

Subscribe to DeepL Pro to edit this document.  
Visit [www.DeepL.com/pro](https://www.deepl.com/pro?cta=edit-document)for more information.



# Tutorial: Cluster Muse with Rstudio

***Transversal axis TIM, UR AIDA***

*Benjamin Heuclin, Statistical Engineer, UR AIDA, CIRAD*

*September 2022*

License: This work is made available under the terms of the Creative Commons Attribution-NonCommercial 4.0 International License.

1. [What is a supercomputer? And parallelization?](#cluster_parallelisation)
   1. [Parallelization](#parallelisation)
   2. [MUSE computing cluster (MESO@LR)](#cluster)
2. [Cluster Connection](#connexion)
3. [File transfer](#transfer)
4. [Storage spaces](#stockage)
5. [Procedure for submitting R jobs](#proc_soumission)
6. [Useful SLURM commands](#commandes)
7. [Rstudio on the cluster](#rstudio)
8. [Resources](#ressources)

* [Annexes](#annexes)
* A.[Examples](#annexes_exemples)
  1. [Example R OpenMP on RF calibration](#ex_R_openMP)
  2. [Example R openMP for loop](#ex_R_forloop)
  3. [Example R array](#ex_R_array)
* B. [Batch script corrupted due to WINDOWS line breaks](#annexes_unix_LF)
* C. [rsync](#annexes_rsunc)

🚨 This document is limited to using R on CPU, with shared memory parallelization (openMP). It is an introduction to quickly get a handle on the job submission process.

For CIRAD agents, you must request access to Bertrand Pitollat [(](mailto:bertrand.pitollat@cirad.fr)bertrand.pitollat@cirad.fr) by sending him an email specifying the unit, your CIRAD telephone number and the signed charter.

You will then get a username (usually the same as your CIRAD account) and a password (usually the same as your CIRAD account).

To use the cluster, you need a unix terminal to connect, submit and manage your jobs and a file transfer software to send your codes, ... from your PC to the cluster and vice versa.

Documentation Muse : <https://meso-lr.umontpellier.fr/documentation-utilisateurs/>

# 1. What is a supercomputer? and parallelization?

[wikipedia definition](https://fr.wikipedia.org/wiki/Superordinateur): A supercomputer is a computer designed to achieve the highest possible performance with the techniques known at the time of its design, particularly with regard to computing speed. For performance reasons, it is almost always a mainframe, with tasks delivered in batch processing. The science of supercomputing is called high-performance computing (HPC).

A supercomputer is generally a grouping of several independent computers called nodes, hence the term cluster.

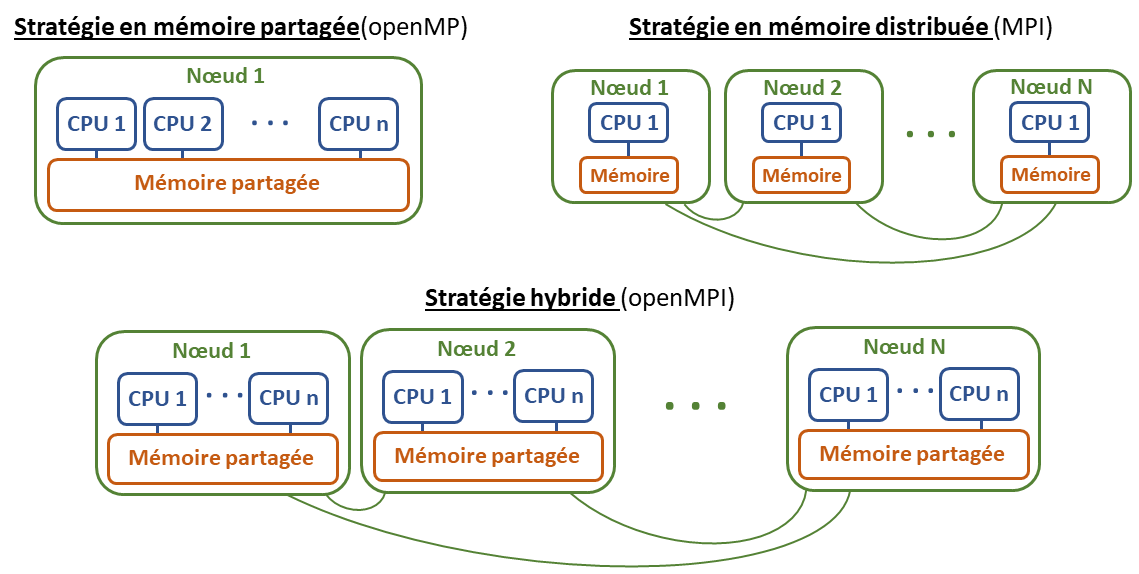
## 1.1 Parallelization

This link explains very well the different types of parallelization:

<https://cwant.github.io/hpc-beyond/21-introduction-to-parallelism/index.html>

A task (or process, or thread) is a logical processing unit

* **Shared memory strategy: this** is the situation where your program runs tasks on several CPUs (1 per task) on the same node, and each CPU can access all the memory used by the program. A widely used library for this type of parallelism is OpenMP (Open Multi-Processing). This type of parallelization can occur during the execution of a specific "for" loop by distributing the loop among the different CPUs. **This type of parallelization is possible on your PC.**
* **Distributed memory strategy:** each task is executed on a CPU (on a node) that has its own private memory, and no other CPU can see this memory (independence). In order to communicate what is in the memory space from one CPU to another, the CPUs "pass messages" to each other. With this design, the code is modularized so that parts of the program can be run on several different machines (nodes), each machine having to work with its own memory space. A popular library for implementing this type of parallelism is called MPI (Message Passing Interface).
* **Hybrid strategy:** memory is distributed between nodes, but on each node the code can use a shared memory strategy. This could be a case where you want to use MPI to pass messages between each node, but on each node you use a shared memory strategy using OpenMP.



**In summary:**

|  | 1 node | n nodes |
| --- | --- | --- |
| 1 CPU | Job in series | MPI |
| n CPUs | OpenMP | hybrid: OpenMPI |

## 1.2 The MUSE computing cluster ([MESO@LR](mailto:MESO@LR))

**The numbers:**

* 308 Dell PowerEdge C6320 compute nodes
  + two Intel Xeon E5-2680 v4 2.4 Ghz processors (broadwell)
  + **28 CPUs per node**, total: 8624 CPUs
  + 128 GB RAM per node
  + 330 Tflops
* 2 nodes large memory 112 cores, 3TB RAM
* 2 GPU visualization nodes, 52 CPU cores (dual 26-core processors), dedicated and configured for post-processing (Poweredge R740 embedding RTX6000)
* 1.3 PB of storage dedicated to computing
  + 1" of fast storage under the chandelier
  + 350 TB of perennial storage
* Intel OmniPath 100 Gb/s Interconnect Network
* No throttle
* Job submission manager: [SLURM (Simple Linux Utility for Resource Management)](https://slurm.schedmd.com/documentation.html)
  + Scheduling of tasks in queues (arbitration)

**Operation:**

* Users belonging to groups run jobs on partitions
* A partition is a set of nodes

**Scores for Ciradians:**

You have to choose the partition on which to launch your jobs. There are several partions for the ciradians:

| Score | Description | Time limit | nb nodes | nb CPUs per node | Default memory \* | Max memory |
| --- | --- | --- | --- | --- | --- | --- |
| agap\_short | For fast jobs | 1 h | 71 | 28 | 4 GB | 128 GB |
| agap\_normal | Default partition | 2 j | 67 | 28 | 4 GB | 128 GB |
| agap\_long | For time-consuming jobs | No limit | 67 | 28 | 4 GB | 128 GB |
| agap\_bigmem | large memory calculations | No limit | 1 | 112 | 28 GB | 3 To |

The RAM per node is limited by default (see column 6) but it can be increased by adding the line :

* --mem=XG (for the memory allocated for the entire job)
* or --mem-per-cpu=XG (for memory allocated for each CPU)

in your batch script (see section [Job submission](#proc_soumission)) with "X" the amount of memory. See column 7 for the maximum amount of memory per node. These 2 parameters are exclusive of each other.

# 2. Connection to the Cluster

It is very simple! The connection to the HPC cluster is done via the SSH protocol. The hostname of the connection machine is muse-login.meso.umontpellier.fr.

Depending on your operating system, you can connect as follows:

**Under linux or Mac :**

Open an ssh connection in a terminal by typing the following command:

ssh "username"@muse-login.meso.umontpellier.fr

Then enter your CDM.

On the Mac, you can also use the Xquartz software.

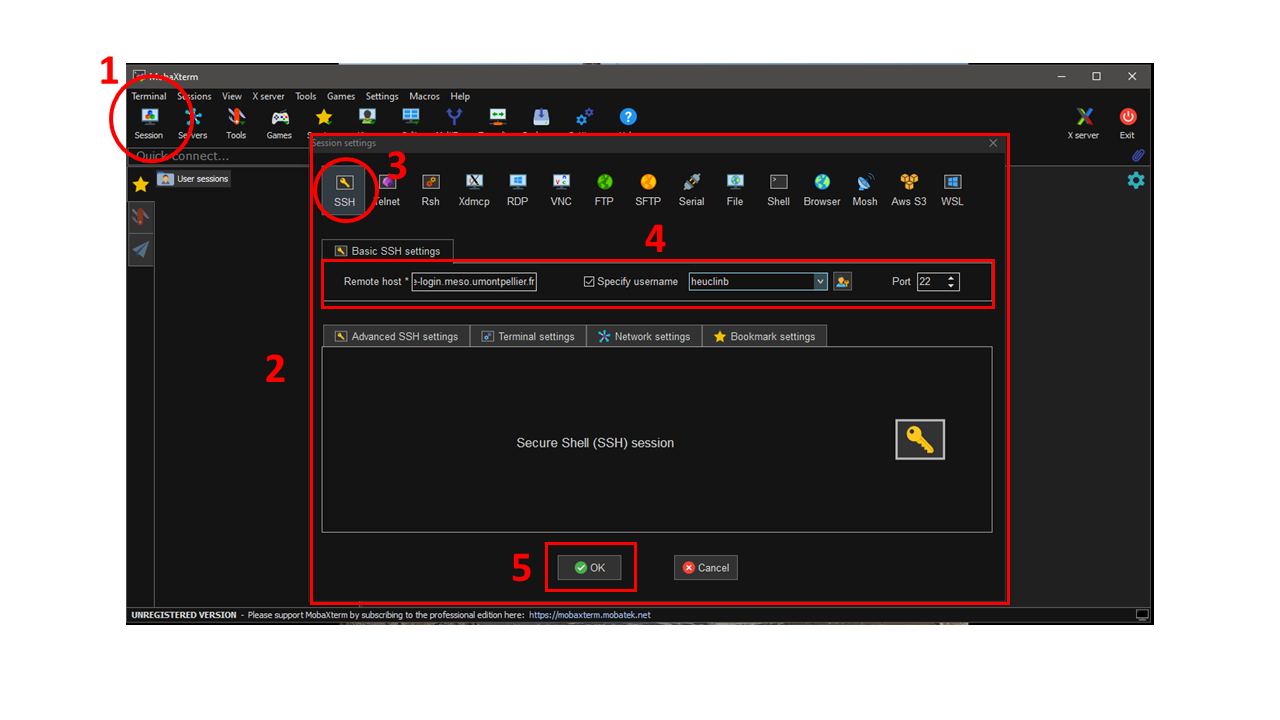
You are now connected to the Muse cluster. The Muse cluster uses the SLURM job manager. It is from here that you will be able to run and manage your jobs with the specific SLURM commands (see below for the main commands).

**Under windows :**

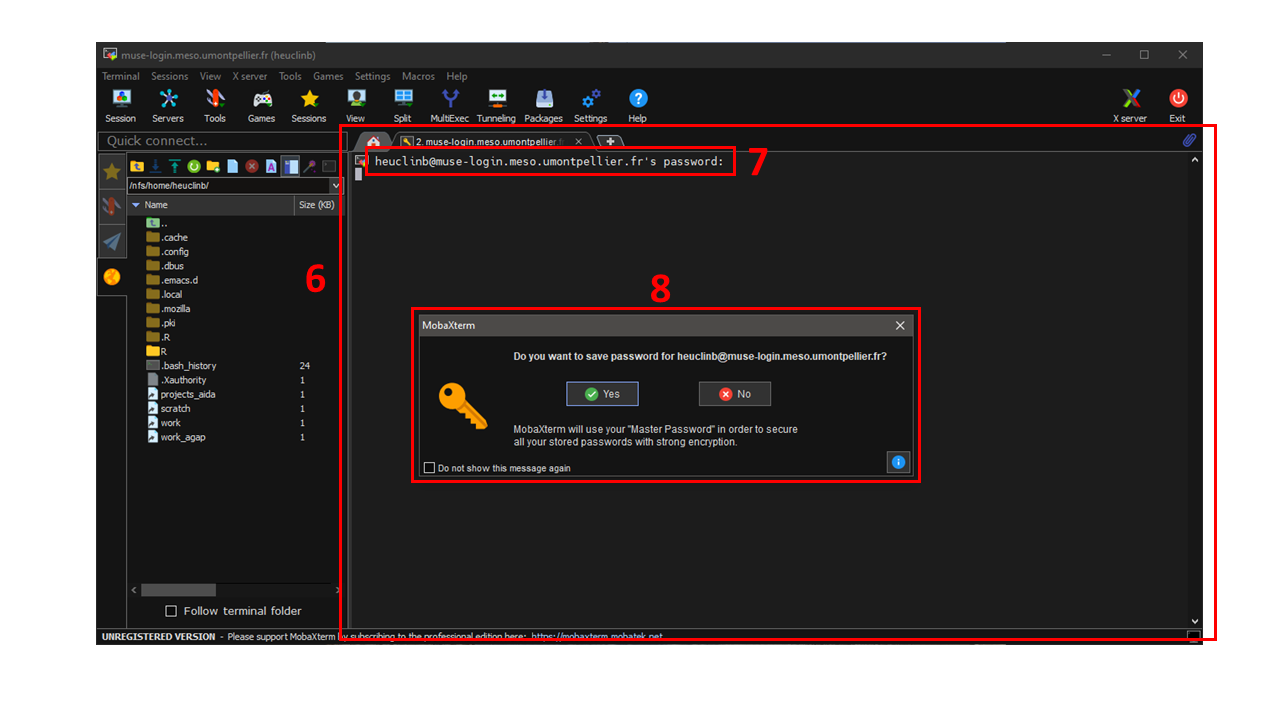
Install the MobXterm software [(](https://mobaxterm.mobatek.net/download-home-edition.html)https://mobaxterm.mobatek.net/download-home-edition.html). The first time you connect, you have to configure it!

Configuration:

1. Click on the Session button (top left)
2. A "*Session settings*" window opens
3. Click on SSH (top left)
4. Fill in the following fields:
   1. *Remote host* : muse-login.meso.umontpellier.fr
   2. Select *Specify username*
   3. Enter your username
   4. *Port* : 22
5. Click on OK



1. A unix terminal opens.
2. You must then enter your password (nothing is displayed when you type the password, it is a security setting) and then validate by pressing "enter".
3. MobaXterm asks you if you want to register the password so that it doesn't ask you again. It's up to you!



🤩 For the next few times, just open MobaXterm and click on your session which you will find in the "*User sessions*" tab on the left.  
 You can also create a shortcut on your desktop by right clicking on it. This will open your session at the same time as the software launches. This is so cool 🤩

You are now connected to the Muse cluster. The Muse cluster uses the SLURM job manager. It is from here that you will be able to run and manage your jobs with the specific SLURM commands (see section [Job submission](#soumission)).

**Some useful Linux commands:**

* ls to display the contents of the current directory
* ls -a to display all files (even hidden) in the current directory
* cd "path" to change directory
* cd .. to go to parent directory
* pwd to display the absolute path of the current directory (from the root)
* ⬆️⬇️ **Up/down arrow** to navigate through the history of used commands

For more info on basic Linux commands: https:[//doc.ubuntu-fr.org/tutoriel/console\_commandes\_de\_base](https://doc.ubuntu-fr.org/tutoriel/console_commandes_de_base)

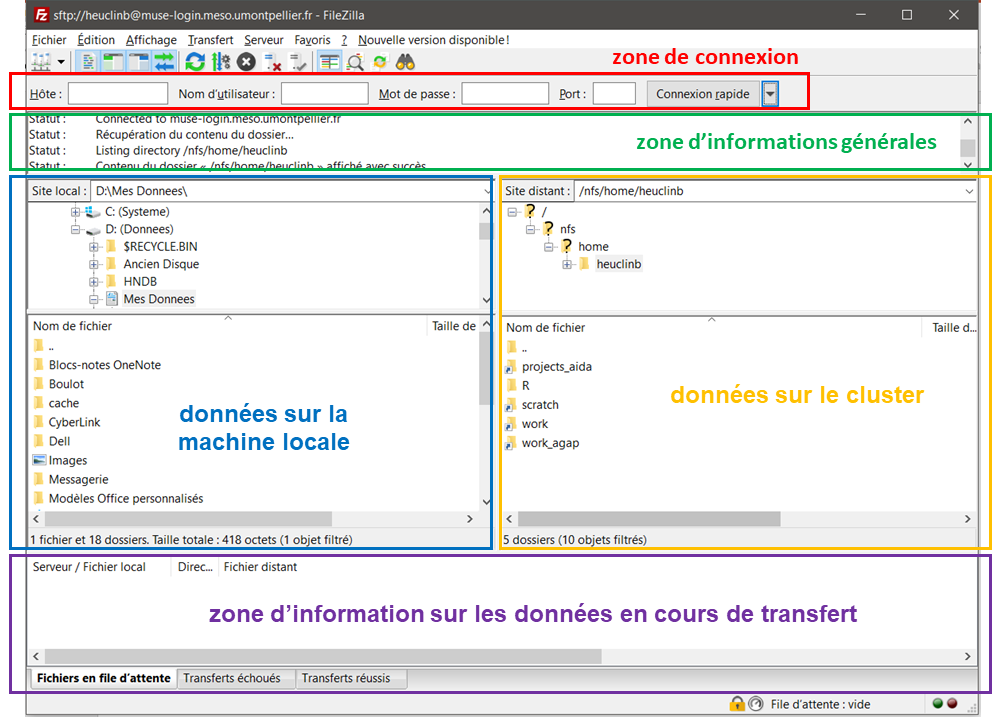
# 3. File transfer

To submit your jobs, you will have to send your scripts on the cluster. You will then have to recover the files generated by your jobs. To do this, we will use the FileZilla software. It is available for Windows, OSX and Linux. To download large files from the cluster to your machine (long with FileZilla) it is possible to use "rsunc" (see [appendix A](#annexes_rsunc))

**Install FileZilla: https:**[//filezilla-project.org/download.php?show\_all=1](https://filezilla-project.org/download.php?show_all=1)

**Note for Linux**: Filezilla is available through your Package Manager apt-get install filezilla

**Presentation of FileZilla :**



To connect, fill in the connection area:

* **Host**: sftp://muse-login.meso.umontpellier.fr
* **User name**: your user name
* **Password**: your password
* **Port** : 22

🤩 After the first login, this information will be saved and you can easily log in by clicking on the little arrow next to "Quick Login."

You can upload a file in either direction by right-clicking on it and then clicking on "Upload" or "Download".

# 4. Storage spaces

There are several storage spaces. Files placed on the "scratch" directory are temporary for your calculations, and are automatically deleted after 60 days. Documents that are to be kept should be placed in your "home" directory.

Email from Bertrand Pitollat on 19/10/2021:

You have several storage spaces on the Muse cluster:

* **home directory**: This is your entry point to the Muse cluster.
  + It is hosted on the NFS array of the Muse cluster.
  + It is not saved or replicated.
  + It is limited to a quota of 50 GB (excluding other storage spaces).
  + It is read and write accessible from login nodes and read-only from compute nodes.
* **home directory on the replicated bay: replicated link**
  + It is hosted on the NetApp array of the Muse cluster.
  + This directory is personal.
  + It is intended for long-term storage of personal data.
  + It is accessible via the replicated link in your home directory (for example, /home/pitollatb/replicated => /storage/replicated/cirad\_users/pitollatb).
  + It is backed up and replicated on a backup bay.
  + It is limited to a quota of 500GB.
  + It is readable and writable from the login and calculation nodes.
* **projects / units / teams space on the replicated bay: projects link**
  + This space is hosted on the NetApp array of the Muse cluster.
  + It is intended for long-term storage of project / unit / team data.
  + The space is accessible via the projects link in your home directory (for example, /home/pitollatb/projects => /storage/replicated/cirad/projects).
  + It is backed up and replicated on a backup bay.
  + It is partitioned by project, unit or team directory with an initial quota of 5TB for each directory.
  + Each project directory can be associated with a group of users to be defined by the collective.
  + The space is readable and writable from the login and calculation nodes.
* **work space : link work\_agap**
  + It is hosted on the NFS array of the Muse cluster.
  + This storage space previously dedicated to the storage of project data should no longer be used.
  + The data in this area must be transferred to the new project / unit / team area.
  + It is not saved or replicated.
  + It is accessible via the work\_agap link in your home directory (for example, /home/pitollatb/work\_agap => /nfs/work/agap).
  + It is read and write accessible from login nodes and read-only from compute nodes.
* **scratch personal space: scratch link**
  + It is hosted on the Lustre bay of the Muse cluster. This space is personal.
  + It is fast and powerful and should be used to host temporary computation data.
  + At the end of the calculation, the data must be deleted or moved.
  + It is accessible via the scratch link in your home directory (for example, /home/pitollatb/scratch => /lustre/pitollatb).
  + It is not saved or replicated.
  + It is limited in time: data older than 2 months will soon be automatically deleted.
  + It is readable and writable from the login and calculation nodes.
* **scratch databases : /lustre/agap**
  + This community space stores the databases.
  + It is hosted on the Lustre bay of the Muse cluster to optimize computation.
  + It is not saved or replicated.
  + It should not be used to store personal data.
  + It is accessible at the location /lustre/agap.
  + The banks maintained by biomaj are accessible at /lustre/agap/BANK/biomaj.
* **web space: In** addition, there is a dedicated space to host the data displayed/distributed by our different web services (genome hubs, ...). Contact us if necessary.

**Important:** I remind you that for performance reasons of the Muse cluster, all writes from jobs must be redirected to your scratch directory and that you must ban all intensive reads from NFS and NetApp spaces.

# 5. Job submission procedure R

In this section I explain how to submit jobs in parallel in R.

To illustrate the submission of a job in parallel, we will use the Example\_R. In this bogus example, I repeat the 2\*k operation for k=1 to 50. I want to parallelize these operations on 10 cores to go 10 times faster. I save each result in a ".Rdata" in a "results" file.

To submit a job, you must choose between :

* A real-time execution mode with the srun command directly in the terminal (not detailed in this tutorial)
* A deferred execution mode by defining the job in a ***batch*** script (.sh) and launching it with the sbatch command in the terminal

⚠️ **I explain here only the delayed procedure with the sbatch command**

This procedure consists in defining the execution parameters in a *batch* file (.sh)

This file can be edited with [**Notepad++**](https://notepad-plus-plus.org/downloads/) or **Rstudio**. To create it :

* with **Notepad++** : *File > New* then save with the *.sh* extension
* with **Rstudio** : *File > New File > Shell Script*

👉 But the easiest way is to take a ***batch*** script (.sh) that we have on hand from another project.

**💥🔥 Windows alert** 🚨🧯 **Beware of carriage returns!!!**  
By default in Windows carriage returns are DOS type and they are not compatible with Linux or UNIX type Max (Posix LF) and it crashes!  
 You need either :

* use **Notepad++** by doing :  
  *Edit > Convert line breaks > Convert to UNIX format (LF)* ⚠️ to do for each new file
* use **Rstudio** by setting the Unix-like line ending option:  
  *Tools > Global options > Code > Saving > Serialization > Line ending conversion > Posix (LF)*   
  To be done only once ❤️💪 He knows how to do everything this Rstudio! 💪❤️

**Let's take the *batch* file of the Example\_R\_openMP\_RF** to see its construction

#!/bin/bash  
#SBATCH --partition=agap\_short # the partition  
#SBATCH --job-name openMP\_RF # job name  
#SBATCH --nodes=1 # nodes (MPI processes, openMP -> 1)  
#SBATCH --ntasks=1 # NB tasks (MPI processes, openMP -> 1)  
#SBATCH --ntasks-per-node=1 # NB tasks per node (MPI processes, openMP -> 1)  
#SBATCH --cpus-per-task=10 # NB CPUs per task  
#SBATCH --mem-per-cpu=100M # Memory per CPU  
#SBATCH --time=0-00:10:00 # Time limit (10 min)  
  
module purge  
module load cv-standard  
module load R/3.6.1  
  
# OpenMP runtime settings  
export OMP\_NUM\_THREADS=$SLURM\_CPUS\_ON\_NODE  
  
cd $SLURM\_SUBMIT\_DIR # To get into the directory where the .sh is running  
  
mkdir ./Rout # Create the Rout folder for R console output  
mkdir ./results # Create the "results" folder for saving my results  
R CMD BATCH ./script\_RF.R ./Rout/script\_RF.Rout

This file is divided into 3 parts:

* 1st part : each line starting with #SBATCH describes a **SLURM** parameter Many options exist, refer to the cluster documentation for more info [(](https://meso-lr.umontpellier.fr/documentation-utilisateurs/)https://meso-lr.umontpellier.fr/documentation-utilisateurs/).
* 2nd part: You must then load the **modules** (software, compilers) with the module load command. [Modules are based on a system of dependencies and conflicts fixed by the person who installed or compiled the software or the library in question]: # We start by unloading all the modules that can be loaded. To use the R software, depending on the version, you must load the cv-standard module and then R/version or R (load the latest version 4.2.2). If your code uses another language (c++, python, ...), you will have to load the modules accordingly. To see the list of available modules : module avail. More info on modules in the cluster documentation [(](https://meso-lr.umontpellier.fr/documentation-utilisateurs/)https://meso-lr.umontpellier.fr/documentation-utilisateurs/) section "Muse Cluster software environment" and the TP : https://meso-lr.umontpellier.fr/wp-content/uploads/2020/03/1-TP-Environment\_module3.pdf. I advise you to specify the version of R you want to work with to avoid surprises if a new version is installed. The available R versions are :
  + locally (installed by [muse@lr](mailto:muse@lr)) :
    - R/3.6.1
    - R/3.6.1-tcltk
    - R/3.6.3
    - R/4.0.2
    - R/4.2.2
  + in cv-standard (most standard compilers and libraries) :
    - R/3.3.1
    - R/3.4.3

**You can also install your own software.**

* 3rd part: Finally the line to execute the R script: R CMD BATCH followed by the path to the script, followed by the path to the **Rout** to write the output (which is displayed in the Rstudio console in normal circumstances). **Remember to create the Rout file in your project**.

To launch your code, you have to execute in the terminal (after being placed in the right directory) the associated batch file with the command :

sbatch job\_submission.sh

**Warning**: if your code requires the loading of packages, you must install them first. To do this, in the terminal, load the modules then launch R :

module load cv-standard R/3.6.1  
R

Then install the packages (you will have to choose a mirror):

install.packages("doParallel")

Finally quit R with the q() command.

# 6. Useful SLURM commands

To see the status of all jobs (of all users)

skeleton

To see the status of your jobs

squeue -u $USER -o '%.18i %.9P %.20j %u %.8T %.10M %.9l %.6D %R'

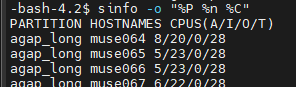
To kill a job

scancel <JOB\_ID>

To see the number of CPUs available per node (very useful to choose the number of cores to pass in front of the whole queue)

sinfo -o "%P %n %C"

* The 1st column (%P) gives the partition
* The 2nd column (%n) gives the identifier of the node
* The 3rd column (%C) gives the number of CPUs per state in the format "allocated/free/other/total"



View the amount of memory consumed by the job after its execution:

sacct -o JobID,Node,AveRSS,MaxRSS,MaxRSSTask,MaxRSSNode,TRESUsageInTot%250 -j <JOB\_ID>

More info on ordering here: https:[//slurm.schedmd.com/man\_index.html](https://slurm.schedmd.com/man_index.html)

# 7. Rstudio on the cluster

<http://193.52.26.138/rstudio/auth-sign-in>

Rstudio is isolated on a dedicated node (96 CPUs and 3TB of memory) and is shared with all Rstudio users.

**It is intended for the development of scripts but it is necessary to avoid launching heavy calculations directly on it. Once developed, it is better to submit them via batch script.**

# 8. Resources

**Mailing list of mutual aid:** [meso-help@umontpellier.fr](mailto:meso-help@umontpellier.fr)

**Website on MUSE** [**MESO@LR**](mailto:MESO@LR) **cluster: https:**[//meso-lr.umontpellier.fr/documentation-utilisateurs/](https://meso-lr.umontpellier.fr/documentation-utilisateurs/)

**Presentation of the MUSE** [**MESO@LR**](mailto:MESO@LR) **cluster: https:**[//meso-lr.umontpellier.fr/wp-content/uploads/2019/11/1-Presentation\_cluster\_Muse.pdf](https://meso-lr.umontpellier.fr/wp-content/uploads/2019/11/1-Presentation_cluster_Muse.pdf)

**TP-Environment Module: https:**[//meso-lr.umontpellier.fr/wp-content/uploads/2020/03/1-TP-Environment\_module3.pdf](https://meso-lr.umontpellier.fr/wp-content/uploads/2020/03/1-TP-Environment_module3.pdf)

**TP-SLURM: https:**[//meso-lr.umontpellier.fr/wp-content/uploads/2020/04/1-TP-SLURM3.pdf](https://meso-lr.umontpellier.fr/wp-content/uploads/2020/04/1-TP-SLURM3.pdf)

**Basic Linux commands: https:**[//doc.ubuntu-fr.org/tutoriel/console\_commandes\_de\_base](https://doc.ubuntu-fr.org/tutoriel/console_commandes_de_base)

**Quick Intro to Parallel Computing in R (and the refs inside): https:**[//nceas.github.io/oss-lessons/parallel-computing-in-r/parallel-computing-in-r.html](https://nceas.github.io/oss-lessons/parallel-computing-in-r/parallel-computing-in-r.html)

# Annexes

## A. The examples

### A.1 Example R OpenMP on RF calibration

⚠️ ATTENTION For this example you need to install the packages "doParallel", "caret" and "randomForest" in your R/3.6.1.

In this example, I want to calibrate a random forest model on the "iris" data (basic in R).

**Data description (?iris)**: This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

I cumulated the dataset 3 times to get to 450 observations otherwise it is too fast.

The objective here is to find the variety according to the measures made on the sepals and petals. We are thus in a classification and we are going to realize that with a Random Forest.

To do this we need to calibrate the "mtry" parameter (Number of variables randomly sampled as candidates at each split). We then perform a CV using the caret package and its train function. This process implies to launch several random forests with different parameters. This parallelizes very well (10 RF in parallel on 10 cores in the example (1 per core)). The caret package handles the parallelization for you with the allowParallel = TRUE option in the trainControl function (OpenMP mode). You just need to declare a number of CPUs available for parallelization at the beginning of the R script. This can be done using the doParallel package:

doParallel::registerDoParallel(cores=10)

the R script looks like :

library(doParallel, caret, randomForest)  
  
# Parallel settings -------------------------------------------------------  
doParallel::registerDoParallel(cores=10)  
  
# the DATA ----------------------------------------------------------------  
iris2 <- rbind(iris, iris, iris)  
  
# Split data in train/test set --------------------------------------------  
trainingIndex <- createDataPartition(iris2$Species, p=0.8, list=FALSE)  
data\_train <- iris2[trainingIndex, ]  
data\_test <- iris2[-trainingIndex, ]  
  
# RF calibration ----------------------------------------------------------  
RF\_model\_train <- train(Species~., data = data\_train,  
 method = "rf",  
 trControl = trainControl(method = "LOOCV", allowParallel = TRUE),  
)   
  
# RF fit ------------------------------------------------------------------  
RFFGrid <- expand.grid(mtry=RF\_model\_train$bestTune$mtry)   
fit\_RF <- train(Species~.,data = data\_train,  
 method = "rf",  
 trControl = trainControl(method = "none"),  
 tuneGrid = RFFGrid  
)  
  
save(fit\_RF, file="results/fit\_RF\_iris.Rdata")  
  
  
# RF Predictive performances -----------------------------------------------  
table(data\_test$Species, predict(fit\_RF, new=data\_test))

This R script can run on your PC (be careful to adapt the number of CPUs). To detect the number of CPU on your PC, you can use the function parallel::detectCores(). Do not take more than max-1 on your PCs!

**Tip:** You can retrieve in R the number of CPUs that you have declared in the batch (.sh) (environment variable SLURM\_JOB\_CPUS\_PER\_NODE) with the command R :

nb\_CPUs <- as.integer(Sys.getenv("SLURM\_JOB\_CPUS\_PER\_NODE"))   
doParallel::registerDoParallel(cores=nb\_CPUs)

This avoids mistakes ;)

### A.2 Example R openMP for loop

Let's say I need to execute a function or code on different input parameters or data (or both). I can then do this with a "for" loop.

For example, I want to apply the function my\_fct = n \* p \* k on different input parameters:

* n = 1 or 2
* p = 1, 2 or 3
* k = 1 or 2

This makes 2\*3\*2 = 12 combinations and therefore 12 executions of my function. I can do this very well with the R code:

# Definition of my function  
my\_fct <- function(n, p, k) return(n\*p\*k)  
  
# Definition of a grid of parameters I want to expand  
pars <- expand.grid(n = 1:2, p = 1:3, k = 1:10)  
  
# execute my function on the different parameter combinations  
for(i in 1:nrow(pars)){  
 result <- my\_fct(n=pars$n[i], p=pars$p[i], k=pars$k[i])  
 save(result, file = paste0("results/my\_result\_n=", pars$n[i], "p=", pars$p[i], "k=", pars$k[i], ".Rdata" )  
}

This can be very long depending on the application, but **a "for" loop parallels very well in openMP**.

Each task (iteration of the loop) needs to access what was loaded in the environment before the loop, we need to be in shared memory  **openMP**).

This is easily done with the foreach function of the package of the same name. However, it is necessary to declare a certain number of CPUs available with the registerDoParallel function of the doParallel package.

**The R script becomes :**

library(doParallel)  
  
# Set the number of cores  
doParallel::registerDoParallel(cores = 10)  
  
# Definition of my function  
my\_fct <- function(n, p, k) return(n\*p\*k)  
  
# Define a grid of parameters that I want to expand  
pars <- expand.grid(n = 1:2, p = 1:3, k = 1:10)  
  
# Parallel for loop  
foreach::foreach(i = 1:nrow(pars), .verbose = FALSE) %dopar% {  
 result <- my\_fct(n=pars$n[i], p=pars$p[i], k=pars$k[i])  
 save(result, file = paste0("results/my\_result\_n=", pars$n[i], "p=", pars$p[i], "k=", pars$k[i], ".Rdata") )  
  
 return() # I return nothing because I save each result in ".Rdata" object in folder "results"  
}

This R script can run on your PC (be careful to adapt the number of CPUs). To detect the number of CPU on your PC, you can use the function parallel::detectCores(). Do not take more than max-1 on your PCs!

**The batch file for the job submission:**

#!/bin/bash  
#SBATCH --partition=agap\_short # the partition  
#SBATCH --job-name ex1 # job name  
#SBATCH --nodes=1 # nodes (MPI processes, openMP -> 1)  
#SBATCH --ntasks=1 # NB tasks (MPI processes, openMP -> 1)  
#SBATCH --ntasks-per-node=1 # NB tasks per node (MPI processes, openMP -> 1)  
#SBATCH --cpus-per-task=10 # NB CPUs per task  
#SBATCH --mem-per-cpu=100M # Memory per CPU  
#SBATCH --time=00:10:00 # Time limit  
#  
#SBATCH --mail-type=begin # send email when job begins  
#SBATCH --mail-type=end # send email when job ends  
#SBATCH --mail-user=benjamin.heuclin@cirad.fr  
  
module purge  
module load cv-standard  
module load R/3.6.1  
  
# OpenMP runtime settings  
export OMP\_NUM\_THREADS=$SLURM\_CPUS\_ON\_NODE  
  
cd $SLURM\_SUBMIT\_DIR  
  
mkdir ./Rout   
mkdir ./results  
R CMD BATCH ./main\_script.R ./Rout/main\_script.Rout  
  
# Rscript ./main\_script.R

### A.3 Example R array

The "array" sub-menu type is suitable for executing a code (a function) several times with different input parameters or on different data (or both).

Let's go back to the [R example of the for loop](#ex_R_forloop) just above. In this example, I want to execute the function my\_fct=n\*p\*k on different input parameters:

* n = 1 or 2
* p = 1, 2 or 3
* k = 1 or 2

**The 12 executions are independent** and therefore we can optimize the launch of this job using an **array**. The principle is that we ask the cluster a certain number of CPUs and the cluster will choose them potentially in different nodes.  
 So we are on a form of hybrid OpenMPI parallelization!

When there are a lot of jobs waiting on the cluster, it allows your job to go faster because it's easier to take *n* CPUs here and there than *n* CPUs on the same node. Resource reservation is optimized!

To do this, we specify the --array option in the batch (.sh) :

#!/bin/bash  
#SBATCH --partition=agap\_short # the partition  
#SBATCH --job-name array # job name  
#SBATCH --array=1-12 # OPTION ARRAY   
#SBATCH -o array-%a.out  
#SBATCH --mem-per-cpu=100M # memory per CPU  
#SBATCH --time=00:30:00 # Time limit  
  
module purge  
module load cv-standard  
module load R/3.6.1  
  
cd $SLURM\_SUBMIT\_DIR  
  
mkdir ./results  
Rscript ./main\_script.R $SLURM\_ARRAY\_TASK\_ID

⚠️ **Warning**: We do not specify the number of nodes, cores or tasks we want. It is SLURM that will distribute according to the available resources.

On the last line of execution of the R script, we add the environment variable $SLURM\_ARRAY\_TASK\_ID, this allows the R script to retrieve the task number.   
And finally in the R script we have to use the command as.numeric(commandArgs(trailingOnly=TRUE)[1]) to get the index (*i)*. I can thus launch the function on the *i* th line of my parameter grid.

# get the task index ($SLURM\_ARRAY\_TASK\_ID)  
i = as.numeric(commandArgs(trailingOnly=TRUE)[1])  
  
# Define a grid of parameters that I want to expand  
pars <- expand.grid(n = 1:2, p = 1:3, k = 1:2)  
  
# Definition of my function  
my\_fct <- function(n, p, k) return(n\*p\*k)  
  
# Execute the function on row i of the parameter grid  
result <- my\_fct(n=pars$n[i], p=pars$p[i], k=pars$k[i])  
print(paste0("The result of my function is: ", result))

The scanning of the R code is totally rethought, no need to declare a number of CPUs as in the previous example. Here the script is designed for one execution and it must be independent of other executions (we load everything the task needs: the parameter grid, the function). This script can't be executed on your PC as is, unlike the previous R script!

To delete tasks in a job array :

# Cancel array ID 1 to 3 from job array 20  
$ scancel 20\_[1-3]  
  
# Cancel array ID 4 and 5 from job array 20  
$ scancel 20\_4 20\_5  
  
# Cancel all elements from job array 20  
$ scancel 20

More info here: https:[//slurm.schedmd.com/job\_array.html](https://slurm.schedmd.com/job_array.html)

## B. Batch script corrupted due to WINDOWS line breaks

**If you have a corrupted file due to line breaks in WINDOWS format**

-bash-4.2$ sbatch job\_submission.sh  
sbatch: error: Batch script contains DOS line breaks (\r\n)  
sbatch: error: instead of expected UNIX line breaks (\n).

You can either:

* in **Notepad++** do: *Edit > Convert line breaks > Convert to UNIX format (LF)* and save
* in **Rstudio** (with the right option for "unix LF" line breaks as described above): modify your .sh slightly (with a line break for example) and save it. Rstudio will automatically convert the line breaks

## C. rsync IN CONSTRUCTION

source : <https://meso-lr.umontpellier.fr/documentation-utilisateurs/>

#! /usr/bin/env bash  
###################################################################  
# rsync.sh : Written by Jérémy Verrier #  
# Script allowing the secure copy of files or folders #  
###################################################################  
  
# Enter your username  
USER=  
# Enter the full path of the directory or file to copy (/home/verrier/work/results.txt)  
FOLDER\_CLUSTER=  
# Enter the full path of the destination directory or file  
DOSSIER\_PERSO=  
  
while [ 1 ]  
do  
 rsync -avz --progress --partial "${USER}"@muse-login.meso.umontpellier.fr:"${DOSSIER\_CLUSTER}" "${DOSSIER\_PERSO}"  
 if [ "$?" = "0" ] ; then  
 echo "Rsync OK"  
 exit  
 else  
 echo "Rsync error, retry in 1 minute..."  
 sleep 60  
 fi  
done

To save in .sh format or to download with the link below.

[This script](https://hpc-lr.umontpellier.fr/wp-content/uploads/2017/05/rsync.txt) allows you to copy data from the Muse cluster to your machine. You have to modify the USER, DOSSIER\_CLUSTER and DOSSIER\_PERSO fields and then launch it with the "bash rsync" command. It is highly recommended to use this script when downloading large files.