Spack Tutorial

Package Creation Tutorial

Yuichi Otsuka

Software Development Technology Unit, R-CCS

- This tutorial is entirely based on the official Spack Tutorial: https://spack-tutorial.readthedocs.io/en/latest/
- The commands used in this tutorial are summarized at the following page: https://github.com/otsukay/spack_tutorial/

0. Setting up Spack

- After starting the container, run the following commands in your home directory.

```
git clone --depth=2 --branch=releases/v1.0 https://github.com/spack/spack
. spack/share/spack/setup-env.sh
spack repo update builtin --tag v2025.07.0
spack mirror add tutorial /mirror
spack buildcache keys --install --trust
spack bootstrap now
spack compiler find
```

- Another way is to run:

```
git clone https://github.com/otsukay/spack_tutorial.git
source ~/spack_tutorial/20251007/0_setup-spack.sh
```

1. Getting Started

- Let us create and add a package repository

```
spack repo create $HOME/my_pkgs tutorial
spack repo add $HOME/my_pkgs/spack_repo/tutorial
```

- Now let's look at the available repositories:

```
spack repo list
```

- Configuration information can be checked as follows:

```
spack config get repos
```

2. Creating the Package File

We will learn how to create a recipe using `mpileaks` as an example.

- Spack can automatically generate a recipe template from a source code archive.

```
spack create --skip-editor --name tutorial-mpileaks \
https://github.com/LLNL/mpileaks/archive/refs/tags/v1.0.tar.gz
```

- Let’s first try `spack install` to see how Spack uses the skeleton.

```
spack install tutorial-mpileaks
```

3. Adding Package Documentation

Before fixing the error, add the info to the recipe (Spack convention).

- Edit the recipe with your favorite editor:

```
spack edit tutorial-mpileaks
```

- Then, you can check the information that Spack derives from the recipe.

```
spack info --phases tutorial-mpileaks
```

4. Adding Dependencies

According to the README file, `mpileaks` depends on three third-party libraries: `mpi`, `adept-utils`, and `callpath`.

- Add the dependencies by specifying them using the `depends_on` directive:

```
spack edit tutorial-mpileaks
```

- Now, let’s check if the dependencies build correctly.

```
spack install tutorial-mpileaks
```

5. Debugging Package Builds

Reviewing the Build Log

- The full build log can be found in the staging directory.

```
cat $(spack location tutorial-mpileaks)/../spack-build-out.txt
```

Building Manually

- Let's try building the package manually to see how to fix the problem.

```
spack cd tutorial-mpileaks
spack build-env tutorial-mpileaks bash
```

```
./configure \
--prefix=/home/spack/spack/opt/spack/linux-x86_64_v3/tutorial-mpileaks-1.0-bsn3xirjorc7l2stboooavdkrd4aa3q4
```

6. Specifying Configure Arguments

We now know which options we need to pass to `configure`.

- Add the `--with-adept-utils` and `--with-callpath` arguments in the configure_args method

```
spack edit tutorial-mpileaks
```

- Now let's try the build again:

```
spack install tutorial-mpileaks
```

Success?

7. Adding Variants

Optional features of the software can be provided through variants.

Here, we use two options that take integer values, `--with-stack-start-c` and `--with-stack-start-fortran`, as examples.

- Add `variant` directive to the recipe:

```
spack edit tutorial-mpileaks
```

- Let us install this variant:

```
spack install --verbose tutorial-mpileaks stackstart=4
```

8. Adding Tests

Spack provides testing features — here we add a simple sanity check to see the result.

- Add `sanity_check_is_dir` (with a typo) to the recipe:

```
spack edit tutorial-mpileaks
```

- Check how it works:

```
spack uninstall -ay tutorial-mpileaks
spack install --test=root tutorial-mpileaks
```

- Let’s properly fix the error and try again:

```
spack edit tutorial-mpileaks
spack install --test=root tutorial-mpileaks
```