Using the clusters the reproducible way

In order for a research article to be complete, the full environment used to produce the results (the description of the experiment, the code, data, etc) should be provided. It can then be used to reproduce the results and create new work based on the research. Reproducibility guarantees we are following the best practices by opening information for greater scrutiny by peers and an informed public.

Step-by-step guide

This page is a work in progress, come back later for a complete guide.

Hosting your code on c4science

One of the first steps to reproducible and open science is a healthy code base. Nowadays most of the projects use a versionning system called Git. It allows to collaborate easily with coworkers, keep track of what has been done and when, go back in a previous version, have separate development done at the same time and then merged together in due time. This is only a few examples of what can be achieved using Git. Git is a command line utility, which means you need a terminal to run and use the command (cmd.exe, Terminal.app, etc). There are also graphical frontends to Git like TortoiseGit or SourceTree).

Here’s a more complete introduction to Git: https://c4science.ch/w/c4science/introgit/

To use Git you have to create a repositories on a server where the Git history will be stored. You can use the c4science.ch service to host your code by following this guide.

Create an account and setup authentication

Create an account on https://c4science.ch/auth/start/ by clicking in the "Login for Swiss Universities" box in the blue button. Select your university and login using your standard account. C4science will then confirm your email and username, after this step you're logged in. You will need to setup one of the two authentication method. They are described in detail on the following pages:

- Setup SSH keys: https://c4science.ch/w/c4science/sshkeys/
- Setup HTTP password: https://c4science.ch/w/c4science/whatisvcs/

Create and use a repository

Follow the guide to create a repository https://c4science.ch/w/c4science/simplerepo/

Then you can go to the main page of your repository by clicking the View Repository green button on the right. The repository is empty by default, you have to clone it on your computer to start working with it. You need a Git client. On Windows, you should use Git Bash, on Linux you can simply download Git using your package manager (eg. apt-get install git) and on MacOS you can just type git in the terminal and it will prompt you to install it if it's not available.

Open a terminal application (on Windows open Git Bash, on MacOS Terminal.app) and clone your repository using the URI that you find on the main page of your repository and by click on the green button Clone. Choose one of the two options according to the authentication method you choosed (VCS password or SSH key). Now you can clone the repository on your local machine, go to a folder where you want to put the repository and clone it:

```
mkdir repos
cd repos
git clone ssh://git@c4science.ch/source/myrepo.git myrepo
cd myrepo
```

Congratulations, you are now using Git !

Collaboration using Git

Git at its core is made to collaborate on source code. The first step is to have separate environment for specific work (like a feature), for this we use a branch.

```
# Create a new branch with the name of the new feature you want to work on
git checkout -b feature/my-awesome-feature
```
You can then work on your branch and push the branch without interfering with other people work on other branches (master, other feature branches). When your work is done and ready to be published, you can merge a branch with an other, typically we want to merge the feature branch back to the master branch.

```
git checkout master
git merge feature/my-awesome-feature
```

At this point, conflicts are possible, in every files that git was unable to solve a conflict on you will then see a special syntax with the two branches version of the code. You'll have to remove this construct and replace it with a manual version of the two.

<table>
<thead>
<tr>
<th>Conflict</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you have a problem,</td>
</tr>
<tr>
<td>&lt;&lt;&lt;&lt;&lt;&lt; HEAD</td>
</tr>
<tr>
<td>open a ticket.</td>
</tr>
<tr>
<td>=======</td>
</tr>
<tr>
<td>ask on the chat.</td>
</tr>
<tr>
<td>&gt;&gt;&gt;&gt;&gt;&gt; feature/my-awesome-feature</td>
</tr>
</tbody>
</table>

Here's an example of a resolution of the previous conflict

<table>
<thead>
<tr>
<th>After conflict resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you have a problem,</td>
</tr>
<tr>
<td>open a ticket or ask on the chat.</td>
</tr>
</tbody>
</table>

An important aspect of collaboration with Git is code review. On c4science you can use the Arcanist tool to allow easy code review. It means that the changes are not committed in your master branch before being validated by a person or a group. More information on the wiki: https://c4science.ch/w/c4science/code_review/

Retrieving information on the provenance of the software provided on the clusters

Nearly every library or application that is available on our clusters via module files has been installed using Spack. While this allows us (as maintainers of the systems) to have a way to automate complex builds and keep track of dependencies, it also permit our users to know how a particular software was built. First of all, configure information is reported directly in the module files for every CMake or Autotools package:

<table>
<thead>
<tr>
<th>Getting configure options from module files</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ uname -a</td>
</tr>
<tr>
<td>Linux fidis 3.10.0-514.26.2.el7.x86_64 #1 SMP Fri Jun 30 05:26:04 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux</td>
</tr>
<tr>
<td>$ module whatis gnuplot</td>
</tr>
<tr>
<td>gnuplot/5.0.5 : Name : gnuplot</td>
</tr>
<tr>
<td>gnuplot/5.0.5 : Version : 5.0.5</td>
</tr>
<tr>
<td>gnuplot/5.0.5 : Short description : Gnuplot is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms. The source code is copyrighted but freely distributed (i.e., you don't have to pay for it). It was originally created to allow scientists and students to visualize mathematical functions and data interactively, but has grown to support many non-interactive uses such as web scripting. It is also used as a plotting engine by third-party applications like Octave. Gnuplot has been supported and under active development since 1986</td>
</tr>
</tbody>
</table>
In the snippet above we see, for instance, which are the flags used to build gnuplot with our system compilers. This information can be used either to retry the build or as a starting point for a custom build in some user's folder.

A more complete report of what happened at build time can then be found in the library or application prefix. Our modules always set a <package>_ROOT environment variable, pointing to the folder where the software has been installed. Within that folder you'll find a .spack directory that contains all the provenance information for the package. Going back to our gnuplot example:

### Getting the full provenance information

```
$ module show gnuplot
```

```
whatis("Name : gnuplot")
whatis("Version : 5.0.5")
whatis("Short description : Gnuplot is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms. The source code is copyrighted but freely distributed (i.e., you don't have to pay for it). It was originally created to allow scientists and students to visualize mathematical functions and data interactively, but has grown to support many non-interactive uses such as web scripting. It is also used as a plotting engine by third-party applications like Octave. Gnuplot has been supported and under active development since 1986")
```

```
help(["Gnuplot is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms. The source code is copyrighted but freely distributed (i.e., you don't have to pay for it). It was originally created to allow scientists and students to visualize mathematical functions and data interactively, but has grown to support many non-interactive uses such as web scripting. It is also used as a plotting engine by third-party applications like Octave. Gnuplot has been supported and under active development since 1986")])
```

```
prepend_path("PATH","/ssoft/spack/cornalin/v1/opt/spack/linux-rhel7-x86_E5v4_Mellanox/gcc-4.8.5/gnuplot-5.0.5-nf6zg64fyqo27ug43n4slqw6tr6ukt/bin")
```

```
$ module load gnuplot
```

```
$ ls -la $(GNUPLOT_ROOT)/.spack
```

```
total 896
drwxr-xr-x 3 scitasbuild scitas-ge   4096  4 lug  2017 .
drwxr-xr-x 6 scitasbuild scitas-ge   4096  4 lug  2017 ..
-rw-r--r-- 1 scitasbuild scitas-ge  33492  4 lug  2017 build.env
-rw-r--r-- 1 scitasbuild scitas-ge  334684  4 lug  2017 build.out
drwxr-xr-x 3 scitasbuild scitas-ge  4096  4 lug  2017 repos
-rw-r--r-- 1 scitasbuild scitas-ge  20479  4 lug  2017 spec.yaml
```

The content in this folder always includes:

- **build.env**: a dump of the build-time environment
- **build.out**: a complete log of the build, from configuration to installation
- **spec.yaml**: a yaml representation of the DAG (directed acyclic graph) that represents the installation
- **repos**: a folder with all the recipes that were used to build each node of the DAG

and represents the complete set of information one needs to know to reproduce a given build on our system.

Reproduce the build of a package
In principle, any user can reproduce the build of a package that is provided via module files. The first step towards this is to checkout the correct release of Spack in a folder:

### Checkout the correct version of Spack

```
$ uname -a
Linux fidis 3.10.0-514.26.2.el7.x86_64 #1 SMP Fri Jun 30 05:26:04 UTC 2017 x86_64 x86_64 x86_64 GNU/Linux

$ git clone https://github.com/epfl-scitas/spack.git
Cloning into 'spack'...
remote: Counting objects: 101257, done.
remote: Compressing objects: 100% (38/38), done.
remote: Total 101257 (delta 10), reused 7 (delta 0), pack-reused 101215
Receiving objects: 100% (101257/101257), 34.47 MiB | 15.32 MiB/s, done.
Resolving deltas: 100% (48437/48437), done.

$ cd spack/
$ git checkout releases/humagne
$ . share/spack/setup-env.sh
```

and set it up to have the same configuration as the one used for module files:

### Set up a local checkout of Spack

```
$ ln -s /ssoft/spack/humagne/spack.v1/etc/spack/packages.yaml etc/spack/packages.yaml
$ ln -s /ssoft/spack/humagne/spack.v1/etc/spack/compilers.yaml etc/spack/compilers.yaml
$ ln -s /ssoft/spack/humagne/spack.v1/etc/spack/modules.yaml etc/spack/modules.yaml

$ ls -larth etc/spack/
total 0
drwxr-xr-x 3 culpo scitas-ge 4,0K 15 gen 15 ..
drwxr-xr-x 3 culpo scitas-ge 4,0K 15 gen 15 defaults
lrwxrwxrwx 1 culpo scitas-ge 51 15 gen 16 packages.yaml -> /ssoft/spack/humagne/spack.v1/etc/spack/packages.yaml
lrwxrwxrwx 1 culpo scitas-ge 52 15 gen 16 compilers.yaml -> /ssoft/spack/humagne/spack.v1/etc/spack/compilers.yaml
drwxr-xr-x 3 culpo scitas-ge 4,0K 15 gen 16 modules
lrwxrwxrwx 1 culpo scitas-ge 50 15 gen 16 modules.yaml -> /ssoft/spack/humagne/spack.v1/etc/spack/modules.yaml
```

Of course, these two operations needs to be done only the first time you checkout Spack. If the local copy was set up correctly, you should be able to see all the compilers available on our machines:
<table>
<thead>
<tr>
<th>Compiler</th>
<th>Version</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc rhel6-x86_E5v1_IntelIB</td>
<td>gcc7.1.0, gcc6.3.0, gcc5.4.0, gcc4.4.7</td>
<td></td>
</tr>
<tr>
<td>gcc rhel6-x86_E5v2</td>
<td>gcc7.1.0, gcc6.3.0, gcc5.4.0, gcc4.4.7</td>
<td></td>
</tr>
<tr>
<td>gcc rhel6-x86_E5v2_IntelIB</td>
<td>gcc7.1.0, gcc6.3.0, gcc5.4.0, gcc4.4.7</td>
<td></td>
</tr>
<tr>
<td>gcc rhel6-x86_E5v2_Mellanox_GPU</td>
<td>gcc7.1.0, gcc6.3.0, gcc5.4.0, gcc4.4.7</td>
<td></td>
</tr>
<tr>
<td>gcc rhel6-x86_E5v3_IntelIB</td>
<td>gcc7.1.0, gcc6.3.0, gcc5.4.0, gcc4.4.7</td>
<td></td>
</tr>
<tr>
<td>gcc rhel7-x86_64</td>
<td>gcc4.8.5</td>
<td></td>
</tr>
<tr>
<td>gcc rhel7-x86_E5v4_Mellanox</td>
<td>gcc7.1.0, gcc6.3.0, gcc5.4.0, gcc4.8.5</td>
<td></td>
</tr>
<tr>
<td>intel rhel6-x86_E5v1_IntelIB</td>
<td>intel@17.0.2</td>
<td></td>
</tr>
<tr>
<td>intel rhel6-x86_E5v2</td>
<td>intel@17.0.2</td>
<td></td>
</tr>
<tr>
<td>intel rhel6-x86_E5v2_IntelIB</td>
<td>intel@17.0.2</td>
<td></td>
</tr>
<tr>
<td>intel rhel6-x86_E5v2_Mellanox_GPU</td>
<td>intel@17.0.2</td>
<td></td>
</tr>
<tr>
<td>intel rhel6-x86_E5v3_IntelIB</td>
<td>intel@17.0.2</td>
<td></td>
</tr>
<tr>
<td>intel rhel7-x86_E5v4_Mellanox</td>
<td>intel@17.0.2</td>
<td></td>
</tr>
</tbody>
</table>

Note that any time you want to build something with the Intel compiler, you need to load the corresponding module file. This is instead not necessary for GNU-GCC.

At this point you are ready to reproduce the build of any of the software provided via module files. Let's say you want to build the serial version of HDF5 with intel.
Build hdf5 in user’s tree

$ module load intel hdf5
$ spack install -f $HDF5_ROOT/.spack/spec.yaml
  ==> Installing zlib
  ==> Fetching file:///ssoft/spack/mirror/zlib/zlib-1.2.11.tar.gz
  ######################################################################## 100,0%
  ==> Staging archive: /home/culpo/custom-installations/spack/var/spack/stage/zlib-1.2.11-bnwfwf3kkvs5vtlefo3k77fwgmi3/zlib-1.2.11.tar.gz
  ==> Created stage in /home/culpo/custom-installations/spack/var/spack/stage/zlib-1.2.11-bnwfwf3kkvs5vtlefo3k77fwgmi3
  ==> Ran patch() for zlib
  ==> Building zlib [Package]
  ==> Executing phase: 'install'
  ==> Successfully installed zlib
    Fetch: 0.09s.  Build: 11.32s.  Total: 11.41s.
  [+] /home/culpo/custom-installations/spack/opt/spack/linux-rhel7-x86_E5v4_Mellanox/intel-17.0.2/zlib-1.2.11-bnwfwf3kkvs5vtlefo3k77fwgmi3
  ==> Installing szip
  ==> Fetching file:///ssoft/spack/mirror/szip/szip-2.1.tar.gz
  ######################################################################## 100,0%
  ==> Staging archive: /home/culpo/custom-installations/spack/var/spack/stage/szip-2.1-7nsvipktv6uxazgolreqw66bq6nfpgbm/szip-2.1.tar.gz
  ==> Created stage in /home/culpo/custom-installations/spack/var/spack/stage/szip-2.1-7nsvipktv6uxazgolreqw66bq6nfpgbm
  ==> Ran patch() for szip
  ==> Building szip [AutotoolsPackage]
  ==> Executing phase: 'autoreconf'
  ==> Executing phase: 'configure'
  ==> Executing phase: 'build'
  ==> Executing phase: 'install'
  ==> Successfully installed szip
    Fetch: 0.05s.  Build: 25.02s.  Total: 25.07s.
  [+] /home/culpo/custom-installations/spack/opt/spack/linux-rhel7-x86_E5v4_Mellanox/intel-17.0.2/szip-2.1-7nsvipktv6uxazgolreqw66bq6nfpgbm
  ==> Installing hdf5
  ==> Fetching file:///ssoft/spack/mirror/hdf5/hdf5-1.10.1.tar.gz
  ######################################################################## 100,0%
  ==> Staging archive: /home/culpo/custom-installations/spack/var/spack/stage/hdf5-1.10.1-nregnfa66p64fxsnl7pfpszeowwd7g6v/hdf5-1.10.1.tar.gz
  ==> Created stage in /home/culpo/custom-installations/spack/var/spack/stage/hdf5-1.10.1-nregnfa66p64fxsnl7pfpszeowwd7g6v
  ==> Ran patch() for hdf5
  ==> Building hdf5 [AutotoolsPackage]
  ==> Executing phase: 'autoreconf'
  ==> Executing phase: 'configure'
  ==> Executing phase: 'build'
  ==> Executing phase: 'install'
  ==> Successfully installed hdf5
    Fetch: 0.18s.  Build: 4m 20.76s.  Total: 4m 20.94s.
  [+] /home/culpo/custom-installations/spack/opt/spack/linux-rhel7-x86_E5v4_Mellanox/intel-17.0.2/hdf5-1.10.1-nregnfa66p64fxsnl7pfpszeowwd7g6v

As you can see the procedure is as simple as moving where the spec.yaml of interest resides and then ask Spack to reproduce the build in your installation root. In case you need to customize the build of a package and want to build just its dependencies, you can do that with the appropriate option to Spack:

Installing only the dependencies of a package

$ module load intel intel-mkl intel-mpi yambo
$ spack install --only=dependencies -f $YAMBO_ROOT/.spack/spec.yaml
  ==> Installing libxc
  ==> Fetching file:///ssoft/spack/mirror/libxc/libxc-3.0.0.tar.gz
  ######################################################################## 100,0%
  ==> Staging archive: /home/culpo/custom-installations/spack/var/spack/stage/libxc-3.0.0-yx3idyotwd37idy7scr54qtnccmcyti/libxc-3.0.0.tar.gz
  ==> Created stage in /home/culpo/custom-installations/spack/var/spack/stage/libxc-3.0.0-
Ran patch() for libibxc
Building libibxc [Package]
Executing phase : 'install'
Successfully installed libibxc
Fetch: 0.23s. Build: 2m 3.53s. Total: 2m 3.76s.

Ran patch() for zlib
Building zlib [Package]
Executing phase : 'install'
Successfully installed zlib
Fetch: 0.03s. Build: 6.16s. Total: 6.19s.

Ran patch() for szip
Building szip [AutotoolsPackage]
Executing phase : 'autoreconf'
Executing phase : 'configure'
Executing phase : 'build'
Executing phase : 'install'
Successfully installed szip
Fetch: 0.04s. Build: 25.73s. Total: 25.77s.

Ran patch() for hdf5
Building hdf5 [AutotoolsPackage]
Executing phase : 'autoreconf'
Executing phase : 'configure'
Executing phase : 'build'
Executing phase : 'install'
Successfully installed hdf5
Fetch: 0.10s. Build: 4m 24.34s. Total: 4m 24.45s.

Ran patch() for fftw
Building fftw [AutotoolsPackage]
Executing phase : 'autoreconf'
Executing phase : 'configure'
Executing phase : 'build'
Executing phase : 'install'
Successfully installed fftw
Fetch: 0.13s. Build: 7m 24.44s. Total: 7m 24.57s.
Installing libsigsegv

> Installing libsigsegv
> Fetching file:///ssoft/spack/mirror/libsigsegv/libsigsegv-2.11.tar.gz
########################################################### 100.0%
> Staging archive: /home/culpo/custom-installations/spack/var/spack/stage/libsigsegv-2.11-ctoqtpmxuvp3dnscmbuc2ugpytamxs/libsigsegv-2.11.tar.gz
> Created stage in /home/culpo/custom-installations/spack/var/spack/stage/libsigsegv-2.11-ctoqtpmxuvp3dnscmbuc2ugpytamxs
> Ran patch() for libsigsegv
> Building libsigsegv [AutotoolsPackage]
> Executing phase : 'autoreconf'
> Executing phase : 'configure'
> Executing phase : 'build'
> Executing phase : 'install'
> Successfully installed libsigsegv
Fetch: 0.09s. Build: 17.94s. Total: 18.03s.

Installing m4

> Installing m4
> Fetching file:///ssoft/spack/mirror/m4/m4-1.4.18.tar.gz
########################################################### 100.0%
> Staging archive: /home/culpo/custom-installations/spack/var/spack/stage/m4-1.4.18-vgeh4a6acefqm74xtjifpawsl44r3igi/m4-1.4.18.tar.gz
> Created stage in /home/culpo/custom-installations/spack/var/spack/stage/m4-1.4.18-vgeh4a6acefqm74xtjifpawsl44r3igi
> Applied patch gnulib-pgi.patch
> Ran patch() for m4
> Building m4 [AutotoolsPackage]
> Executing phase : 'autoreconf'
> Executing phase : 'configure'
> Executing phase : 'build'
> Executing phase : 'install'
> Successfully installed m4
Fetch: 0.09s. Build: 1m 27.36s. Total: 1m 27.45s.

Installing netcdf

> Installing netcdf
> Fetching file:///ssoft/spack/mirror/netcdf/netcdf-4.4.1.1.tar.gz
########################################################### 100.0%
> Staging archive: /home/culpo/custom-installations/spack/var/spack/stage/netcdf-4.4.1.1-5cziwx52tgjm2exoztoh4m4gipezzl/netcdf-4.4.1.1.tar.gz
> Created stage in /home/culpo/custom-installations/spack/var/spack/stage/netcdf-4.4.1.1-5cziwx52tgjm2exoztoh4m4gipezzl
> Ran patch() for netcdf
> Building netcdf [AutotoolsPackage]
> Executing phase : 'autoreconf'
> Executing phase : 'configure'
> Executing phase : 'build'
> Executing phase : 'install'
> Successfully installed netcdf
Fetch: 0.11s. Build: 1m 8.61s. Total: 1m 8.72s.
After this is done you can modify your package (more information on how to do that in this tutorial), and then install your modified version. Remember that to pin your dependencies, you can always specify them by hash:
Check dependencies are correct when installing

```
$ spack find -l
=> 11 installed packages.
-- linux-rhel7-x86_E5v4_Mellanox / intel@17.0.2 ---------------
von3c3p fftw@3.3.6-pl2 cpry34b intel-mkl@2017.2.174 ctogtpm libsigsegv@2.11 vgeh4a6 m4@1.4.18 m3nfulf netcdf-fortran@4.4.4 bnwiwf zlib@1.2.11 nregnfa hdf5@1.10.1 dshqfoo intel-mpi@2017.2.174 yx3dyco libxc@3.0.0 5cziwx5 netcdf@4.4.1.1 7nsvipk zip@0.1

$ spack spec -Il yambo ~/m3nfulf ~/von3c3p
```

Input spec
-------------------------------
```
er75iiv  yambo
[+] von3c3p  ^fftw@3.3.6-pl2%intel@17.0.2+double+float+long_double+mpi-openmp-pfft_patches-quad arch=linux-rhel7-x86_E5v4_Mellanox
(+) m3nfulf  ^netcdf-fortran@4.4.4%intel@17.0.2 arch=linux-rhel7-x86_E5v4_Mellanox
(+) 5cziwx5  ^netcdf@4.4.1.1%intel@17.0.2-ccdmremote-dap-hdf4 maxdims=1024 maxvars=8192 -mpi-parallel-netcdf shared arch=linux-rhel7-x86_E5v4_Mellanox
(+) nregnfa  ^hdf5@1.10.1%intel@17.0.2+cxx-debug+fortran-mpi+pic+shared+szip-threadsafe arch=linux-rhel7-x86_E5v4_Mellanox
(+) 7nsvipk  ^szip@2.1%intel@17.0.2 arch=linux-rhel7-x86_E5v4_Mellanox
(+) bnwiwf  ^zlib@1.2.11%intel@17.0.2+pic+shared arch=linux-rhel7-x86_E5v4_Mellanox
```
Normalized
-------------------------------
```
adbv64z  yambo
tlu1k6o  ^blas
(+) von3c3p  ^fftw@3.3.6-pl2%intel@17.0.2+double+float+long_double+mpi-openmp-pfft_patches-quad arch=linux-rhel7-x86_E5v4_Mellanox
2xnea7r  ^mpi
(+) nregnfa  ^hdf5@1.10.1%intel@17.0.2+cxx-debug+fortran-mpi+pic+shared+szip-threadsafe arch=linux-rhel7-x86_E5v4_Mellanox
(+) 7nsvipk  ^szip@2.1%intel@17.0.2 arch=linux-rhel7-x86_E5v4_Mellanox
(+) bnwiwf  ^zlib@1.2.11%intel@17.0.2+pic+shared arch=linux-rhel7-x86_E5v4_Mellanox
qyk5o15  ^lapack
ttafm2   ^libxc
(+) 5cziwx5  ^netcdf@4.4.1.1%intel@17.0.2-ccdmremote-dap-hdf4 maxdims=1024 maxvars=8192 -mpi-parallel-netcdf+shared arch=linux-rhel7-x86_E5v4_Mellanox
zwmmot   ^m4
(+) m3nfulf  ^netcdf-fortran@4.4.4%intel@17.0.2+openmp+shared arch=linux-rhel7-x86_E5v4_Mellanox
zh7dk6   ^scalapack
Concretized
-------------------------------
```
ix4fnyp  yambo@0.1.3-intel@17.0.2-ccdmremote+fortran+mpi+shared+szip-threadsafe arch=linux-rhel7-x86_E5v4_Mellanox
(+) von3c3p  ^fftw@3.3.6-pl2%intel@17.0.2+double+float+long_double+mpi-openmp-pfft_patches-quad arch=linux-rhel7-x86_E5v4_Mellanox
(+) dshqfoo  ^intel-mpi@2017.2.174%intel@17.0.2 arch=linux-rhel7-x86_E5v4_Mellanox
(+) nregnfa  ^hdf5@1.10.1%intel@17.0.2+cxx-debug+fortran-mpi+pic+shared+szip-threadsafe arch=linux-rhel7-x86_E5v4_Mellanox
(+) 7nsvipk  ^szip@2.1%intel@17.0.2 arch=linux-rhel7-x86_E5v4_Mellanox
(+) bnwiwf  ^zlib@1.2.11%intel@17.0.2+pic+shared arch=linux-rhel7-x86_E5v4_Mellanox
(+) cpry34b  ^intel-mkl@2017.2.174%intel@17.0.2+ilp64+openmp+shared arch=linux-rhel7-x86_E5v4_Mellanox
(+) yx3dyco  ^libxc@3.0.0%intel@17.0.2 arch=linux-rhel7-x86_E5v4_Mellanox
(+) 5cziwx5  ^netcdf@4.4.1.1%intel@17.0.2-ccdmremote-dap-hdf4 maxdims=1024 maxvars=8192 -mpi-parallel-netcdf+shared arch=linux-rhel7-x86_E5v4_Mellanox
(+) vgeh4a6  ^m4@0.1.2%intel@17.0.2+sigsegv arch=linux-rhel7-x86_E5v4_Mellanox
(+) ctogtpm  ^libsigsegv@2.11%intel@17.0.2 arch=linux-rhel7-x86_E5v4_Mellanox
(+) m3nfulf  ^netcdf-fortran@4.4.4%intel@17.0.2+openmp+shared arch=linux-rhel7-x86_E5v4_Mellanox
```
```
• Using ANSYS with Remote Solve Manager at Windows
• Compiling codes on different systems
• Using the clusters
• How to configure PuTTY SSH to transfer files to Cluster from Windows