Applications and libraries

Fantastic Applications and how to load them

An important number of applications and libraries are available on Baobab and Yggdrasil, and we often offer multiple versions.

The magic which allows that is `lmod` with the `module` command. This is the recommanded way to load any application or libraries on the clusters.

On this page, we will give the most common `module` usage and give some example for a selection of applications.

Module - lmod

The recommanded way to load an application on the clusters is to use the `module` command. By using `module`, you don't need to know where the software and libraries are physically located (the full path), but instead you can just type the application name as if its path was in your PATH (this is indeed what `module` does).
The `module` command can also set other important environment variables for an application, so it is always recommanded to use it.

Find installed applications with 'module'

How to use 'module'

These are the most common options you will use with `module`.

To get a **complete list of applications available** with `module`

```
module spider
```

To find the available version of a certain application :

```
module spider <app_name>
```

To load one (or more) application :

```
module load <app_name_1> <app_name_2> ...
```

To load a specific version of an application :

```
module load <app_name/version>
```

Hint : Module version. By choosing a module without specifying a version, you will always get the latest version available. However, we always recommand to specify the version, as your code might produce different results if you are using another version. If **reproducibility of results** is important for you, you should definitely used a fixed version.

You can see the help for a particular module (it must be loaded first):

```
module help R
```

See the list of currently loaded modules :

```
module list
```

To unload all currently loaded modules :

```
module purge
```

Hint : If you are in a hurry you can also use `ml` instead of `module` for any of the above mentioned commands :

```
ml spider
```

For more information, use the manpage `man module`.

What do I do when an application is not available via 'module' ?

If the application you need or the exact version is not available via `module` :

- First drop us an email at hpc@unige.ch and explain with as much detail as possible what you need (provide scripts, links, etc.). If we can install what you need, we will do it.
- If we cannot, you can compile binaries in your `$HOME` directory and use them on any node of the cluster (since your `$HOME` is accessible from any node). Make sure you load the compiler with module first!
- Another interesting option is to use Singularity which allows you to run containers on the clusters.

Detailed example of using 'module'

Let's go through an example of loading R.

First, let's find all available version of R :

```
[brero@login2 ~]$ module spider R

----------------------------------------------------------------------------
----------------------------------------------
  R:
```

```
-------------------------------------------------------------------------------
-------------------------------------------
    Description:
      R is a free software environment for statistical computing and
graphics.

     Versions:
        R/3.2.3
        R/3.3.1
        R/3.3.2
        R/3.5.1
        R/3.6.0
        R/3.6.2
        R/4.0.0
     Other possible modules matches:
        APR  APR-util  BioPerl  Bismark  Blender  CellProfiler  CellRanger
CoordgenLibs  DISCOVARdenovo  DendroPy  ...
[...]
```

Now some people just need the latest version available and can simply load it with `module load R` ; whitout specifying a version, you will always get the latest version.
But sometimes, you need to use the same version everytime and we recommand it. This is very important as your code might produce different results if you are using another version. If reproducibility of results is important for you, you should definitely used a fixed version.

To load a specific version, in this case R version 3.6.2 :

```
[brero@login2 ~]$ module load R/3.6.2
Lmod has detected the following error:  These module(s) or extension(s)
exist but cannot be loaded as requested:
"R/3.6.2"
   Try: "module spider R/3.6.2" to see how to load the module(s).
```

This fails as the module “cannot be loaded as requested”. This is usually because you are missing dependencies.
The message also suggest to try the following command :

```
[brero@login2 ~]$ module spider R/3.6.2

-------------------------------------------------------------------------------
-------------------------------------------
  R: R/3.6.2
-------------------------------------------------------------------------------
-------------------------------------------
    Description:
      R is a free software environment for statistical computing and
graphics.


    You will need to load all module(s) on any one of the lines below before
the "R/3.6.2" module is available to load.
```

```
      GCC/8.3.0  OpenMPI/3.1.4
[...]
```

As the message explains, you **need** to load 2 dependencies `GCC/8.3.0` and `OpenMPI/3.1.4` before you can load R.

You can then simply execute the following command :

```
[brero@login2 ~]$ module load GCC/8.3.0 OpenMPI/3.1.4 R/3.6.2
```

Then you can just invoke R by typing R in the terminal (instead of using the full path). Of course you are still required to use SLURM to launch your software. Fix Me! link to Slurm example Fix Me! link to R doc.

Be aware that this is a basic example. In many cases, the dependencies will themselves require other dependencies, so you would have to load all dependencies in the correct order. And in some cases, if you use `MPI` for instance, you can simplify that by using the correct compiler toolchain (see below).

Hint : To automatically load some modules at login, you can add something like this in your `$HOME/.bashrc`:

```
if [ -z "$BASHRC_READ" ]; then
  export BASHRC_READ=1
  # Place any module commands here
  module load GCC/8.3.0 OpenMPI/3.1.4 R/3.6.2
fi
```

Choosing the compiler toolchain

You have the choice between FOSS toolchain or Intel toolchain (license required - see Fix Me!).

module	compiler	mpi
foss/2016a	gcc 4.9.3	openmpi 1.10.2
foss/2016b	gcc 5.4.0	openmpi 1.10.3
foss/2017a	gcc 6.3.0	openmpi 2.0.2
foss/2017b	gcc 6.4.0	openmpi 2.1.1
foss/2018a	gcc 6.4.0	openmpi 2.1.2
foss/2018b	gcc 7.3.0	openmpi 3.1.1
foss/2019a	gcc 8.2.0	openmpi 3.1.3
foss/2019b	gcc 8.3.0	openmpi 3.1.4
foss/2020a	gcc 9.3.0	openmpi 4.0.3
intel/2016a	icc 16.0.1	impi 5.1.2

module	compiler	mpi
intel/2016b	icc 16.0.3	impi 5.1.3
intel/2017a	icc 17.0.1	impi 2017 Update 1
intel/2018a	icc 18.0.1	impi 2018.1.163
intel/2019a	icc 19.0.1	impi 2018.4.274

If you want to compile your software against MPI, it is very important not to compile using directly `gcc`, `icc` or similar commands, but rather rely on the wrappers `mpicc`, `mpic++`, `mpicxx` or similar ones.

All the newer versions of `mpi` will be available through the use of module.

Example for the latest version of `gcc`:

```
module load foss
```

If needed, you can stick to a particular (or legacy) version:

```
module load foss/2017a
```

Example for the latest version of icc from Intel:

```
module load intel
```

You can see the details of what is loaded in foss or intel with:

```
module list
```

Intel compiler licenses

If you want to use the Intel compiler, you need to have your own intel license compiler.

If you are an academic you can get one free of charge here. You should ask for the “parallel studio xe”. Once you get the license you should copy it to your home directory in a directory named `Licenses` ($HOME/Licenses)

Examples for selected applications

ADF

You can use SCM ADF on one ore more nodes of the cluster. Please note that you must use `module` (see Module - lmod) in you sbatch script to set the variables correctly.

Please see ADF example on GitLab for some example and scripts related to ADF.

Important

Do not launch ADF using srun. ADF is a wrapper which uses srun internally.

Important

ADF needs a fast local scratch space. On Baobab, the local scratch of each node is only about 180GB. If you need more space, you need to find another solution (do the calculation on more nodes, do not use local scratch, buy us new hard disks)

Important

By default, the local scratch space is defined in the SCM_TMPDIR environment variable, /scratch as set by module load ADF . This default value will gives error when calling ADFview on the login nodes, given that /scratch does not exist there. You can overcome it by using the Linux default /tmp :

```
capello@login2:~$ module load ADF/2019.104
capello@login2:~$ SCM_TMPDIR=/tmp adfview
```

Ansys Fluent

You can use Fluent on Baobab. To do so, please see Fluent on gitlab and adapt the sbatch script.

Fall3d

You can use Fall3D on Baobab. To do so, please see Fall3d on gitlab and adapt the sbatch script.

Gaussian

You can use Gaussian g09 on one node of the cluster. Please note that you must use `module` (see Module - lmod) in you sbatch script to set the variables correctly. When using the module, it will set the variable GAUSS_SCRDIR to /scratch of the local hard disk of the allocated node. This should lower the calculation time and as well lower the usage of the shared filesystem. See below for other optimizations.

There are two versions of g09 on the cluster. Revision c01 and d01.

Please see Gaussian example on GitLab for some examples and scripts related to Gaussian.

To optimize the run, you can add some lines in your job file. If you need more than 190 GB of scratch space, you should add the line (adapt /home/yourusername to your own path):

```
%RWF=/scratch/,170GB,/home/yourusername/scratch/,-1
```

You may as well specify how much memory you want to use. By default, Gaussian will use 250MB of ram. You can try with 50GB for example:

```
%Mem=50GB
```

You need to specify as well how many CPU cores you want to use:

```
%NProcShared=16
```

Git

To use git on the cluster you need to do the following:

Add that to your `${HOME}/.gitconfig`:

```
[core]
createObject = rename
```

To invoke git:

```
git clone --no-hardlinks
```

Gnuplot

Usage :

```
gnuplot
```

Gurobi

NB , the following instructions come from <https://hpc-community.unige.ch/t/gurobi-solver-license-issue/459>.

We provide an internal Gurobi token server, here how to use it:

```
capello@login2:~$ module load Gurobi
capello@login2:~$ gurobi_cl --tokens

Checking status of Gurobi token server 'master.cluster'...

Token server functioning normally.
Maximum allowed uses: 4096, current: 0

capello@login2:~$
```

Jupyter notebook

Jupyter notebook can run on our clusters, however we do not have a dedicated server for Jupyter. This means you need to submit a job and request some resources before you can connect to your instance of Jupyter notebook.

Please, read : [tutorial] Jupyter notebook

For interactive programs such as Jupyter notebook, you might want to use the `interactive-cpu` partition (Which partition for my job)

Keras

Please see Keras on gitlab for some examples and scripts related to Keras

Mathematica

If you would like to use a different license server (by default mathlm.unige.ch), you can specify its URL in the ${HOME}/.Mathematica/Licensing/mathpass file, prependend by an exclamation mark:

```
capello@login2:~$ MATHEMATICA_LICENSE_SERVER_URL=mathlm.unige.ch
capello@login2:~$ mkdir -p ~/.Mathematica/Licensing/
capello@login2:~$ cat <<EOF >~/.Mathematica/Licensing/mathpass
!${MATHEMATICA_LICENSE_SERVER_URL}
EOF
capello@login2:~$ cat ${HOME}/.Mathematica/Licensing/mathpass
!mathlm.unige.ch
capello@login2:~$
```

See http://support.wolfram.com/kb/25655 and http://support.wolfram.com/kb/112 for more information.

Matlab

Matlab is available in Baobab in different versions:

```
$ module spider matlab

-----------------------------------------------------------------------------
-------
  matlab:
-----------------------------------------------------------------------------
```

```
-------
     Versions:
        matlab/2017b
        matlab/2018b
        matlab/2019b
```

Keep in mind that it's a licensed program, and that the licenses are shared with the whole university. To be fair with the other users, we have set up a limitation on the number of licenses you can use. We kindly ask you to specify in your sbatch file that you are using Matlab in order to keep the limitation effective. If you are using some licensed toolbox, like Wavelet_Toolbox, you need to specify it as well. If you don't, your job may be killed without further notice in case we are out of licenses.

Example to specify that you need `Matlab`:

```
#SBATCH --licenses=matlab@matlablm.unige.ch
```

Example to specify that you need the `Wavelet_Toolbox`:

```
#SBATCH --licenses=wavelet-toolbox@matlablm.unige.ch
```

See the licenses available on Baobab:

```
scontrol show lic
```

If you need a license not listed here, please ask us at **hpc [at] unige [dot] ch**.

Important

You need to specify at LEAST the Matlab license, and zero or more toolbox.

To run Matlab in batch mode, you can create a batch file like this one:

```
#!/bin/bash

#SBATCH --cpus-per-task=1
#SBATCH --ntasks=1
#SBATCH --licenses=matlab@matlablm.unige.ch

module load matlab

BASE_MFILE_NAME=hello

unset DISPLAY

echo "Running ${BASE_MFILE_NAME}.m on $(hostname)"

srun matlab -nodesktop -nosplash -nodisplay -r ${BASE_MFILE_NAME}
```

In this example, you need to have your code in the file hello.m.

You submit the Matlab job like a normal sbatch SLURM job:

```
sbatch ./yourBatch
```

Parallel with Matlab

Since version 2014, you are not anymore limited to 12 cpu cores.

Please see Matlab on gitlab for some examples and scripts related to Matlab parallel

As we are talking about parallel and not distributed Matlab, you can consided Matlab as a multithread application.

see how to submit Multithreaded jobs.

Attention

If you are a parallel or distributed Matlab specialist and you have some hints, you are very welcome to contact us!

Pass sbatch arguments to Matlab

You can pass arguments from sbatch to Matlab as described below.

Example of sbatch file (sbatch.sh)

```
[sbatch part as usual]

BASE_MFILE_NAME=test
MATLAB_MFILE=${BASE_MFILE_NAME}.m

unset DISPLAY

module load matlab

#the variable you want to pass to matlab
job_array_index=${SLURM_ARRAY_TASK_ID}

echo "Starting at $(date)"
echo "Running ${MATLAB_MFILE} on $(hostname)"


# we call the matlab function (see the parenthesis around the argument) and
the argument type will be integer.
srun matlab -nodesktop -nosplash -nodisplay -r
"${BASE_MFILE_NAME}($job_array_index)"
echo "Finished at $(date)"
```

Example of Matlab file (test.m)

```
function test(job_array_index)

fprintf('array index: %d\n', job_array_index)
```

See arguments with Matlab on gitlab for more examples.

Compile your Matlab code

Thanks to Philippe Esling for his contribution to this procedure.

The idea of compiling Matlab code is to save on licenses usage. Indeed, once compiled, a Matlab code can be run without using any license.

The Matlab compiler is named MCC

First load the needed modules:

```
module load foss/2016a matlab/2016b
```

Let's say you want to compile this .m file:

```
function hello(name)
   strcat({'Hello '}, name)
end
```

This operation compiles it (this takes some time) :

```
DISPLAY="" mcc -m -v -R '-nojvm, -nodisplay' -o hello hello.m
```

If you have some other .m files that you need to include, you need to explicitly specify their location as follows:

```
mcc -m -v -R '-nojvm, -nodisplay' -I /path/to/functions/ -I
/path/to/other/functions/ [...]
```

The resulting files are a text file named readme.txt, a script named run_hello.sh and an executable named hello.

You can then launch the executable hello like any other executable using a sbatch script:

```
#!/bin/bash

#SBATCH --partition=debug-EL7
#SBATCH --ntasks=1

module load foss/2016a matlab/2016b

srun ./hello Philippe
```

In this case, Philippe is an argument for the function hello. Be careful, arguments are always passed to Matlab as strings.

Important

You do NOT need to specify the Matlab license once compiled, and you are not restricted by the number of available licenses.

Please see compile Matlab on gitlab for some examples and scripts related to Matlab compilation.

Matlab PATH

If you need to add a directory to the Matlab path, for example to use a toolbox installed by you, please proceed as follow.

Add the needed path recursively to matlab path:

```
addpath(genpath('/home/sagon/tests/matlab/dtb/decoding_toolbox_v3.991'))
```

Save the current matlab path to the default matlab definition path:

```
savepath('/home/sagon/pathdef.m')
```

You can now access any file from the toolbox directly:

```
demo2_simpletoydata
```

Matlab java.opts

If you are getting erros such as the one listed here when using Matlab through X2go, you can try the following changes to mitigate the issue.

Create a file named java.opts in the default startup folder which is the folder from which you started MATLAB.

Put the following content in it:

```
-Dsun.java2d.xrender=false
```

And restart Matlab. This will probably reduce the graphical performance but should not impact the computation time.

Wavelab

To use the Wavelab library with Matlab, load Matlab 2014. :

```
module load matlab/2014b
```

Launch Matlab as usual and type this command to go to the Wavelab library:

```
cd /opt/wavelab/Wavelab850/
Wavepath (answer /opt/wavelab/Wavelab850)
```

Meep

See Meep on gitlab for an example how to use Meep.

Nvidia CUDA

Currently on Baobab there are several nodes equiped with GPUs. To request a GPU, it's not enough the specify a partition with nodes having GPUs, you must as well specify how many GPUs and optionnaly the needed type.

To specify how many GPU you request, use the option `–gres=gpu:n` with n having a value between 1 and the maximum according to the table below.

You should also specify the type of GPU you want:

- titan, for single precision computation, like machine learning.
- pascal, for double precision computation like physical simulations.
- rtx, to accelerate machine learning, data science workflows and ray tracing.

Example to request three titan cards: `–gres=gpu:titan:3`.

In the following table you can see which type of GPU is available on Baobab.

GPU model	Compute Capability	SLURM resource	numbers per node	node	partition
titan x (pascal)	6.1	titan	3	gpu[002]	shared-gpu, dpnc-gpu
p100	6.0	pascal	5	gpu[004-005]	shared-gpu, dpt-gpu, kruse-gpu
p100	6.0	pascal	8	gpu[006]	shared-gpu, kruse-gpu, hepia-gpu
p100	6.0	pascal	4	gpu[007]	shared-gpu, schaer-gpu
titan x (pascal)	6.1	titan	8	gpu[008]	shared-gpu, kalousis-gpu
titan x (pascal)	6.1	titan	8	gpu[009-010]	shared-gpu, cui-gpu
RTX 2080 Ti (turing)	7.5	rtx	2	gpu[011-014]	shared-gpu, hepia-gpu, gervasio-gpu

Partitions:

- `shared-gpu` with a max execution time of 12h, tesla, titan and pascal cards
- `dpt-gpu` with a max execution time of 2 days, 5 x pascal GPU cards
- `dpnc-gpu` with a max execution time of 2 days, 6 x titan GPU cards
- `kruse-gpu` with a max execution time of 2 days, 10 x pascal GPU cards
- `hepia-gpu` with a max execution time of 2 days, 3 x pascal GPU cards

- `schaer-gpu` with a max execution time of 2 days, 4 x pascal GPU cards
- `kalousis-gpu` with a max execution time of 7 days, 8 x titan GPU cards
- `cui-gpu` with a max execution time of 2 days, 16 x titan GPU cards
- `hepia-gpu` with a max execution time of 2 days, 2 x rtx GPU cards

Example of script (see also https://gitlab.unige.ch/hpc/softs/tree/master/c/cuda):

```
#!/bin/env bash

#SBATCH --partition=shared-gpu
#SBATCH --time=01:00:00
#SBATCH --gres=gpu:titan:1

module load CUDA

# see here for more samples:
# /opt/cudasample/NVIDIA_CUDA-8.0_Samples/bin/x86_64/linux/release/

# if you need to know the allocated CUDA device, you can obtain it here:
echo $CUDA_VISIBLE_DEVICES

srun deviceQuery
```

If you want to see what GPUs are in use in a given node:

```
scontrol -d show node gpu002
[...]
Gres=gpu:titan:3
GresUsed=mic:0,gpu:titan:3(IDX:0-2)
[...]
```

In this case, this means that node gpu002 has three Titan cards, and all of them are allocated.

If you just need a gpu and you don't care of the type, don't specify it:

```
#SBATCH --gres=gpu:1
```

It's not possible to put two types in the GRES request, but you can ask for specific compute capability, for example you don't want a compute capability smaller than 6.0:

```
#SBATCH --gres=gpu:1
#SBATCH --constraint="COMPUTE_CAPABILITY_6_0|COMPUTE_CAPABILITY_6_1"
```

Resources :

- P100 specifiations
- Titan x (pascal) specifications
- RTX 2080 Ti (turing) specifications
- Generic Resource (GRES) Scheduling in SLURM

Octave

Usage:

```
octave
```

OpenCL

You can use OpenCL on CPU. To compile your software, please proceed as following:

```
gcc  -I/opt/intel/opencl-1.2-sdk-6.3.0.1904/include/ -
L/opt/intel/opencl-1.2-sdk-6.3.0.1904/lib64/ -Wl,-
rpath,/opt/intel/opencl/lib64/ -lOpenCL -o hello hello.c
```

Palabos

Usage:

See http://www.palabos.org

Install:

Download the latest palabos version on http://www.palabos.org

```
wget http://www.palabos.org/images/palabos_releases/palabos-v1.3r0.tgz
tar xzf palabos-v1.3r0.tgz
cd palabos-v1.3r0/examples/benchmarks/cavity3d
make
```

If you want to use more cores to compile it, edit the Makefile of your project and change the line :

```
Cons     = $(palabosRoot)/scons/scons.py -j 2 -f $(palabosRoot)/SConstruct
```

Replace the number 2 by the number of cores that you have at your disposal.

For example if you want to use a full 16 core node to do that, you can compile Palabos like this:

```
srun --cpus-per-task 16 --ntasks=1 --exclusive make
```

Distant Paraview

Thanks to Orestis for this tutorial.

Warning

1. Do not use at any point X11 forwarding it will be done by paraview itself. 2. You must have the SAME version of paraview on your local machine and on baobab (5.3.0).

Baobab connection:

```
ssh baobab2.hpc.unige.ch
```

Get some ressources in interactive mode (4 cores here, you can add a lot more options here if you want). You can even ask for gpu nodes (more on this afterwards):

```
salloc -n 4
```

Determine on which node your ressources are affacted (here node001):

```
echo $SLURM_JOB_NODELIST

SLURM_JOB_NODELIST=node001
```

On another terminal (in your pc) open an ssh tunnel to your NODE (in this case node001):

```
ssh -L 11150:node001:11111 baobab2.hpc.unige.ch
```

On the first terminal load paraview (if not loaded by default) and launch pserver:

```
module load foss/2016b ParaView/5.3.0-mpi
srun pvserver --server-port=11111
```

If you asked for GPU nodes the command is slightly different:

```
srun pvserver --server-port=11111 -display :0.0 --use-offscreen-rendering
```

On your local machine launch paraview and click on Connect. There you should find the menu to add a server. Put the name you want in Name, leave Client/Server as Server Type, and localhost in Host. The only very important configuration is the port which should be 11150 (the same number as in 11150:node001 from before). Save the configuration (click on Configure and Save, leave startup as Manual) and the click on Connect. The remote paraview session should start immedtiatly. There should be an error message: *Display is not accessible on the server side. Remote rendering will be disabled*. This message is normal.

Python

Default Python version on the cluster is Python 2.7.5.

You can have access to modern Python through Module - lmod

The Python version provided by module come with a lot of packages already installed.

You can check with module spider what are the packages provided. Example:

```
ml spider Python/3.7.4
[...]
Included extensions
==================
alabaster-0.7.12, asn1crypto-0.24.0, atomicwrites-1.3.0, attrs-19.1.0,
Babel-2.7.0, bcrypt-3.1.7, bitstring-3.1.6, blist-1.3.6, certifi-2019.9.11,
cffi-1.12.3, chardet-3.0.4, Click-7.0, cryptography-2.7, Cython-0.29.13,
deap-1.3.0, decorator-4.4.0, docopt-0.6.2, docutils-0.15.2, ecdsa-0.13.2,
future-0.17.1, idna-2.8, imagesize-1.1.0, importlib_metadata-0.22,
ipaddress-1.0.22, Jinja2-2.10.1, joblib-0.13.2, liac-arff-2.4.0,
MarkupSafe-1.1.1, mock-3.0.5, more-itertools-7.2.0, netaddr-0.7.19,
netifaces-0.10.9, nose-1.3.7, packaging-19.1, paramiko-2.6.0,
pathlib2-2.3.4,
paycheck-1.0.2, pbr-5.4.3, pip-19.2.3, pluggy-0.13.0, psutil-5.6.3,
py-1.8.0,
py_expression_eval-0.3.9, pyasn1-0.4.7, pycparser-2.19, pycrypto-2.6.1,
Pygments-2.4.2, PyNaCl-1.3.0, pyparsing-2.4.2, pytest-5.1.2, python-
dateutil-2.8.0, pytz-2019.2, requests-2.22.0, scandir-1.10.0,
setuptools-41.2.0, setuptools_scm-3.3.3, six-1.12.0, snowballstemmer-1.9.1,
Sphinx-2.2.0, sphinxcontrib-applehelp-1.0.1, sphinxcontrib-devhelp-1.0.1,
sphinxcontrib-htmlhelp-1.0.2, sphinxcontrib-jsmath-1.0.1, sphinxcontrib-
qthelp-1.0.2, sphinxcontrib-serializinghtml-1.1.3, sphinxcontrib-
websupport-1.1.2, tabulate-0.8.3, ujson-1.35, urllib3-1.25.3,
virtualenv-16.7.5, wcwidth-0.1.7, wheel-0.33.6, xlrd-1.2.0, zipp-0.6.0
```

If you need numpy, SciPy, pandas, mpi4py, they are provided by the SciPy-bundle module.

Example to load a recent version of Python with SciPy:

```
ml GCC/8.2.0-2.31.1 OpenMPI/3.1.3 Python/3.7.2 SciPy-bundle/2019.03
```

Custom Python lib

If you need to install a python library or a different version of the ones already installed, virtualenv is the solution.

Python-virtualenv is installed on Baobab http://www.virtualenv.org/en/latest/

Begin by loading a version of python using module (see above)

Create a new virtualenv if it's not already existing (put it where you want and name it like you want):

```
virtualenv --no-site-packages ~/baobab_python_env
```

This will create a directory named baobab_python_env in your home directory.

Every time you want to use your virtualenv, you should activate it first:

```
. ~/baobab_python_env/bin/activate
```

Install all the needed packages in the environment:

```
~/baobab_python_env/bin/pip install mpi4py
```

Use your new environment:

```
~/baobab_python_env/bin/python
```

R project and RStudio

The latest version of R (if it's not the latest, you can ask us to install it) is installed on the cluster.

Please see here for an sbatch example with R. You will find as well an exemple using the R package parallel.

RStudio

RStudio is IDE (Integrated Development Environment) for R, basically a more user friendly version than plain R.

With the Baobab upgrade to CentOS 7 (cf. https://hpc-community.unige.ch/t/baobab-migration-from-centos6-to-centos7/361) we do not provide anymore a central RStudio.

Instead, you can download the upstream Open Source binary RStudio Desktop version (cf. https://rstudio.com/products/rstudio/download/) and directly use it, here the instructions:

1. install it in your ${HOME} folder:

```
capello@login2:~$ mkdir Downloads
capello@login2:~$ cd Downloads
capello@login2:~/Downloads$ wget ${URL_FOR_rstudio-${VERSION}-x86_64-
fedora.tar.gz}
[...]
capello@login2:~/Downloads$ tar axvf rstudio-${VERSION}-x86_64-
fedora.tar.gz
[...]
capello@login2:~/Downloads$
```

2. launch an interactive graphical job:
    1. connect to the cluster using `ssh -Y` from a machine with an X server (see GUI access / Desktop with X2Go).
    Attention: If you work within X2Go, please connect again to the login node from within X2Go, thus:

```
capello@login2:~$ ssh -Y baobab2.hpc.unige.ch
[...]
 monitoring and documentation: http://baobab.unige.ch
```

```
 support: hpc@unige.ch
capello@login2:~$
```

2. start an interactive session on a node (see Interactive Slurm session):

```
capello@login2:~$ salloc -p debug-EL7 -n 1 -c 16 srun --pty $SHELL
salloc: Pending job allocation 39085914
salloc: job 39085914 queued and waiting for resources
salloc: job 39085914 has been allocated resources
salloc: Granted job allocation 39085914
capello@node001:~$
```

Doing so, you will have 16 cores on one node of the partition debug-EL7 for a max time of 15 minutes. Specify the appropriate duration time, partition, etc. like you would do for a normal job.

3. load one of the R version supported by RStudio, for example:

```
capello@node001:~$ module spider R/3.6.0

----------------------------------------------------------------------------
----------------
  R: R/3.6.0
----------------------------------------------------------------------------
----------------
    Description:
      R is a free software environment for statistical computing
and
      graphics.


    You will need to load all module(s) on any one of the lines
below
    before the "R/3.6.0" module is available to load.

      GCC/8.2.0-2.31.1  OpenMPI/3.1.3
[...]
capello@node001:~$ module load GCC/8.2.0-2.31.1  OpenMPI/3.1.3
capello@node001:~$ module load R/3.6.0
capello@node001:~$
```

4. run RStudio :

```
capello@node001:~$ ~/Downloads/rstudio-${VERSION}/bin/rstudio
```

R packages

You can install R packages as a user. Just follow once the steps given below:

Create a file named .Rprofile (note the dot in front of the file) in your home directory with the

following content:

```
cat(".Rprofile: Setting Switzerland repository\n")

local({
r = getOption("repos") # hard code the Switzerland repo for CRAN
r["CRAN"] = "https://stat.ethz.ch/CRAN/"
options(repos = r)
})
```

Create a file named .Renviron (note the dot in front of the file) in your home directory with the following content:

```
R_LIBS=~/Rpackages/
```

Create a directory where to store the installed R packages:

```
mkdir ~/Rpackages
```

Once done, make sure you have loaded R with module. Then, from a R command line you can install a R package.

For R version 3.5 and below :

```
install.packages("ggplot2")
```

For R version 3.6 or above :

```
Sys.setenv(R_INSTALL_STAGED = FALSE)
install.packages("ggplot2")
```

Use your newly installed package:

```
library(ggplot2)
```

Singularity

Singularity is a Docker like for HPC. It is available through module. :

```
module load GCC/6.3.0-2.27 Singularity/2.4.2
```

As you don't have root access on Baobab, you cannot build recipe file on Baobab. If you need to do this, you may want to build the image on your own machine and transfer the image to Baobab.

You can download existing images from shub (singularity hub) or from docker (docker public or private registry).

The image will be converted to a read-only squashfs on Baobab disk. It's not possible to have

writeable images as user. Instead you should build the image as sandbox.

Example to download an existing docker image and build a squashf image from it:

```
singularity build lolcow.simg docker://godlovedc/lolcow
```

Then you will end-up with a singularity image named `lolcow.simg`.

Singularity images are immutable.

Once you have a singularity image, you can do various things:

- run to run a container (the script named singularity at the root of the image, proceed as follow

  ```
  singularity run lolcow.simg
  ```

When you import a docker image, the ENTRYPOINT is used to create the singularity run script.

- exec to execute a single command inside the container.
- shell to launch a container and to spawn a shell (/bin/bash)

Download an image from a custom registry:

```
singularity build docker://registry.gitlab.com/flowkit/webservice/compute
```

For more examples, please check the following posts :

- [tutorial] launch openpose with GPU support through Singularity
- Help needed running MPI/Palabos software using Singularity
- Quick guide to Docker, singularity and shifter

Stata

Stata version 13 mp 16 and version 14 mp 24 are available on the cluster.

To use it, you need to add the stata path to your `PATH`:

```
module load stata/13mp16
```

or :

```
module load stata/14mp24
```

The Stata binaries are stata-mp or xstata-mp for the graphical interface.

If you need a graphical interactive session, please proceed as follows:

```
salloc -n1 -c 16 --partition=debug-EL7 --time=15:00 srun -n1 -N1 --pty
$SHELL
```

```
xstata-mp
```

Doing so will launch a graphical Stata on a debug node with 16 cores for a 15 minutes session. See on this document for other partition/time limits.

Hint

Please keep in mind that the cluster may be full and that you will have to wait until the resources are allocated to you. It's best to launch stata in batch mode.

To launch stata in batch mode, see the Multithreaded jobs section and specify that you want one task and n cores.

Please see here for an sbatch example with stata.

TensorFlow

Please see TensorFlow on gitlab for some examples and scripts related to TensorFlow.

ATTENTION , the module:TensorFlow we provide are compiled with GPU support (see https://www.tensorflow.org/install/source#gpu_support), thus they must be used on a GPU partition (cf. Nvidia CUDA section).

You can also read this post :

- Getting started with TensorFlow on Baobab (ImportError: libcuda.so.1)

Theano

Please see Theano on gitlab for some examples and scripts related to Theano

Install an application in your /home

Example with cudimot

This is the kind of software that can be installed directly by the user, so here is an example you can use for inspiration if you need to install another software.

Source

Cudimot NODDI Bingham Installation

```
export CUDIMOT=${HOME}/cudimot
```

```
mkdir -p $CUDIMOT/bin
cd $CUDIMOT
wget
http://users.fmrib.ox.ac.uk/~moisesf/cudimot/NODDI_Bingham/CUDA_9.1/NODDI_Bi
ngham.zip
unzip NODDI_Bingham.zip
cp NODDI_Bingham/bin/* $CUDIMOT/bin
```

Usage

Create an sbatch script as usual. As Cudimot needs or can use cuda, you need to use a node with GPU.

```
#SBATCH --time=XXX # to adapt
#SBATCH --cpus-per-task=XX # to adapt.
#SBATCH --partition=shared-gpu-EL7
#SBATCH --gres=gpu:1

export CUDIMOT=${HOME}/cudimot

ml GCC/7.3.0-2.30  OpenMPI/3.1.1 FSL/5.0.11

srun $CUDIMOT/bin/Pipeline_NODDI_Bingham.sh [SubjectDirectory]
```

About the `#SBATCH –cpus-per-task=X` line, if unsure : - start with 1, measure the run time. - set it to 2, measure run time. If the run is almost twice as fast, you can continue to increase. If not, stick to 1.