


R en el CCAD

Lic. Marcos Mazzini - CCAD / UNC
CPA CONICET

Lic. Juan Cruz Rodriguez - FAMAF / UNC
CIDIE - CONICET



Motivación

En una computadora de escritorio hay cálculos que son casi instantáneos pero hay casos en los que es necesario más de un equipo porque:

- tardan mucho (o son muchos)
- ocupan mucha memoria
- ocupan mucho espacio en disco

Usar los varios procesadores de una sola computadora ayuda.

Usar muchas computadoras organizadas y administradas es una solución.

Supercomputadoras - Historia

- Monoprocesador ultra-rápido (1960s)
- Procesador vectorial: mono (1970s) / multi 4-16 (1980s)
- Múltiples procesadores estándar con memoria compartida o distribuida (1990s)
- Múltiples procesadores (cores) en un sólo chip (2000s)
- GPU: cientos de cores especializados (2010s)

Importancia de la programación paralela

Cada 10 años tenemos un cluster condensado en nuestra computadora.

La programación paralela se va tornando esencial aún para las computadoras de escritorio.

Cómputo intensivo en R

Uso interactivo de R

R es un lenguaje **interpretado**. Estos lenguajes trabajan en un bucle REPL cuando se usan interactivamente

- read
- eval
- print
- loop

Esto puede no ser lo más apropiado en ciertos casos.

Uso no interactivo de R

Cuando ya nos familiarizamos con el dataset y ejecutamos repetidamente las mismas líneas o vamos a correr una simulación larga, la mejor alternativa es salvar la secuencia de comandos en un archivo (script) y que R lo ejecute de forma no interactiva.

R cuenta con diversos mecanismos para **ejecutar scripts**, uno de los más extendidos es **Rscript** (ya viene con R), otro es **littler**.

Rscript

Rscript es un **comando**, que a diferencia de RStudio no necesita de una interfaz gráfica para ejecutarlo, sólo un intérprete modo texto: consola / terminal / shell.

Rscript

Si guardamos todas las instrucciones en un archivo llamado **script.R** podremos ejecutarlo de modo no-interactivo abriendo una consola y ejecutando el siguiente comando:

```
C:\Users\Usuario> Rscript.exe script.R
```

```
[usuario@localhost ~]$ Rscript script.R
```

Rscript

También podemos evaluar **una expresión** para un cálculo rápido:

```
Rscript -e 'sqrt(81) + abs(-10) - sin(pi/2) '  
Rscript -e "library(rmarkdown); render('document.Rmd')"
```

Rscript

En LINUX también sirve para crear scripts que sean tratados como comandos / ejecutables. Para esto creamos un archivo de texto encabezado con:

```
#!/usr/bin/env Rscript  
<script R>
```

```
$ chmod +x calcular.R
```

Sería el equivalente a Abrir con...

Parametrizar nuestro script

Como vamos a ejecutar nuestros scripts de forma no interactiva necesitamos **parametrizarlos** para poder reutilizarlos.

Una forma de hacerlo es tomar los parámetros desde la línea de comandos cuando invocamos a nuestro script.

Parametrizar nuestro script

Si incluimos las siguientes líneas en nuestro script

```
args = commandArgs(TRUE)
input1 = args[1]
input2 = args[2]
input3 = args[3]
...
```

y las guardamos en un archivo llamado “run.R” podemos ejecutar:

```
$ Rscript run.R arg1 arg2 arg3
```

Parametrizar nuestro script

Es importante **verificar los parámetros**, en particular si se trata de **paths** (rutas a archivos o directorios). Es una fuente común de errores.

Verificar un parámetro path

```
...
input_path = args[3]

(!file.exists(input_path)) {
    cat('No se puede abrir el archivo', input_path, '\n')
    stop()
}
```

Organizar Inputs y resultados

En un script no-interactivo nuestros resultados necesariamente deben ir a un archivo. Cuando tenemos gran cantidad de simulaciones corriendo al mismo tiempo, con distintos archivos para entradas y salidas, la desorganización nos puede dar problemas.

En un sistema LINUX cada usuario escribe dentro de

/home/usuario (\$HOME , atajo ~)

En nuestro cluster hay **500GB** de quota disponible.

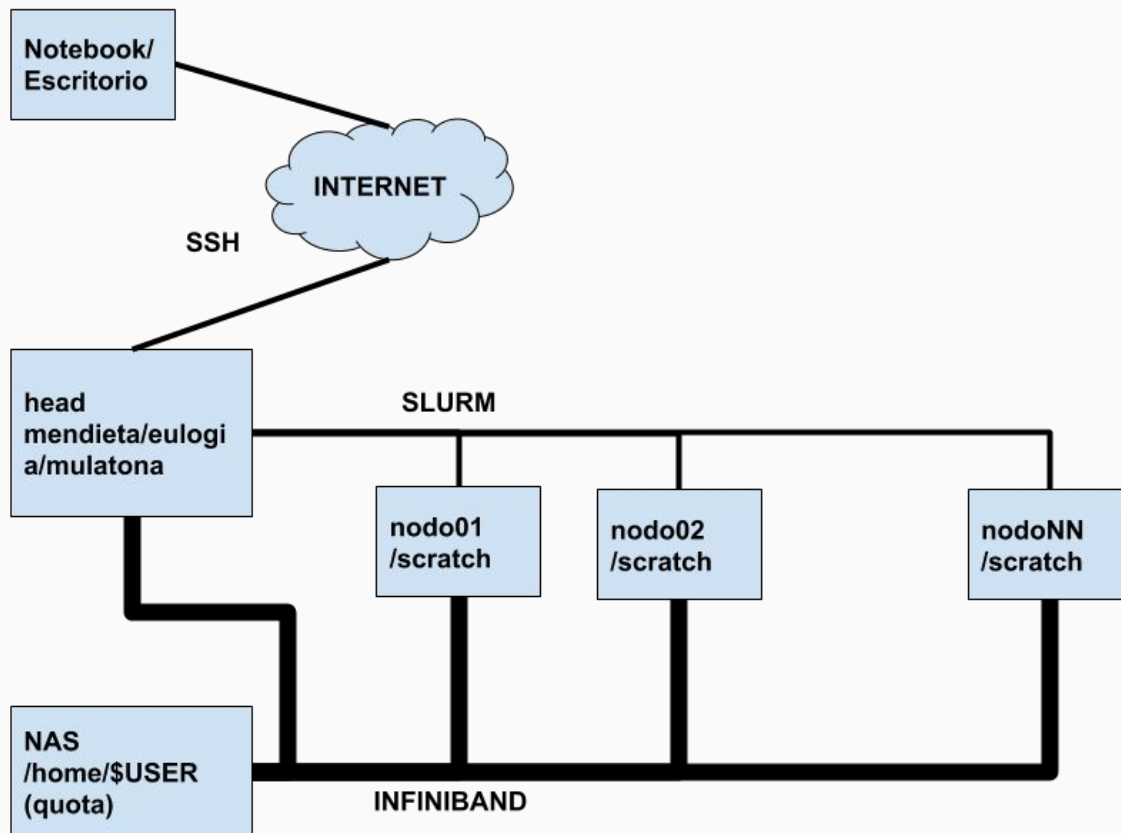
Organizar Inputs / resultados

```
proyecto1\  
  
  casoA\  
  
    input\  
  
    resultado\  
  
  casoB\  
  
    input\  
  
    resultado\  

```

Comandos: cd, mkdir, mv, cp, rmdir, rm

Los Clusters



Mecanismo de uso del Cluster

Para utilizar los **nodos** debemos establecer una **conexión remota** y enviar los trabajos de forma **no interactiva** al sistema de colas **SLURM**.

Para conectarnos usaremos una **terminal** modo texto. Mantener múltiples conexiones gráficas al servidor es costoso y poco fluido sobre internet.

Para establecer la conexión nuestro sistema deberá contar con un **cliente SSH**.

SSH

Secure Shell

Es un **protocolo de red** criptográfico que permite realizar tareas privadas sobre una red insegura.

Es la forma más extendida de conectarse a una terminal remota.

Al conectarnos por SSH se abre un shell y todos los comandos que introducimos se transmiten **hacia el servidor** por un túnel encriptado.

SSH

El **cliente SSH** para línea de comandos viene instalado por defecto en la amplia mayoría de las distribuciones LINUX y MacOS.

Para windows recomendamos **mobaXterm**. Ofrece una terminal y un cliente SSH entre otras cosas.

Se puede descargar para instalar o portable de

<https://mobaxterm.mobatek.net/download-home-edition.html>

SSH

Para conectarnos debemos tener una cuenta de usuario **en el servidor remoto** (que podría ser distinta a la cuenta local).

Para obtener su cuenta de usuario en el cluster deben completar el siguiente formulario:

<https://tinyurl.com/rccad>

Llave público-privada

En el CCAD no autenticamos los usuarios con contraseña sino que utilizamos el mecanismo de **llave público-privada**.

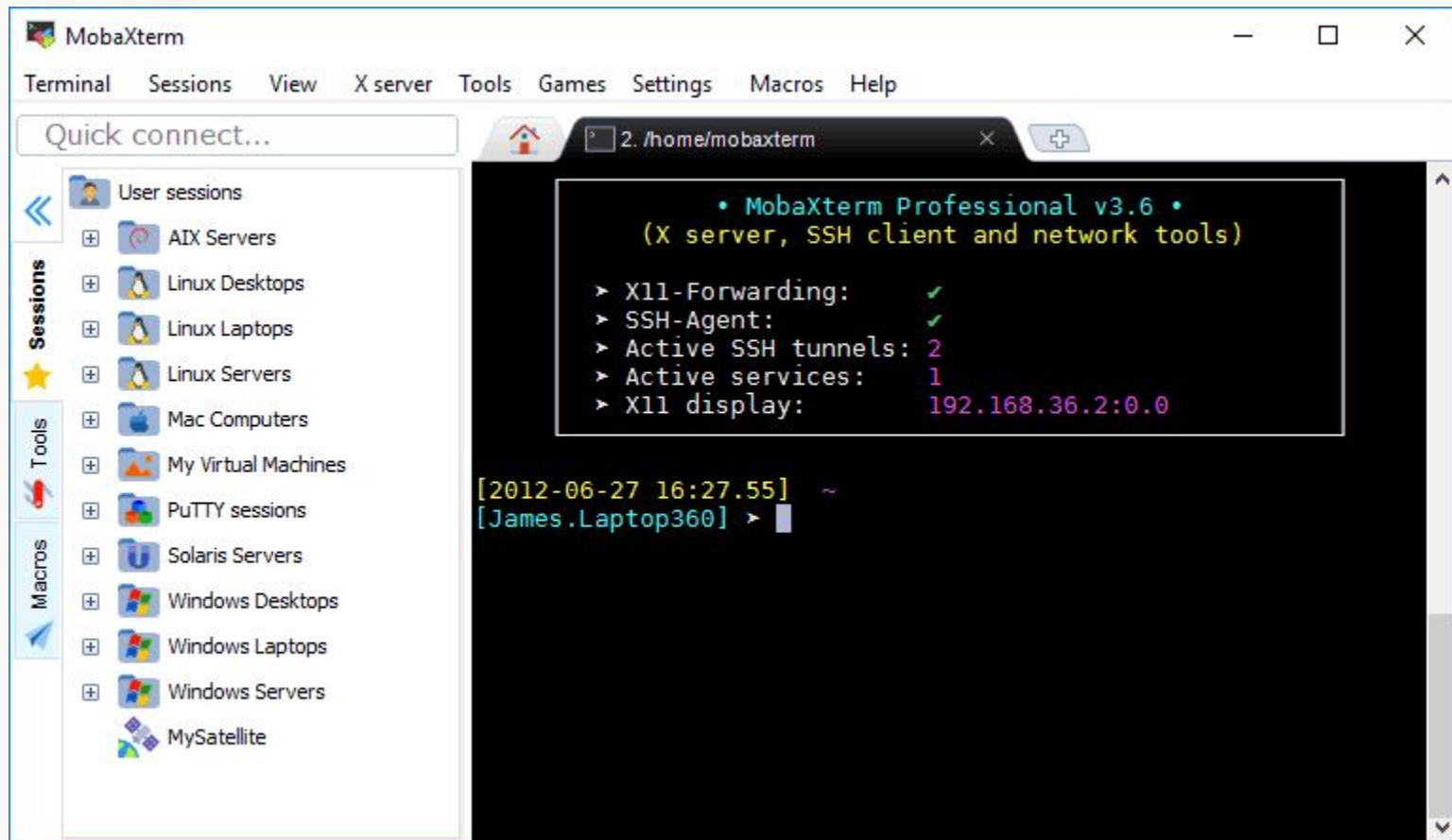
En el formulario se brindan detalles sobre cómo generar la llave.

SSH

Una vez que confirmaron nuestra cuenta podemos conectarnos abriendo la terminal y ejecutamos el comando:

```
[usuario@localhost ~]$ ssh usuario@mendieta.ccad.unc.edu.ar
```

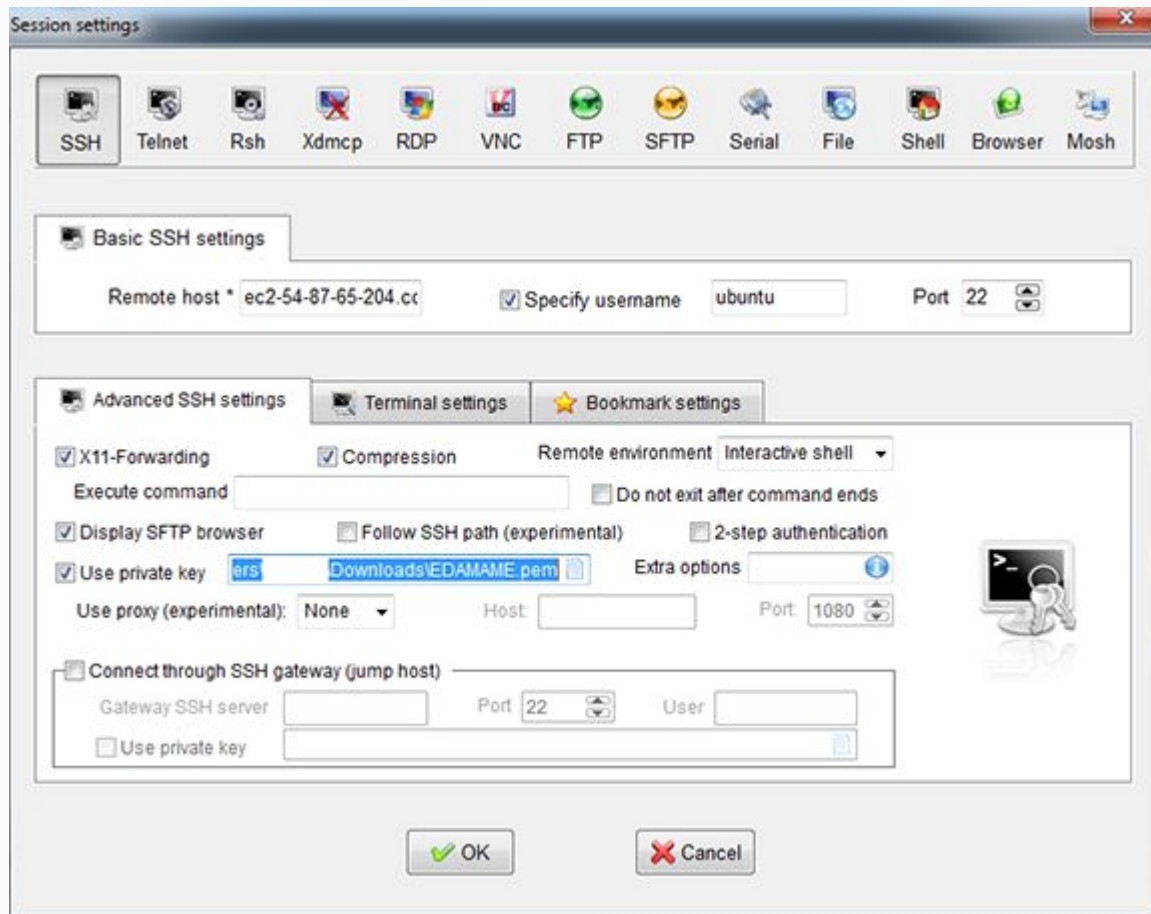
Con mobaXterm abrimos el programa y configuramos la conexión remota:



```

b00m@acer:~/PCsuggest$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/b00m/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/b00m/.ssh/id_rsa.
Your public key has been saved in /home/b00m/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:
The key's randomart image is:
+---[RSA 2048]----+
|  ..  o. .      |
| .o.+  o .     |
|  o=   o       |
|  o.    o      |
| . . . + S o   |
| * .   + = * +  |
| 0..o. o = = o |
| ==. .  B = .   |
| +=o .Eo *..   |
+-----[SHA256]-----+

```



Remote host: mendieta.ccad.unc.edu.ar | username: usuario remoto | Use private key: tildar y especificar archivo de llave

SSH

Con la conexión establecida ya podemos introducir comandos que se ejecutarán en el **nodo cabecera** del cluster.

El siguiente paso es cargar el entorno para R.

Módulos

La instalación de R en el cluster sólo cuenta con los paquetes de R básicos.

En el nodo cabecera debemos realizar la instalación de paquetes extra **dentro del intérprete de R** dado que no está disponible un entorno de desarrollo gráfico como Rstudio.

Para cargar el intérprete de R usaremos el mecanismo de **environment modules**.

Módulos / Instalar paquetes R

```
[usuario@mendieta ~]$ module load R  
[usuario@mendieta ~]$ R
```

Los comandos para instalar los paquetes son realmente sencillos. Podemos instalar uno solo con el comando

```
> install.packages("xbreed")
```

o bien varios en un mismo comando guardandolos en un arreglo de caracteres:

```
> install.packages(c("xbreed", "fpc", "cluster", "clValid"))
```

SSH - Copiando archivos

Una vez que instalamos todas las dependencias ya podemos transferir nuestros scripts de R a nuestro home. Para copiar archivos desde y hacia el cluster se usa el **protocolo sftp**.

El cliente y el servidor SSH lo soportan para poder transferir archivos **por el mismo canal encriptado** por el que se envían los comandos.

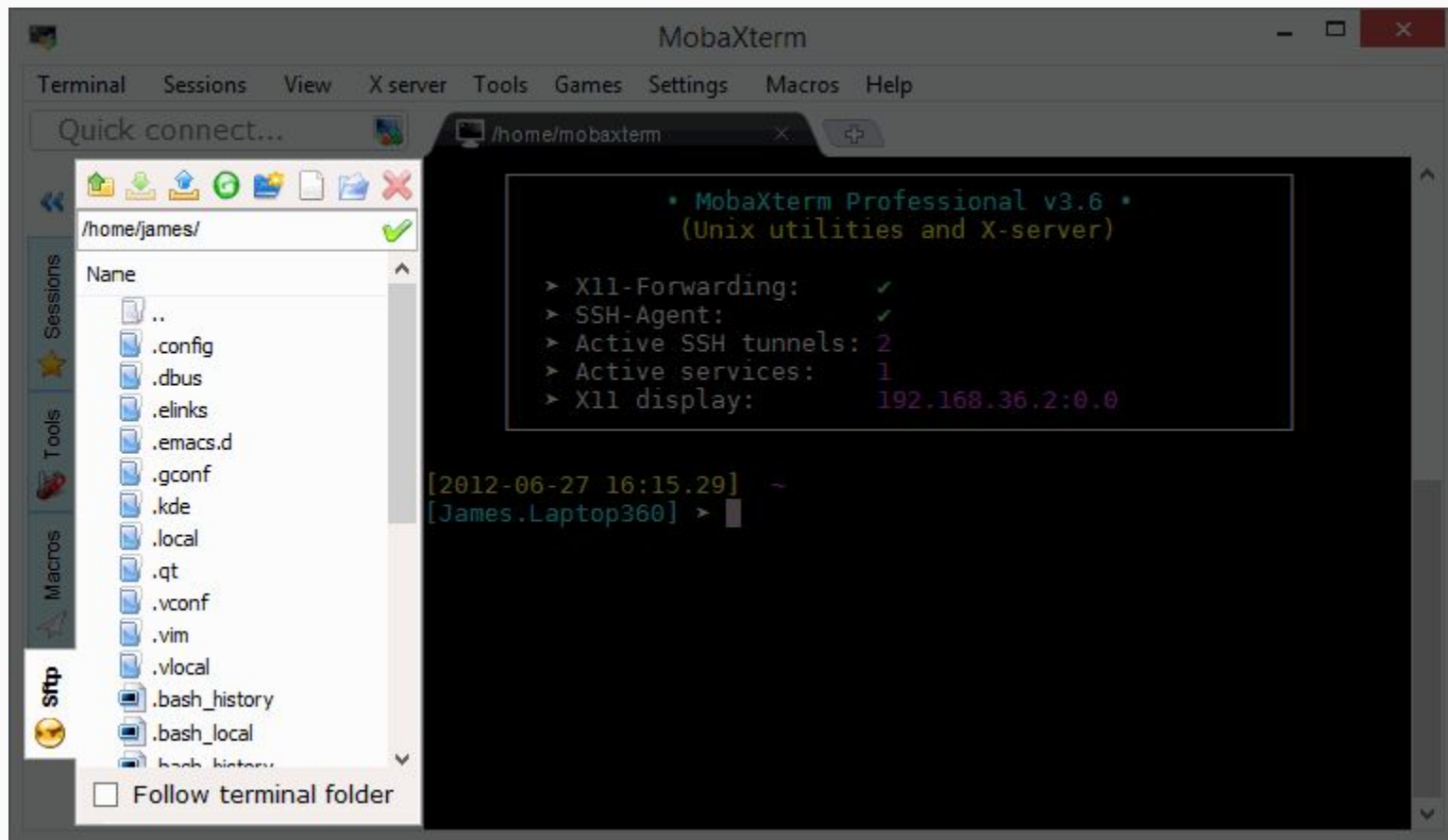
Existen varios exploradores de archivos que también soportan estas conexiones.

SSH: scp

```
[usuario@localhost ~]$ scp hola.R mmazzini@mendieta.ccad.unc.edu.ar:/home/mmazzini/curso
```

```
[usuario@localhost ~]$ scp -r mmazzini@mendieta.ccad.unc.edu.ar:/home/mmazzini/curso .  
data          100% 0      0.0KB/s   00:00  
hola.R        100% 0      0.0KB/s   00:00  
input         100% 0      0.0KB/s   00:00
```

Si vamos a transferir gran volumen de información conviene comprimir los archivos antes “scp -C”.



Conexión SFTP en mobaXterm: permite arrastrar y soltar

Editando archivos

Los archivos se pueden editar desde la línea de comandos con un editor de texto como **nano** o **vi** o con un **editor local** sobre la conexión SFTP.

mobaXterm tiene también un editor de texto interno compatible con LINUX

SSH

Con nuestros archivos copiados al cluster estamos listos para enviar nuestros **scripts no interactivos** al **sistema de colas** para que se ejecuten.

SLURM: Sistema de colas

SLURM - Lanzar simulaciones

La sesión de R en la **cabecera** NO debe usarse para correr simulaciones. Para que nuestros scripts se ejecuten de forma no interactiva en los nodos de cómputo debemos crear un **script de submit** especificando los recursos que necesitaremos y por cuánto tiempo.

SLURM

```
#!/bin/bash
#SBATCH --job-name=test
#SBATCH --partition=mono
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --time=0-01:00

. /etc/profile

module load R

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export MKL_NUM_THREADS=$SLURM_CPUS_PER_TASK

srun Rscript hola.R
```

hola.R

```
cat ("Hola mundo desde", Sys.info() ["nodename"], "\n")
```


SLURM

Una vez que nuestro script de submit está listo se envía al sistema de colas con el comando

```
[usuario@mendieta ~]$ sbatch submit.sh
```

- A partir de ese momento se le asigna un **JobID** y queda **pendiente** esperando que los recursos estén disponibles.
- Cuando comienza a ejecutar (**running**) toda la salida a pantalla se irá escribiendo en el archivo **slurm-JOBID.out**

SLURM: squeue

```
[mmazzini@mendieta ~]$ sbatch submit.sh
[mmazzini@mendieta ~]$ squeue -u mmazzini
PARTI  JOBID PRIOR USER      NAME      ST      TIME NO CPU  GRES NODELIST(REASON)
mono   73878  5279 mmazzini  sbatch    PD      0:00  1  1  (null (Resources)

[mmazzini@mendieta ~]$ squeue -u mmazzini --start
JOBID PARTITION NAME      USER      ST      START_TIME  NODES SCHEDNODES  NODELIST(REASON)
73878      mono  test  mmazzini  PD  2018-10-02T10:15:25      1  (null)      (AssocMaxJobsLimit)

[mmazzini@mendieta ~]$ squeue -u mmazzini
PARTI  JOBID PRIOR USER      NAME      ST      TIME NO CPU  GRES NODELIST(REASON)
mono   73878  5279 mmazzini  test      R      0:02  1  1  (null mendieta08

[mmazzini@mendieta ~]$ cat slurm-73878.out
hola mundo desde mendieta08.mendieta.ccad.unc.edu.ar
```

SLURM: scancel

Si algo va mal podemos cancelar el job

```
[mmazzini@mendieta ~]$ scancel 73878
```

O cancelar todos nuestros jobs

```
[mmazzini@mendieta ~]$ scancel -u mmazzini
```

O cancelar todos los jobs que todavía no corrieron

```
[mmazzini@mendieta ~]$ scancel -u mmazzini -state=pending
```

SLURM: scontrol

```
[mmazzini@mendieta ~]$ scontrol show job 73878
JobId=73890 JobName=test
  UserId=mmazzini(10370) GroupId=mmazzini(10370) MCS_label=N/A
  JobState=COMPLETED Reason=None Dependency=(null)
  Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
  RunTime=00:00:20 TimeLimit=1-00:00:00 TimeMin=N/A
  SubmitTime=2018-10-01T07:39:32 EligibleTime=2018-10-01T07:39:32
  Command=/soporte/mmazzini/script.sh
  ...
```

Recapitulando:

- Rscript / Uso no-interactivo
- Parametrizar el script
- Organizar inputs/resultados
- Descargar mobaXterm
- Generar la llave ssh
- Solicitar cuenta en el formulario
<https://tinyurl.com/rccad>
- Conectarse al cluster
- Instalar paquetes R
- Copiar archivos (scripts inputs)
- Crear el script de submit
- Enviar a correr
- Monitorear las corridas
- Transferir los resultados

soporte@ccad.unc.edu.ar

Muchas Gracias!

