# UG2.6.2: How to install via Spack on Cineca clusters

*In this page:*

---

Spack is a multi-platform package manager that allows to easily install multiple versions and configurations of software. It is especially useful for building and maintaining installations of many different versions of the same software.
Currently Spack is available on MARCONI, GALILEO100 and LEONARDO. Please refer to the spack documentation page  for more detailed information.

## Loading the spack module available on the cluster

We provide a module to load a pre-configured Spack instance:

```
$ modmap -m spack
$ module load spack/<version>
```

When you load the "prod" module, you will use a Spack instance configured to compile software in the CINECA environment.
When you connect to a CINECA cluster, "preprod" and "prod" Spack modules are available. The pre-production module should only be used when no production module of Spack is available.

By loading a spack module, the setup-env.sh file is sourced. Then $SPACK_ROOT is initialized to /cineca/prod/opt/tools/spack/<vers>/none, spack command is added to your PATH, and some nice command line integration tools too.
The directory /spack-<version> is automatically created into a default space, and it contains some sub-directories created and used by Spack during the phase of the package installation. When you load the spack module, the list of the sub-directories is shown, with the full paths: on MARCONI and GALILEO100, the default area is $WORK/$USER, while on LEONARDO this is $PUBLIC. You will find then, for example on LEONARDO:

- sources cache: $PUBLIC/spack-<version>**/cache**
- software installation root: $PUBLIC/spack-<version>**/install**
- modulefiles location: $PUBLIC/spack-<version>**/modules**
- user scope:  $PUBLIC/spack-<version>**/user_cache**

For MARCONI and GALILEO100 users, notice that $WORK space will be removed at the end of the corresponding project. If you want to define different paths for cache, installation, modules and user scope directories, please refer to Spack manual.

## Listing the software packages available to be installed

You can check if the software package you want install is available via spack with the comand spack list, which prints out a list of all of the packages Spack can install. You can also specified the name of the package:

```
$ spack list <package_name>
```

or

```
$ spack list | grep <package_name>
```

### Providers

In the spack environment, "virtual packages" are defined (e.g. mpi), which are provided by multiple specific packages (e.g. intel-oneapi-mpi, openmpi, ...). The list of the available virtual packages is given by

```
$ spack providers
```

List the packages that are able to provide a specific virtual package

```
$ spack providers <virtual_package_name>
e.g $ spack providers mpi
```

(e.g. intel-oneapi-mpi and openmpi packages provide the mpi virtual package; netlib-lapack and openblas provide the lapack virtual package).

### Variants and dependencies

If the package of your interest is available to be installed via spack, look at its build “variants” that yon can switch in/off or customize, and the “dependencies” that will be used for the building, linking and running phases and you are able to personalize:

```
$ spack info <package_name>
```

## Listing the software packages installed from Cineca staff via spack

Cineca staff has already installed through the spack module that you loaded a suite of compilers, libraries, tools and applications and you can use them to install additional software (it is strongly recommended). List the packages already installed

```
$ spack find
```

Check if a specific package is already installed or what packages have been already installed to provide a specific virtual package (e.g mpi)

```
$ spack find <package_name>
$ spack find <virtual_package_name>
```

List the packages already installed and see e.g. the used variants (-v), dependencies (-d), the installation path (-p) and the hash (-l). The meaning of the hash is discussed in the next paragraph.

```
$ spack find -ldvp <package>
```

You can also list the packages already installed with a specific variant

```
$ spack find -l +<variant>
e.g. $ spack find -l +cuda
```

or which depends on a specific package (e.g openmpi) or a generic virtual package (e.g. mpi)

```
$ spack find -l ^<package_name>
e.g. $ spack find -l ^openmpi
e.g. $ spack find -l ^mpi
```

or installed with a specific compiler

```
$ spack find %<compiler>
```

The list of all the compilers already installed and ready to be used can be seen with

```
$ spack compilers
```

## Installing a new package

### Spec command

In order to install a package with the spack module you have to select for it a version, a compiler, the dependencies and the building variants (we will show how to do it in the following). The combination of all these parameters is the "spec" with which the package will be installed.

If you don’t select any combination during the installation, a default spec is selected from spack. Before installing a package, it is strongly recommended to check the default spec with which the package would be installed:

```
$ spack spec <package_name>
```

Each spec is identified by a unique number (e.g. aouyzha), named “hash” and displayed by “-l” (long) option, that distinguishes the several installations of a same package.
See the hash for all the installations of a specific package

```
$ spack spec -l <package_name>
```

e.g. see the different hash for openmpi%intel and openmpi%gcc

```
$ spack spec -l openmpi %intel
aouyzha  openmpi@4.1.1%intel@2021.4.0~atomics+cuda~cxx~cxx_exceptions~gpfs~internal-
hwloc~java+legacylaunchers~lustre~memchecker~pmi+pmix~singularity~sqlite3+static~thread_multiple+vt+wrapper-rpath
fabrics=ucx schedulers=slurm arch=linux-centos8-cascadelake

$ spack spec -l openmpi %gcc
2ovg6ub  openmpi@4.1.1%gcc@10.2.0~atomics+cuda~cxx~cxx_exceptions~gpfs~internal-
hwloc~java+legacylaunchers~lustre~memchecker~pmi+pmix~singularity~sqlite3+static~thread_multiple+vt+wrapper-rpath
fabrics=ucx schedulers=slurm arch=linux-centos8-cascadelake
```

Another suggested option of the "spec" command is " -I" (install), which shows the installation status of the requested package and its dependencies.
See the hash and install status of the selected packages and its dependencies

```
$ spack spec -Il <package_name>
```

e.g. see hash and installing status of openmpi and its cuda dependency

```
$ spack spec -Il openmpi
-   aouyzha openmpi@4.1.1%intel@2021.4.0~atomics+cuda~cxx~cxx_exceptions~gpfs~internal-
hwloc~java+legacylaunchers~lustre~memchecker~pmi+pmix~singularity~sqlite3+static~thread_multiple+vt+wrapper-rpath
fabrics=ucx schedulers=slurm arch=linux-centos8-cascadelake
  [-]  rdd7huh      ^cuda@11.5.0%intel@2021.4.0~dev arch=linux-centos8-cascadelake
```

The installation status is described by the symbol “-” (not installed) or “+/^” (installed/installed from an other user) that precedes the hash of the spec.
If the dependency spec (e.g. cuda) of the selected package (e.g openmpi) is not installed (-), it will be by default from spack during the installation of the selected package. The dependency spec installed by default from spack to meet  the building requirement of the selected package is called “implicit” installation, the selected package “explicit” installation.

On Cineca clusters it’s recommended to execute always “spec” command before installing a package to make sure its dependencies are fullfilled by the cineca installations (^) if they are available. The Cineca installations are optimised and tested for the architecture of the specific cluster.
If a dependency is not fullfilled by the Cineca installation even if it is available, it shows “-” symbol and it will be installed with a different hash, because it's a different spec.

In the following example, openmpi dependency is not fullfilled by the cineca installation (-) even if it is already available with a specific hash as the "spack find -l openmpi"  command  shows. This means that openmpi will be installed again from spack with a different combination of building values and consequently with a different hash (aouyzha). In order to use the Cineca openmpi installation you would have to provide its spec to spack, for example through the hash shown by the following command

```
$ spack find -l openmpi
==> 1 installed package
-- linux-centos8-cascadelake / gcc@10.2.0 -----------------------
ov3ei7j openmpi@4.1.1
```

# Example: parmetis will be installed with a spec of openmpi that has not been installed from Cineca staff (-)

```
$ spack spec -Il parmetis %gcc@10.2.0 ^openmpi
-   oytpej7  parmetis@4.0.3%gcc@10.2.0~gdb~int64~ipo+shared build_type=RelWithDebInfo
patches=4f892531eb0a807eb1b82e683a416d3e35154a455274cf9b162fb02054d11a5b,
50ed2081bc939269689789942067c58b3e522c269269a430d5d34c00edbc5870,
704b84f7c7444d4372cb59cca6e1209df4ef3b033bc4ee3cf50f369bce972a9d arch=linux-centos8-cascadelake
  [-]      aouyzha   ^openmpi@4.1.1%gcc@10.2.0~atomics+cuda~cxx~cxx_exceptions~gpfs~internal-
hwloc~java+legacylaunchers~lustre~memchecker~pmi+pmix~singularity~sqlite3+static~thread_multiple+vt+wrapper-rpath
fabrics=ucx schedulers=slurm arch=linux-centos8-cascadelake
```

# Example: parmetis will be installed with the spec installed from Cineca staff defined by the hash "ov3ei7j" as the command "spack find -l openmpi" shows

```
$ spack spec -Il parmetis %gcc@10.2.0 ^/ov3ei7j
-   oefbg6x  parmetis@4.0.3%gcc@10.2.0~gdb~int64~ipo+shared build_type=RelWithDebInfo
patches=4f892531eb0a807eb1b82e683a416d3e35154a455274cf9b162fb02054d11a5b,
50ed2081bc939269689789942067c58b3e522c269269a430d5d34c00edbc5870,
704b84f7c7444d4372cb59cca6e1209df4ef3b033bc4ee3cf50f369bce972a9d arch=linux-centos8-cascadelake
 [^]      ov3ei7j      ^openmpi@4.1.1%gcc@10.2.0~atomics+cuda~cxx~cxx_exceptions~gpfs~internal-
hwloc~java+legacylaunchers~lustre~memchecker~pmi+pmix~singularity~sqlite3+static~thread_multiple+vt+wrapper-rpath
fabrics=ucx schedulers=slurm arch=linux-centos8-cascadelake
```

## Install command

For a specific package you can install the default spec  or a custom spec by selecting a specific version (through @ symbol), and/or a specific compiler (through % symbol), building variants (through +,- or ~,=  symbols) and specific dependencies (through ^ symbol).

Default package installation:

```
$ spack install <package_name>
```

or

```
$ spack install /<package_hash>
```

You can use <package_name> or /<package_hash> without distinction, in the following for simplicity all the examples are reported with <package_name>.

Overall view of a custom package installation:

```
$ spack install <package_name>@<version> +/~/<variant> <variant>=<value> %<compiler>@<version> ^<dependency_name>
```

The available versions of the package, the building variants and the dependencies that you can customise can be shown by executing the command "spack info <package_name>", as described above.

The command "spack compilers" shows all the compilers you can use for your installation and "spack find -l" all the dependencies installed from Cineca staff with their hash.

Remember to install the new package after executing “spack spec” command as described previously and use the Cineca installations for the dependencies if they are available.

If you installed a new compiler, in order to use it remember to add it to your “compilers.yaml” file in the following way:

```
$ spack install <compiler_package>
$ spack load <compiler_package>
$ spack compiler add
$ spack compilers
```

## Module command

After installing a package, you can load it with the spack command

```
$ spack load <package_name>
```

Otherwise you can create a modulefile and load the software package as a module. To create the modulefile of your installed software

```
$ spack module tcl refresh --upstream-modules <package_name>
```

Then you can find and load the new modulefile as

```
$ module load spack
$ module av <package_module>
$ module load <package_module>
```

Please refer to the spack documentation page for more detailed information.