Basic tools on HPC clusters

Imad Kissami, Nouredine Ouhaddou, Mouad Elalj, Mostafa Mamouni, Noureddine Hamid
Mohammed VI Polytechnic University, Benguerir, Morocco

Outline of this training

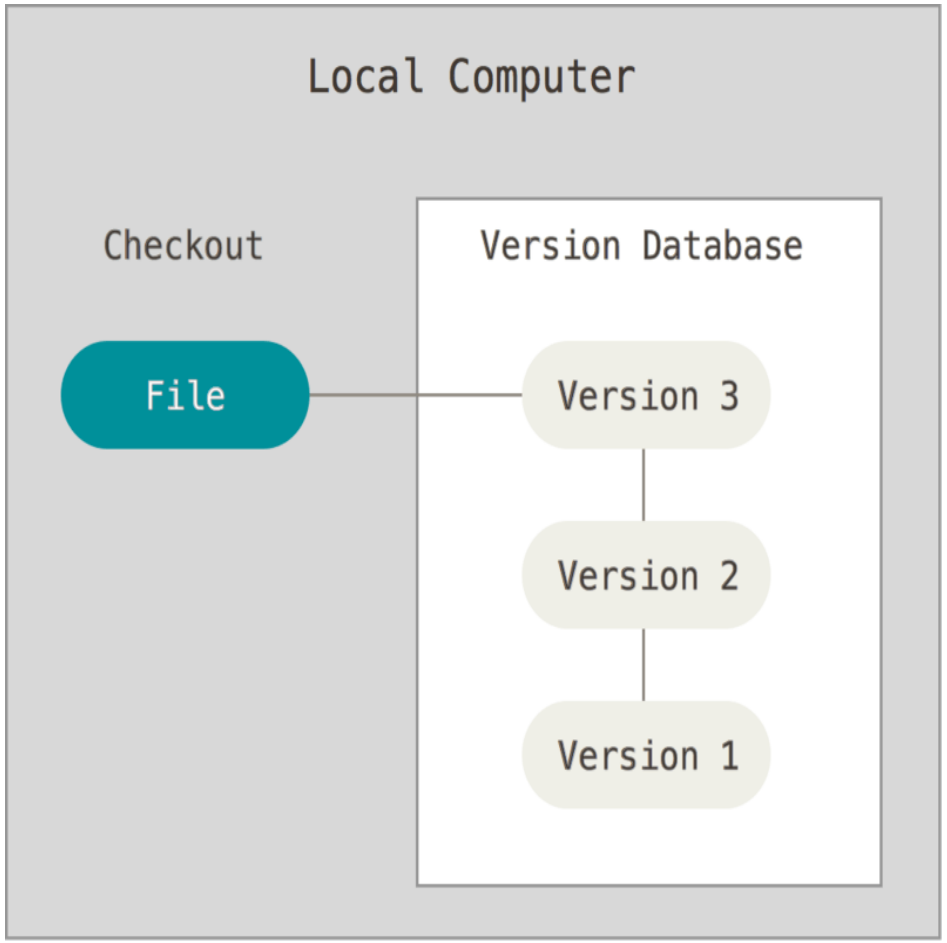
- Git
- Ssh on Unix/Windows
- Module commands
- Slurm commands
- Futures

Git

What is version control?

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

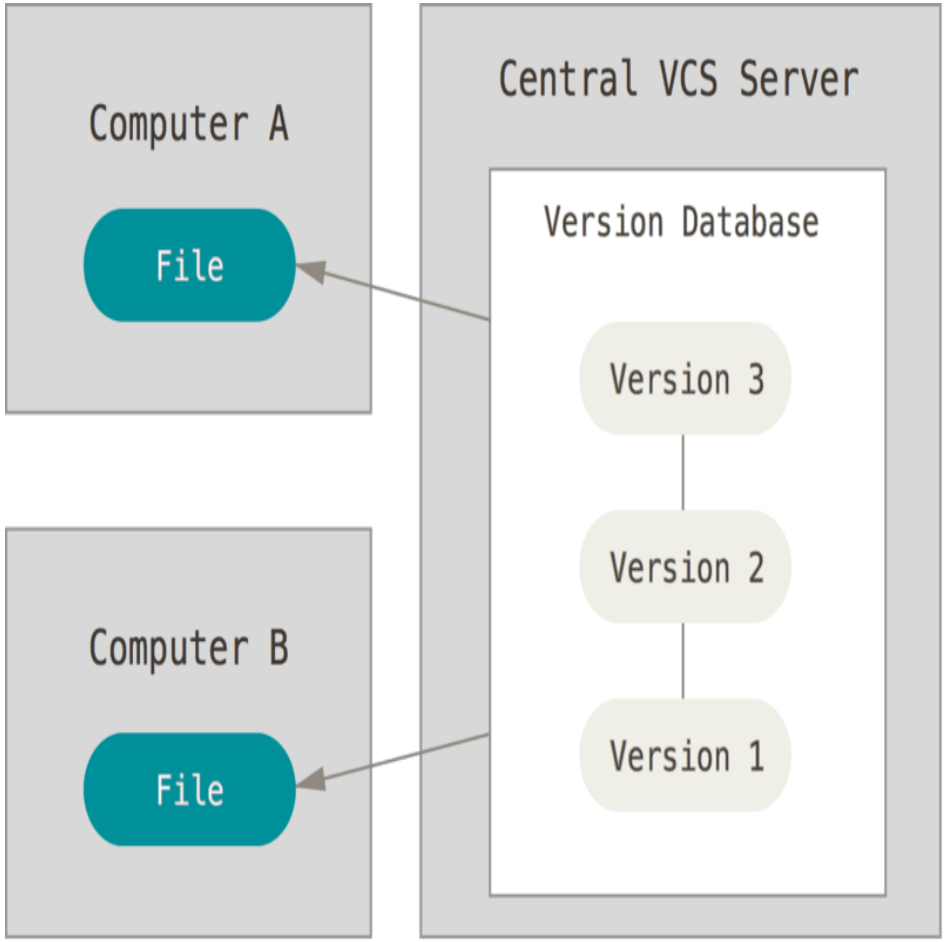
- Local Version Control Systems



Git

What is version control?

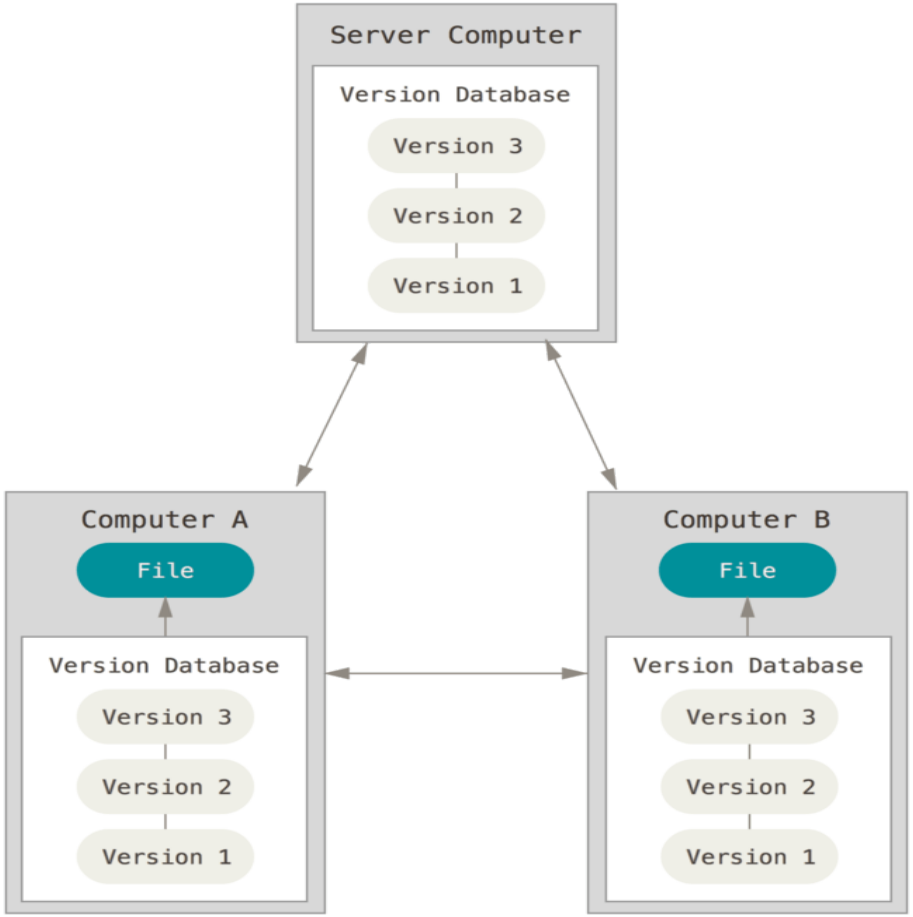
- Centralized Version Control Systems



Git

What is version control?

- Distributed Version Control Systems



Git

What is Git?

Git is the most commonly used version control system. Git tracks the changes you make to files, so you have a record of what has been done, and you can revert to specific versions should you ever need to.

- Tracking code changes
- Tracking who made changes
- Coding collaboration

Git

Why Git?

- Over 70% of developers use Git!
- Developers can work together from anywhere in the world.
- Developers can see the full history of the project.
- Developers can revert to earlier versions of a project.

Git

What is GitHub?

- Git is not the same as GitHub.
- GitHub makes tools that use Git.
- GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.
- In this tutorial, we will focus on using Git with GitHub.

Git

Getting started: Install Git and create a GitHub account

- CentOS, RedHat:

```
sudo yum install git
sudo yum update git
```

- Debian, Ubuntu:

```
sudo apt-get install git
sudo apt-get update git
```

- MacOS, use Homebrew:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
brew install git
brew upgrade git
```

- Windows: download Git for Windows and install it. Also, this tutorial utilizes a Bash command line interface, therefore, you should use Git Bash, which is a part of the Git installation package for Windows.
- On SIMLAB (version 2.23.0):

```
module load git
```

Git

Getting started: Create Github account

If you don't have Github account yet, go to 'github.com/signup' and follow the instructions to create your account.

Setting up a repository

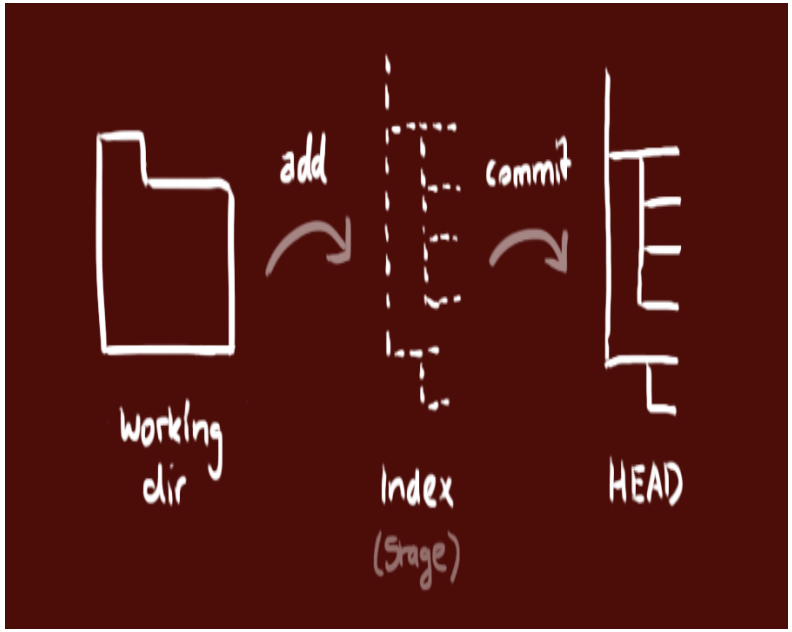
1. Let Git know who you are:

```
git config --global user.name "Nouredine Ouhaddou"
git config --global user.email "Nouredine.Ouhaddou@um6p.ma"
```

2. Initialize Git:
   * 'git init': to create a new git repository
   * 'git clone': create a working copy of a local repository
3. Adding a remote repository
   * 'git remote add' command takes two arguments:
     - A remote name, for example, origin
     - A remote URL, for example, https://github.com/user/repo.git
     - ex: 'git remote add origin https://github.com/user/repo.git'

Getting started: Workflow

Saving changes
your local repository consists of three "trees" maintained by git. the first one is your 'Working Directory' which holds the actual files. the second one is the 'Index' which acts as a staging area and finally the 'HEAD' which points to the last commit you've made.



* 'git add': To add a file to the staging area
* 'git commit': commits a change set from the working directory into the repository
* 'git log': To check your commit history
* 'git push': To upload local repository content to a remote repository.

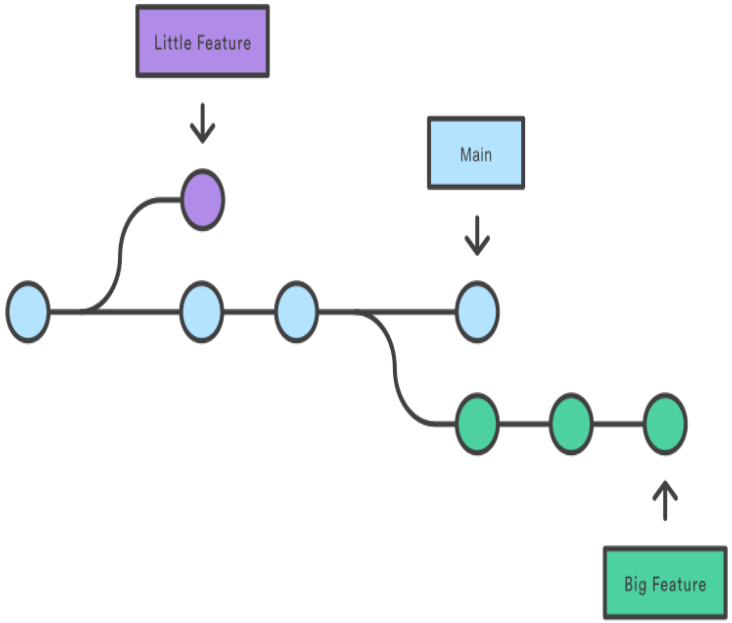
Undoing commits/changes

* ‘git clean‘: git clean is a convenience method for deleting untracked files in a repo’s working directory.
* ‘git revert‘ : invert the changes introduced by the commit and appends a new commit with the resulting inverse content
* ‘git reset‘: is the command we use when we want to move the repository back to a previous commit, discarding any changes made after that ‘commit‘.
* ‘git rm‘ git rm is used to remove a file from a Git repository. It is a convenience method that combines the effect of the default shell rm command with git add . This means that it will first remove a target from the filesystem and then add that removal event to the staging index. repository.

Git

Getting started: Git Branch

In Git, branches are a part of your everyday development process. Git branches are effectively a pointer to a snapshot of your changes. When you want to add a new feature or fix a bug—no matter how big or how small—you spawn a new branch to encapsulate your changes.



* ‘git branch $<$ *branch* $>$‘
* ‘git branch -d $<$ *branch* $>$‘ Safe mode for delete a specified branch.
* ‘git branch -m $<$ *branch* $>$‘ Rename the current branch to $<$ *branch* $>$
* ‘git branch -a‘ List all remote branches.
* ‘git checkout‘ The git checkout command lets you navigate between the branches created by git branch.
* ‘git merge new-feature‘ merge the new feature branch into the main branch.

Git

Getting started: Syncing with remote repository

* git remote‘ : The git remote command lets you create, view, and delete connections to other repositories.
  - ‘git remote -v‘ list the remote connections
  - ‘git remote add‘ Adding Remote Repositories
* ‘git fetch‘ downloads commits, files, and refs from a remote repository into your local repo.
* ‘git pull‘ to update your current HEAD branch with the latest changes from the remote server.

SSH

Ssh using shell commands

- Connect to the remote host:

```
$ ssh -CY <login>@simlab-cluster.um6p.ma
```

Connecting directly to the cluster frontends is restricted to networks within the university. So being connected to the university network is needed, or using a VPN, then you can connect via this command.

- Configure '~/.ssh/config' file:

```
$ cat ~/.ssh/config
host simlab
    hostname simlab-cluster.um6p.ma
    user <username>
    Compression yes
    ForwardX11 yes
```

- Now you can connect directly using:

```
ssh simlab
```

SSH

Ssh using shell commands

- Generating private and public keys:

```
$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/machine-locale/login/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/machine-locale/login/.ssh/id_rsa.
Your public key has been saved in /home/machine-locale/login/.ssh/id_rsa.pub.
The key fingerprint is:
26:e3:d4:29:b7:5b:29:15:d7:68:39:eb:a3:12:0b:02 login@machine-locale.domaine.fr
```

Two files ('id_rsa' and 'id_rsa.pub') are created in the .ssh directory of the local machine.

- Transferring the file containing your public key to the distant machine

```
$ ssh-copy-id <username>@simlab-cluster.um6p.ma
```

Now you can connect without entering the password

SSH

Copy data to/from the remote host

- Copy data to the remote host:
  - using scp:

```
$ scp -r <filename>  <login>@simlab-cluster.um6p.ma:<remote_directory>
```

  - using rsync:

```
$ rsync -avz <filename> simlab:<remote_directory>
```

- Copy data from the remote host:
  - using scp:

```
$ scp -r <remote_dir> <login>@simlab-cluster.um6p.ma:<path/to/filename>
```

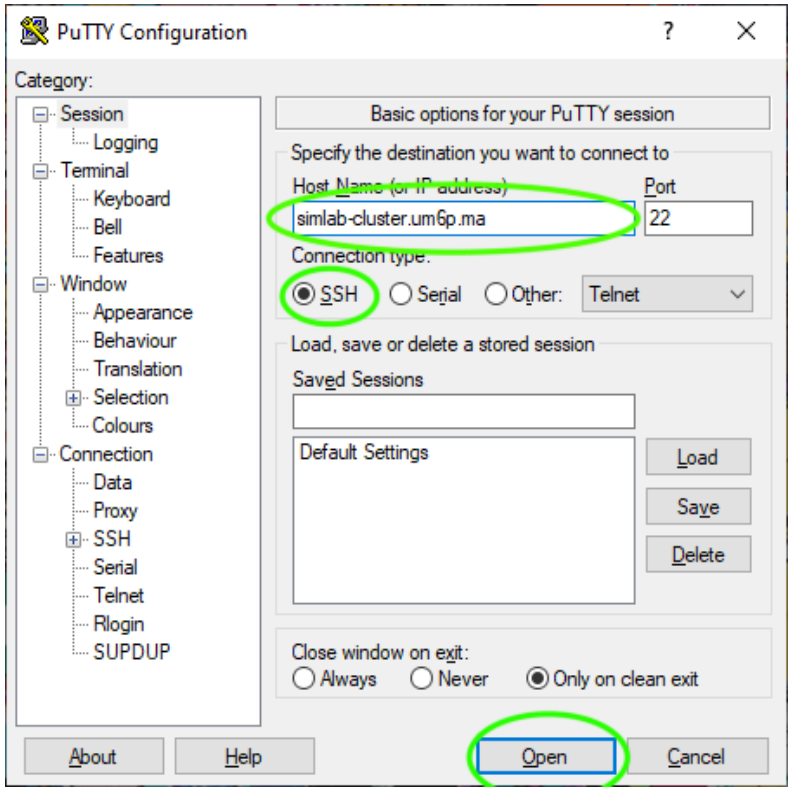
  - using rsync:

```
$ rsync -avz <remote_dir> <login>@simlab-cluster.um6p.ma:<path/to/filename>
```

SSH

Ssh using Putty

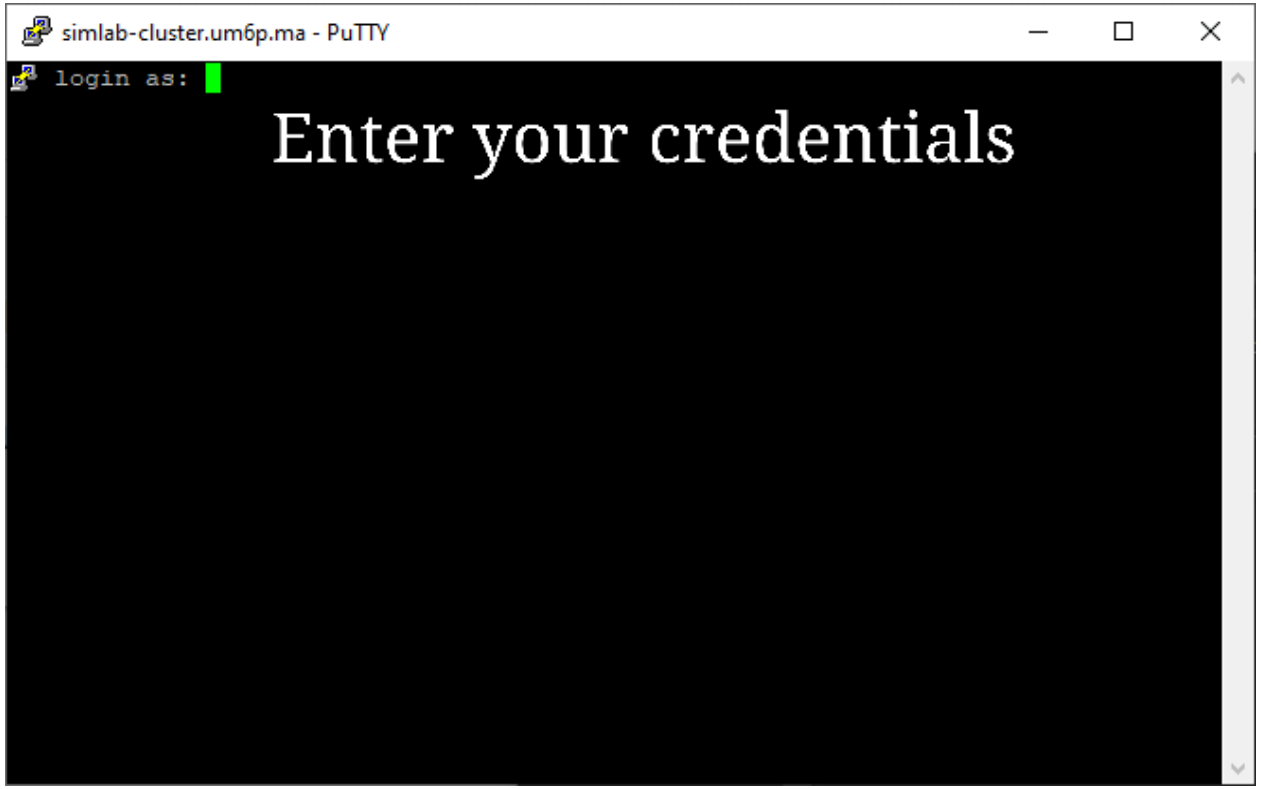
- PuTTY is a free and open-source terminal emulator, serial console and network file transfer application.
- It supports several network protocols, including SCP, SSH, Telnet, rlogin, and raw socket connection.
- Download: https://www.putty.org/



SSH

Ssh using Putty

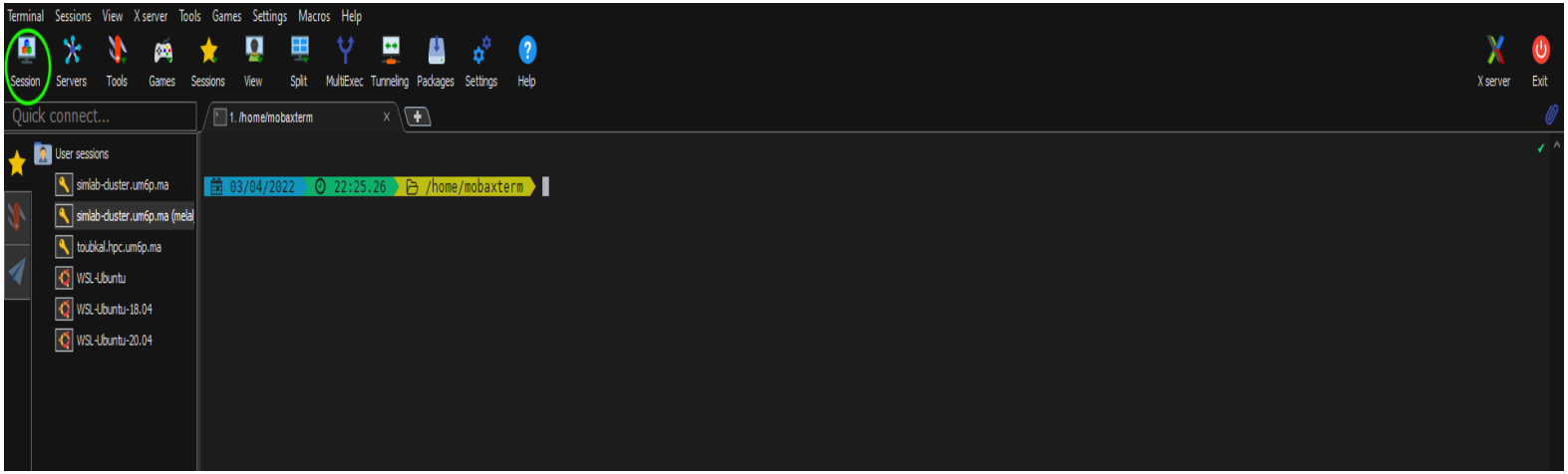
- Once we are in this windows we need to enter our login and password.
- After that we are successfully connected to the cluster.



SSH

Ssh using Mobaxterm

- Free X server for Windows with tabbed SSH terminal, telnet, RDP, VNC, Xdmcp, Mosh and X11-forwarding.
- Download: https://mobaxterm.mobatek.net/

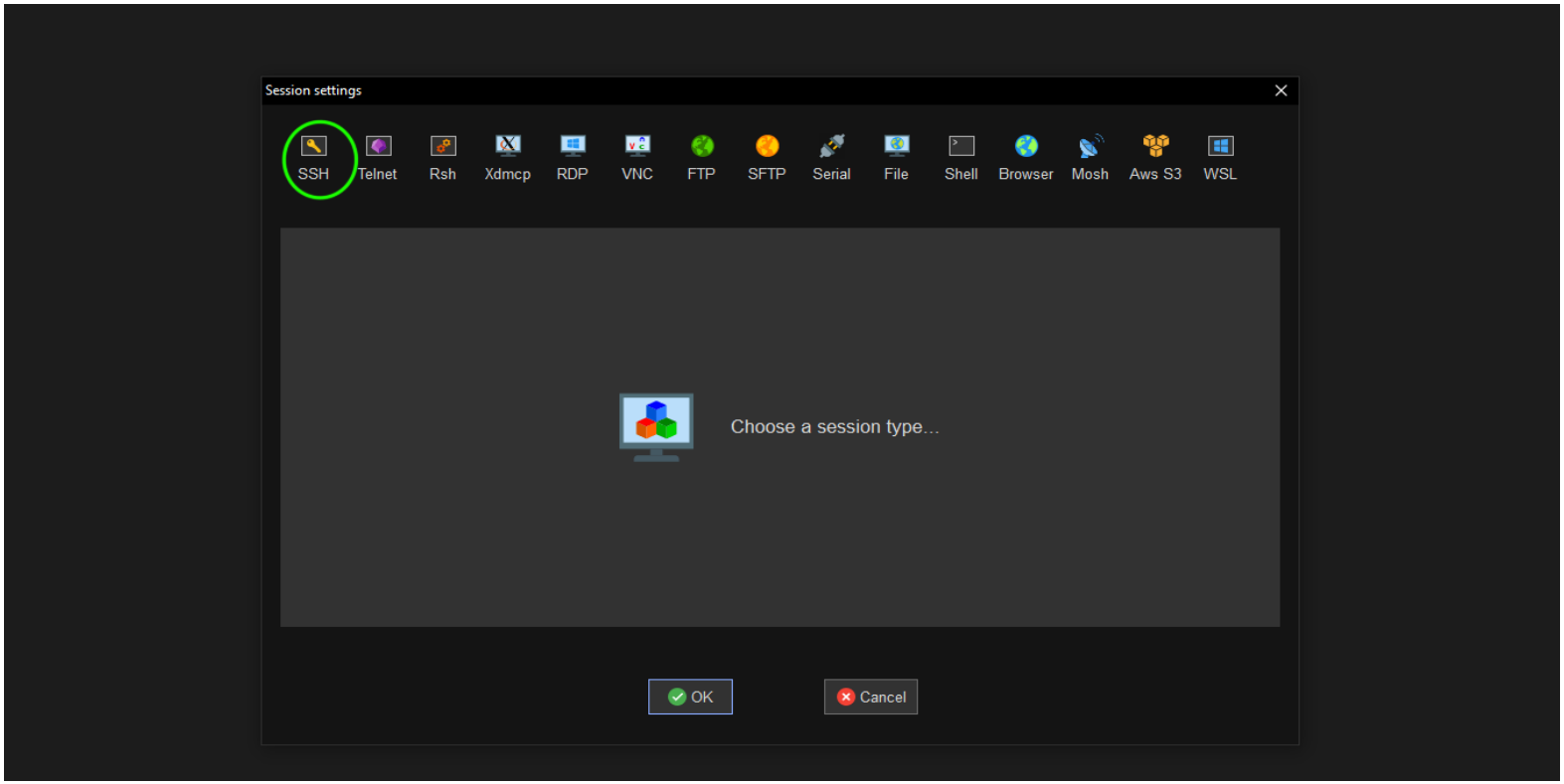


- To start configuring mobaxterm we can press the Session icon in top left.

SSH

Ssh using Mobaxterm

- Session setting:

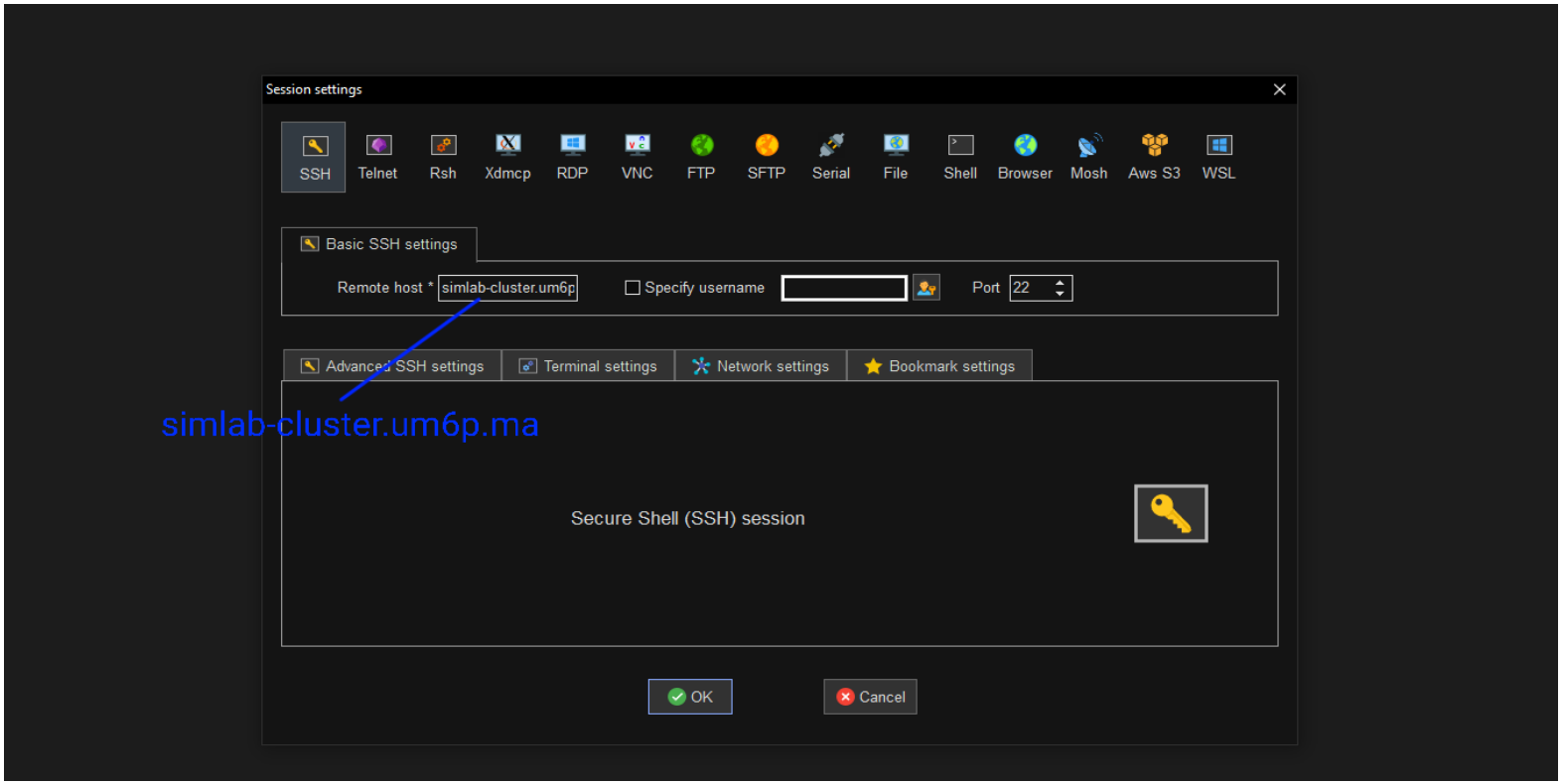


- We need now to enter the desired remote host that we want to connect into.
- We can specify our username so we are not required to write it every time.
- After we finish we press the OK button.

SSH

Ssh using Mobaxterm

- Session setting:

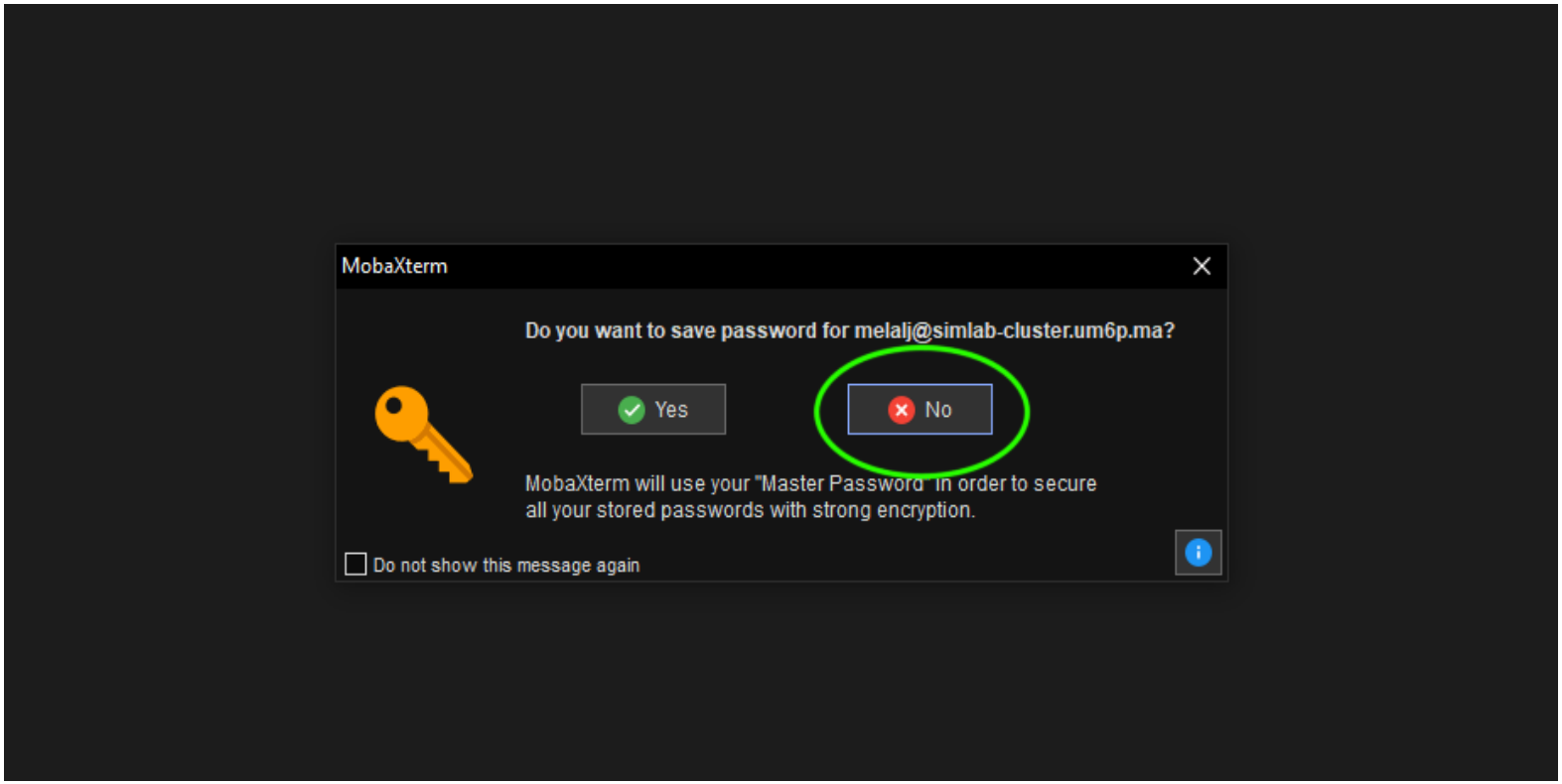


- This message will ask you if you want mobaxterm to save your password so it's not needed to be entered every time.
- You can choose No if you don't want to save your password and check the bottom left box to not get this message every time.

SSH

Ssh using Mobaxterm

- Session setting:



Module commands

Modules

- Displaying the installed modules:

```
$ module avail
------------- /cm/local/modulefiles -----------------------
cluster-tools-dell/8.1  freeipmi/1.5.7       lua/5.3.4    openldap
cluster-tools/8.1       gcc/7.2.0            module-git   openmpi/mlnx/gcc↩
     /64/3.1.1rc1
cmd                     intel/mic/sdk/3.8.4  module-info  shared
dot                     ipmitool/1.8.18      null

------------ /cm/shared/modulefiles -------------------------
ABINIT/8.10.1                              libpng12/gcc/1.2.56
ABINIT/9.4.1-foss-2020b                    libreadline/8.0-GCCcore-8.3.0
abinit/gcc/64/8.10.1                       libreadline/8.0-GCCcore-9.3.0
acml/gcc-int64/64/5.3.1                    libreadline/8.0-GCCcore-10.2.0
[...]
```

- Load a module (default one):

```
$ module load GCC
```

- List the loaded modules:

```
$ module list
Currently Loaded Modulefiles:
1) GCCcore/10.2.0                3) binutils/2.35-GCCcore-10.2.0
2) zlib/1.2.11-GCCcore-10.2.0    4) GCC/10.2.0
```

Module commands

Modules

- Unload all loaded modules:

```
$ module purge
```

- Load more than module:

```
$ module unload UXC
$ module list
Currently Loaded Modulefiles:
1) GCCcore/10.2.0                    4) GCC/10.2.0
2) zlib/1.2.11-GCCcore-10.2.0        5) numactl/2.0.13-GCCcore-10.2.0
3) binutils/2.35-GCCcore-10.2.0
```

- Load GCC (version 9.3.0):

```
$ module load GCC/9.3.0
$ module list
Currently Loaded Modulefiles:
1) GCCcore/9.3.0                  3) binutils/2.34-GCCcore-9.3.0
2) zlib/1.2.11-GCCcore-9.3.0      4) GCC/9.3.0
```

Remark: All loaded modules have the same GCC version.

Module commands

Install modules: EasyBuild

- List available versions:

```
$ module avail EasyBuild
---------------- /cm/shared/modulefiles -------------------
EasyBuild/4.4.2
```

- Load EasyBuild (version 4.4.2)

```
$ module load EasyBuild
$ module list
Currently Loaded Modulefiles:
1) python/3.5.0   2) EasyBuild/4.4.2
```

- Searching for available easyconfigs files (gmsh)

```
$ eb --modules-tool EnvironmentModules --module-syntax Tcl -S gmsh
== found valid index for /cm/shared/apps/easybuild/4.4.2/lib/python3.5/site-↩
      packages/easybuild_easyconfigs-4.4.2-py3.5.egg/easybuild/easyconfigs, so ↩
      using it...
CFGS1=/cm/shared/apps/easybuild/4.4.2/lib/python3.5/site-packages/↩
      easybuild_easyconfigs-4.4.2-py3.5.egg/easybuild/easyconfigs
* $CFGS1/g/gmsh/gmsh-3.0.6-foss-2017b-Python-2.7.14.eb
* $CFGS1/g/gmsh/gmsh-3.0.6-foss-2018b-Python-3.6.6.eb
* $CFGS1/g/gmsh/gmsh-4.2.2-foss-2018b-Python-3.6.6.eb
[...]
```

Module commands

Install modules: EasyBuild

- Install module:

  $--robot$: to enable dependency resolution.
  $--detect-loaded-modules=error$: to print a clear error and stop when any (non-allowed) loaded modules are detected.
  $--detect-loaded-modules=purge$: to run module purge if any (non-allowed) loaded modules are detected.
  $--optarch=GENERIC$: to optimize for a generic processor architecture.

```
$ eb --modules-tool EnvironmentModules --module-syntax Tcl --prefix=/path/to/easybuild --robot --detect-loaded-modules=error --detect-loaded-modules=purge --optarch=GENERIC gmsh-4.7.1-foss-2020a-Python-3.8.2.eb
```

- List new modules:

```
$ module avail
module avail gmsh-4.7.1-foss-2020a-Python-3.8.2.eb
--------------- /path/to/easybuild ----------------
gmsh/4.7.1-foss-2020a-Python-3.8.2
```

Slurm commands

CPU code

- Simple Python example using MPI.

```python
from mpi4py import MPI

COMM = MPI.COMM_WORLD
np = COMM.Get_size(); id = COMM.Get_rank()

print("I am process", id, "of", np);
```

- Modules needed (modules loaded must be from the same GCC version):
  - Python
  - OpenMPI

  mpi4py is already installed in some Python versions. If not use:

```
module load Python/<version-x>
pip install mpi4py
```

Slurm commands

srun/sbatch: CPU code

- Submitting a job using srun:

```
srun -p visu -N 1 -n 1 --pty bash
[team1337@visu01 ~]$
```

- Submitting a job using sbatch:

```
$ cat run_mpi.slurm
#!/bin/bash
#SBATCH --partition=shortq        # partition name
#SBATCH --job-name=mpi            # name of the job
#SBATCH --nodes=1                 # number of nodes
#SBATCH --ntasks-per-node=4       # number of MPI processes per node
#SBATCH --time=00:01:00           # maximum execution time requested (HH:MM:SS)
#SBATCH --output=mpi%j.out        # name of output file
#SBATCH --error=mpi%j.out         # name of error file (here, in common with output)

# cleans out the modules loaded in interactive and inherited by default
module purge

# load modules
module load slurm Python/3.8.2-GCCcore-9.3.0 OpenMPI/4.0.3-GCC-9.3.0

mpirun python hello_mpi.py
```

Before submitting the job let's verify the nodes availability!

Slurm commands

sinfo

sinfo sinfo is used to view partition and node information for a system running Slurm.

- To see available nodes run:

```
$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
defq*        up    1:00:00      1   mix$ node04
defq*        up    1:00:00      1  maint node05
defq*        up    1:00:00      3    mix node[01-03]
gpu          up 2-00:00:00      7    mix node[06-07,09-10,13,16-17]
gpu          up 2-00:00:00      5   idle node[08,11-12,14-15]
shortq       up    4:00:00      1   mix$ node04
shortq       up    4:00:00      1  maint node05
shortq       up    4:00:00      3    mix node[01-03]
...
```

Now you could modify the partition name and/or choose the required node(s) for the test.

- To choose the list of nodes:

```
#SBATCH --nodelist=node01,node02
```

- Submit this script via the 'sbatch' command:

```
$ sbatch run_mpi.slurm
```

Slurm commands

squeue

squeue is used to view job and job step information for jobs managed by Slurm.

- View all runned jobs:

```
$ squeue
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
30599    shortq petsc_1l amal.mac PD       0:00      4 (Resources)
30689     longq       71 amoutaki PD       0:00      1 (Resources)
30690     longq       72 amoutaki PD       0:00      1 (Resources)
...
```

also, squeue provides some options to either simplify the output or get more details, those can be fully shown by ‘squeue –help‘.

- View only jobs for 'team1337' user (add $--name < job\_name >$ to view only jobs with this name):

```
$ squeue -u team1337
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
30979    shortq      mpi team1337 PD       0:00      2 (Priority)
30980       gpu      mpi team1337 PD       0:00      1 (Priority)
30981    shortq      mpi team1337 PD       0:00      1 (Priority)
```

  - To make it as default, add this line to '~/.bashrc' file:

```
emacs ~/.bashrc
alias squeue="squeue -u team1337"
```

Slurm commands

scontrol

scontrol is used to view or modify Slurm configuration and state.

- To obtain complete information about a job:

```
$ scontrol show job 30981
JobId=30981 JobName=mpi
UserId=team1337(1213) GroupId=team1337(1238) MCS_label=N/A
Priority=4294875888 Nice=0 Account=(null) QOS=normal
JobState=PENDING Reason=Priority Dependency=(null)
Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
RunTime=00:00:00 TimeLimit=00:01:00 TimeMin=N/A
SubmitTime=2022-04-04T16:07:32 EligibleTime=2022-04-04T16:07:32
StartTime=2022-04-09T13:10:17 EndTime=2022-04-09T13:11:17 Deadline=N/A
PreemptTime=None SuspendTime=None SecsPreSuspend=0
LastSchedEval=2022-04-04T16:07:34
Partition=shortq AllocNode:Sid=frontend01:116373
ReqNodeList=(null) ExcNodeList=(null)
NodeList=(null)
NumNodes=1-1 NumCPUs=4 NumTasks=4 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
TRES=cpu=4,node=1
Socks/Node=* NtasksPerN:B:S:C=4:0:*:* CoreSpec=*
MinCPUsNode=4 MinMemoryNode=0 MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
Gres=(null) Reservation=(null)
OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
Command=/home/team1337/training_bench/batch.slurm
WorkDir=/home/team1337/training_bench
StdErr=/home/team1337/training_bench/mpi30981.out
StdIn=/dev/null
StdOut=/home/team1337/training_bench/mpi30981.out
Power=
```

Slurm commands

scancel

scancel is used to kill the jobs.

- To kill a job, run:

```
$ scancel JOBID
```

- To cancel all jobs for a user:

```
$ scancel -u <username>
```

- To cancel all pending jobs for a user:

```
$ scancel -t PENDING -u <username>
```

Slurm commands

GPU code

- Simple Python example using Cupy.

```python
import cupy as cp

arr_g1= cp.random.rand(10000,10000)
arr_g2= cp.random.rand(10000,10000)
r1 = arr_g1 * arr_g2
```

- Modules needed (modules loaded must be from the same GCC version):
  - Python and CUDA

  Cupy is already installed in some Python versions. If not use:

```
module load Python/<version-x>
pip install cupy-cuda111
```

Slurm commands

srun/sbatch: GPU code

- Submitting a job using srun:

```
srun -p gpu --gres=gpu:1 --mem=8g --constraint=V100 --pty bash
[team1337@gpu ~]$
```

- Submitting a job using sbatch:

```
$ cat cuda.slurm
#!/bin/bash
#SBATCH --partition=gpu                # partition name
#SBATCH --job-name=single_gpu          # name of job
#SBATCH --gres=gpu:1                   # number of GPUs
#SBATCH --nodes=1                      # we request one node
#SBATCH --ntasks-per-node=1            # with one task per node
#SBATCH --time=00:10:00                # max. exec. time requested (HH:MM:SS)
#SBATCH --mem=8g                      # memory required per node
#SBATCH --constraint=V100              # Type of GPU
#SBATCH --output=gpu_single%j.out      # name of output file
#SBATCH --error=gpu_single%j.out       # name of error file (here, in common ↩
      with the output file)

module purge
module load Python/3.8.6-GCCcore-10.2.0 CUDA/11.1.1-GCC-10.2.0
python Cupy_example.py
```

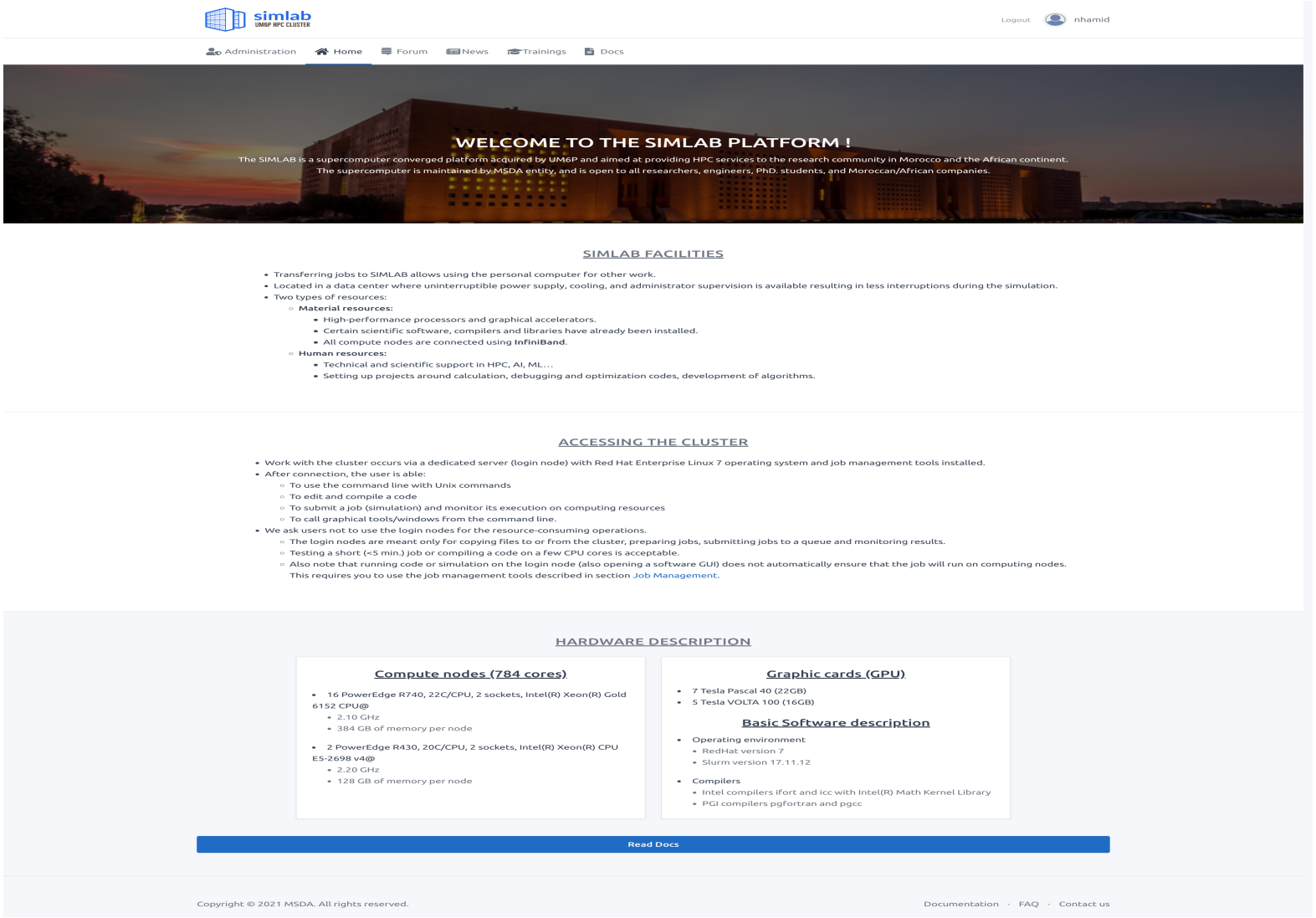
  - Only one GPU per node is allowed for the ‘gpu‘ partition

- Submit this script via the 'sbatch' command:

```
$ sbatch cuda.slurm
```

Futures

Simlab Website: Home page



simlab
UM6P HPC CLUSTER

Logout nhamid

Administration | Home | Forum | News | Trainings | Docs

WELCOME TO THE SIMLAB PLATFORM !

The SIMLAB is a supercomputer converged platform acquired by UM6P and aimed at providing HPC services to the research community in Morocco and the African continent.
The supercomputer is maintained by MSDA entity, and is open to all researchers, engineers, PhD. students, and Moroccan/African companies.

SIMLAB FACILITIES

- Transferring jobs to SIMLAB allows using the personal computer for other work.
- Located in a data center where uninterruptible power supply, cooling, and administrator supervision is available resulting in less interruptions during the simulation.
- Two types of resources:
  - **Material resources:**
    - High-performance processors and graphical accelerators.
    - Certain scientific software, compilers and libraries have already been installed.
    - All compute nodes are connected using **InfiniBand**.
  - **Human resources:**
    - Technical and scientific support in HPC, AI, ML...
    - Setting up projects around calculation, debugging and optimization codes, development of algorithms.

ACCESSING THE CLUSTER

- Work with the cluster occurs via a dedicated server (login node) with Red Hat Enterprise Linux 7 operating system and job management tools installed.
- After connection, the user is able:
  - To use the command line with Unix commands
  - To edit and compile a code
  - To submit a job (simulation) and monitor its execution on computing resources
  - To call graphical tools/windows from the command line.
- We ask users not to use the login nodes for the resource-consuming operations.
  - The login nodes are meant only for copying files to or from the cluster, preparing jobs, submitting jobs to a queue and monitoring results.
  - Testing a short (<5 min.) job or compiling a code on a few CPU cores is acceptable.
  - Also note that running code or simulation on the login node (also opening a software GUI) does not automatically ensure that the job will run on computing nodes. This requires you to use the job management tools described in section Job Management.

HARDWARE DESCRIPTION

Compute nodes (784 cores)

- 16 PowerEdge R740, 22C/CPU, 2 sockets, Intel(R) Xeon(R) Gold 6152 CPU@
  - 2.10 GHz
  - 384 GB of memory per node
- 2 PowerEdge R430, 20C/CPU, 2 sockets, Intel(R) Xeon(R) CPU E5-2698 v4@
  - 2.20 GHz
  - 128 GB of memory per node

Graphic cards (GPU)

- 7 Tesla Pascal 40 (22GB)
- 5 Tesla VOLTA 100 (16GB)

Basic Software description

- Operating environment
  - RedHat version 7
  - Slurm version 17.11.12
- Compilers
  - Intel compilers ifort and icc with Intel(R) Math Kernel Library
  - PGI compilers pgfortran and pgcc

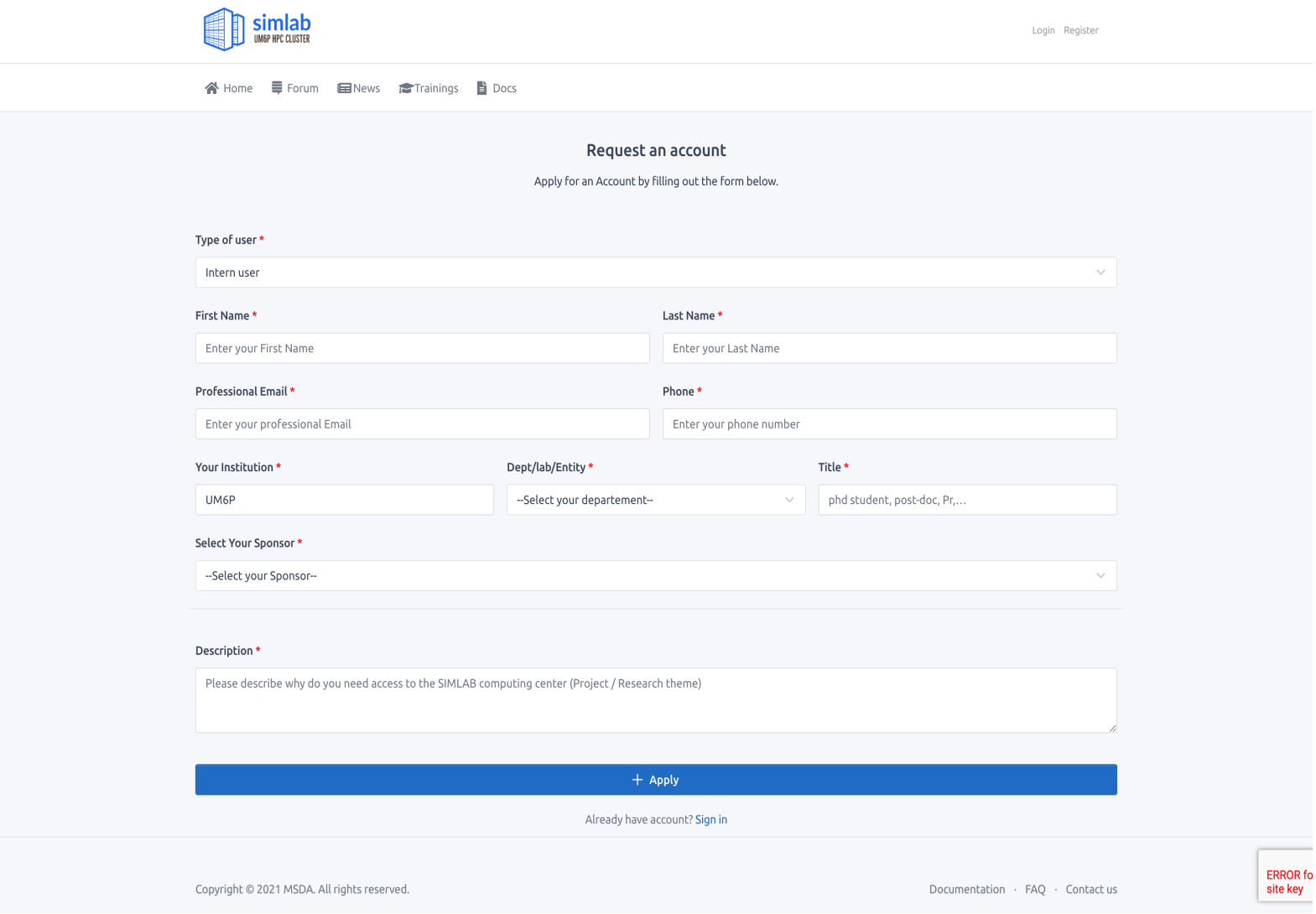
Read Docs

Copyright © 2021 MSDA. All rights reserved.

Documentation · FAQ · Contact us

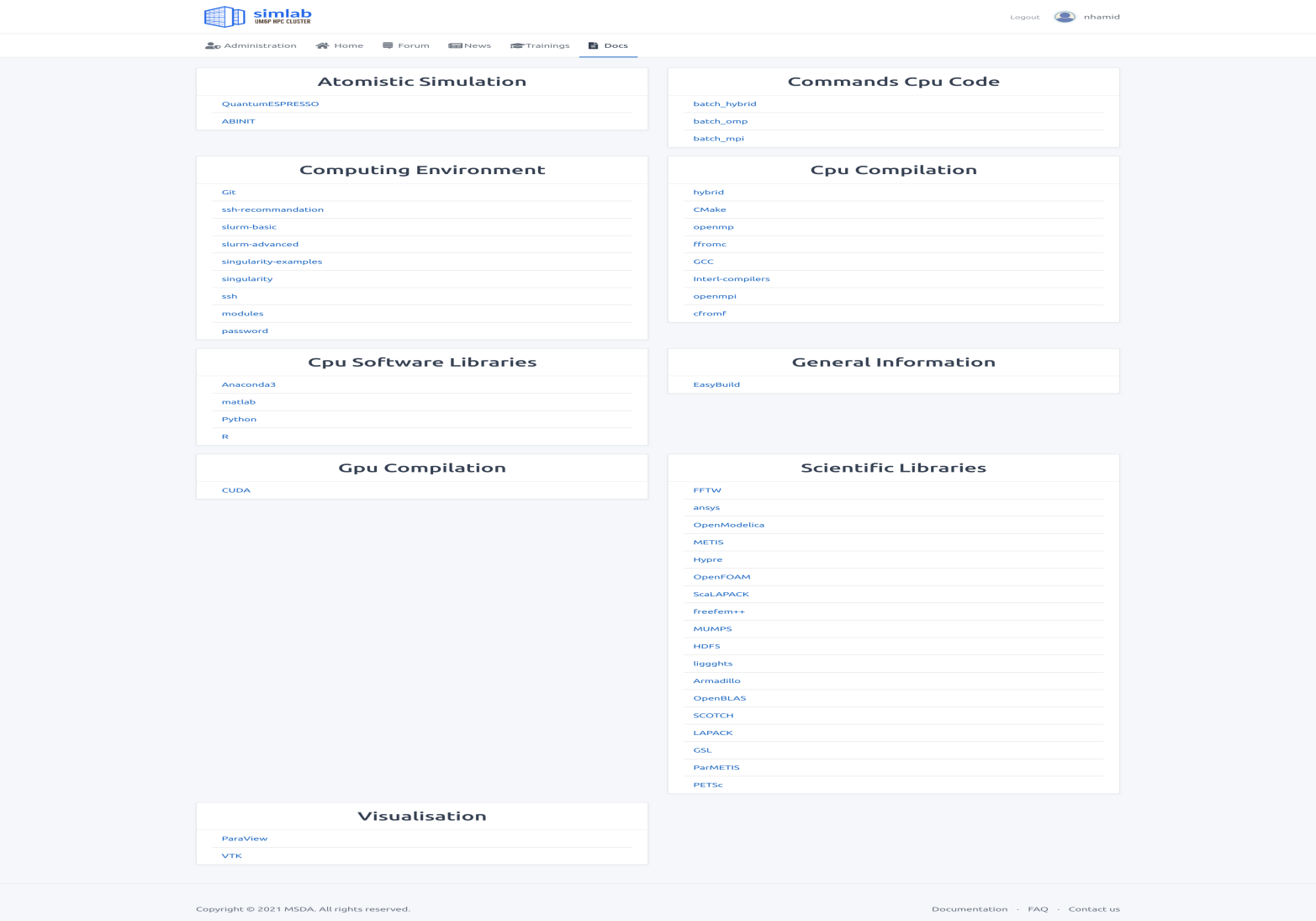
Futures

Simlab Website: Account request



Futures

Simlab Website: Documentation



Futures

Simlab Website: Issues

simlab
UM6P HPC CLUSTER

Logout nhamid

Administration Home Forum News Trainings Docs

reporting an issue

Closed Issue

By Noureddine Hamid (Jan. 24, 2022, 5:55 p.m.) Edit Delete

Mark as closed Mark as open Submit

~~Describe the bug~~ (resolved)

When checking type compatibility some exceptions are in place for `AugAssign` s. This is done to allow expressions such as:

```
a = 2.5
a += 3
```

as `AugAssign` s are handled in the same way as `Assign` s and `a = 3` would not be allowed. However these exceptions mean that the correct type check is not done

To Reproduce
Provide code to reproduce the behavior:

```
a = 2
a += 3.5

if __name__ == '__main__':
    print(a)
```

This code compiles and produces the following output:
`bash/fortran/c 5`

COMMENTS (1)

By Noureddine Hamid Jan. 25, 2022, 11:04 a.m.

The code above should raise an error. The simplest way to do this is to save `a+=3.5` as `a=a+3.5` however this may lead to less readability in languages that support `AugAssign` . That said, given that Fortran, our standard translation language, doesn't support `AugAssign` this may not be a big loss.

Write a Comment:

Submit Comment

Documentation · FAQ · Contact us

Thanks for your attention

Questions?