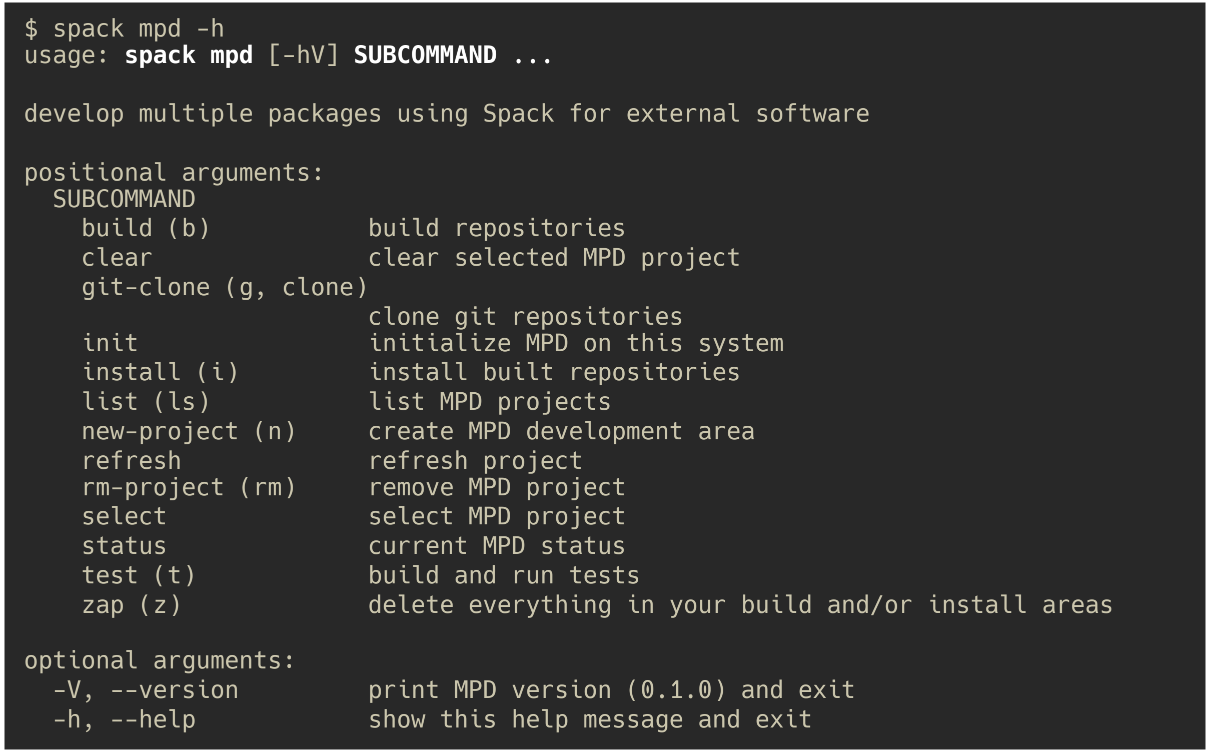
  Avoid reliance on environment variables

- Easy to list which MPD projects are available to me

In the last few months…

- Presented MPD at CHEP24
- Removed need for explicit environment activation by the user
- Can now install MPD projects as Spack packages/environments
- Support variants and the specification of virtual packages
- Merged package specifications from environments specified at the command line
- Removed under-the-covers Spack repository handling

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
    init                 initialize MPD on this system
    install (i)          install built repositories
    list (ls)            list MPD projects
    new-project (n)      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

MRB-like commands

```
mrb i → install
mrb uc → refresh
. localProducts/... → select
```

```
$ 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
    init                initialize MPD on this system
    install (i)         install built repositories
    list (ls)           list MPD projects
    new-project (n)     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

MRB-like commands

```
mrb i → install
mrb uc → refresh
. localProducts/... → select
```

Project commands

You can `clear` a project selection—i.e. start a fresh session without restarting a shell.

```
$ 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
    init                initialize MPD on this system
    install (i)         install built repositories
    list (ls)           list MPD projects
    new-project (n)     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

MRB-like commands

`mrb i` → `install`
`mrb uc` → `refresh`
`. localProducts/...` → `select`

Project commands

You can `clear` a project selection—i.e. start a fresh session without restarting a shell.

Usability

Helper commands exist to let you know 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
    init                  initialize MPD on this system
    install (i)           install built repositories
    list (ls)             list MPD projects
    new-project (n)       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

MRB-like commands

`mrb i` → `install`
`mrb uc` → `refresh`
`. localProducts/...` → `select`

Project commands

You can `clear` a project selection—i.e. start a fresh session without restarting a shell.

Usability

Helper commands exist to let you know 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
    init                initialize MPD on this system
    install (i)         install built repositories
    list (ls)           list MPD projects
    new-project (n)     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
```

Demonstrating how to use MPD to develop software

- This is an example of how MPD can be used; **it is not a tutorial**.
- I will develop 3 packages: `cetlib-except`, `hep-concurrency`, and `cetlib`.

Development workflow

Create new project

```
$ spack mpd new-project --name my-art-devel -T my-art-devel -E gcc-14-2 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)
```

Development workflow

Create new project

```
$ spack mpd new-project --name my-art-devel -T my-art-devel -E gcc-14-2 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

```
$ spack mpd refresh

==> Refreshing 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

  Will develop:
    - cetlib@develop %gcc@14 cxxstd=20 generator=make
    - cetlib-except@develop %gcc@14 cxxstd=20 generator=make
    - hep-concurrency@develop %gcc@14 cxxstd=20 generator=make

==> Determining dependencies (this may take a few minutes)
 ⋮
==> Installing development environment

[+] /usr (external glibc-2.34-mgzp5mjf45pdahg4swhwponbemnk3hmv)
[+] /usr (external glibc-2.34-omul2odw7h4qffee7o63wkjjcmzw2vjr)
 ⋮
[+] /scratch/.../intel-tbb-2021.12.0-cunlldcofd3azuz4urcrzsxbr5vwpodc

==> my-art-devel is ready for development (e.g type spack mpd build ...)
```

Development workflow

Build

```
$ spack mpd build -j12

==> Activating development environment (/scratch/knoepfel/my-art-devel/local)

==> 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 "2021.12.0")
-- The C compiler identification is GNU 14.2.0
 ⋮
-- Configuring done (2.2s)
-- Generating done (0.2s)
-- Build files have been written to: /home/knoepfel/scratch/my-art-devel/build

==> Building with command:

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

 ⋮
[100%] Linking CXX executable ../../bin/regex_t
[100%] Built target regex_t
```

Development workflow

Test

```
$ spack mpd test -j12

==> Activating development environment (/scratch/knoepfel/my-art-devel/local)

==> 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
```

Development workflow

Install

```
$ spack mpd install

==> Activating development environment (/scratch/knoepfel/my-art-devel/local)

==> Installing developed packages with command:

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

==> Installing environment
[+] /usr (external glibc-2.34-mgzp5mjf45pdahg4swhwponbemnk3hmv)
[+] /usr (external glibc-2.34-omul2odw7h4qffee7o63wkjjcmzw2vjr)
 ⋮
[+] /scratch/knoepfel/.../gcc-14.2.0/cetlib-except-develop-f7mbdai7dger5knryqz6inkw4jinuoxo
 ⋮
[+] /scratch/knoepfel/.../gcc-14.2.0/hep-concurrency-develop-tvumszcbf3gg3vistbtahhazkn33jvvc
[+] /scratch/knoepfel/.../gcc-14.2.0/cetlib-develop-lexqohszwojtfkjplmblqji7fxezn5fo
==> Updating view at /scratch/knoepfel/.../my-art-devel/.spack-env/view

==> The my-art-devel environment has been installed.
```

Development workflow

Install

```
$ spack mpd install

==> Activating development environment (/scratch/knoepfel/my-art-devel/local)

==> Installing developed packages with command:

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

==> Installing environment
[+] /usr (external glibc-2.34-mgzp5mjf45pdahg4swhwponbemnk3hmv)
[+] /usr (external glibc-2.34-omul2odw7h4qffee7o63wkjjcmzw2vjr)
  ⋮
[+] /scratch/knoepfel/.../gcc-14.2.0/cetlib-except-develop-f7mbdai7dger5knryqz6inkw4jinuoxo
  ⋮
[+] /scratch/knoepfel/.../gcc-14.2.0/hep-concurrency-develop-tvumszcbf3gg3vistbtahhazkn33jvvc
[+] /scratch/knoepfel/.../gcc-14.2.0/cetlib-develop-lexqohszwojtfkjplmblqji7fxezn5fo
==> Updating view at /scratch/knoepfel/.../my-art-devel/.spack-env/view

==> The my-art-devel environment has been installed.
```

`my-art-devel` can now be used as a base environment for other MPD projects (e.g.):

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

Other commands

Status

```
$ spack mpd status
==> Selected project:    my-art-devel
    Development status:  ready
    Last installed:      2024-12-09 15:48:43
```

List projects

```
$ spack mpd ls

==> Existing MPD projects:

   Project name    Sources directory
   ------------    -----------------------------------
 ▶ my-art-devel    /scratch/knoepfel/my-art-devel/srcs

Type spack mpd ls <project name> for more details about a project.
```

See the documentation at https://github.com/FNALssi/spack-mpd for more info.

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.

  The bootstrap script adds multiple recipe repositories to your Spack configuration.

  If you would like to package a new thing, then spack has tools to help you create a recipe.

- **You should not rely on the presence of specific environment variables**

  Spack recipes can (and do) set environment variables during `spack load`. But when developing that code outside of Spack, those variables will need to be set.

- **To use Ninja you must specify it as part of the `new-project` command:**

```
$ spack mpd new-project --name my-art-devel -T my-art-devel -E gcc-14-2 generator=ninja ...
```

  Because the build generator is part of Spack's concretization, it must be specified before build time.

Upshot

- MPD is the Spack-based replacement of MRB.
- It is ready for beta-testing.

  Pull requests and bug reports at https://github.com/FNALssi/spack-mpd are welcome.
- We are working on things now that are not specific to MPD:

  How to best use upstream Spack instances.

  Why FNAL's Spack fork results in different concretization than a different fork.

Thanks for your time.