FACULTAD DE CIENCIAS BIOQUÍMICAS Y FARMACÉUTICAS - UNR

Instrucciones para el uso del cluster de Conicet (u otros servidores)

Marcos Prunello

28 de diciembre de 2018

Aclaración: este material es un conjunto de notas sobre el uso de servidores, probablemente no usa términos correctos o específicos, puede tener errores y la redacción fue rápida sin cuidado.

Índice

1. Registro en el cluster de Conicet

Para registrarse como usuario en el cluster hay que seguir las instrucciones disponibles acá.

2. Sistema de claves para el acceso al cluster

Para poder conectarse se utiliza un sistema de claves públicas. En la página del cluster indican los pasos a seguir, tanto para Linux como para Windows. Qué es el sistema de claves públicas: chusmear acá, acá o acá.

3. Acceso remoto con ssh

ssh es el sistema que se emplea para poder conectarse remotamente a un servidor de forma segura. SSH significa Secure Shell y es un protocolo de administración remota que permite a los usuarios controlar y modificar sus servidores remotos a través de Internet. En la primera parte de esta página del cluster hay algunas indicaciones, pero a continuación me extiendo un poco más.

3.1. Con Linux (o macOS)

Se puede hacer *ssh* directamente desde la ventana de la terminal. El comando SSH consta de 3 partes: `ssh {user}@{host}`. `ssh` abre una Conexión de Shell Segura y cifrada, `{user}` representa la cuenta a la que vas a acceder y `{host}` hace referencia al equipo al que desees acceder (puede ser una dirección IP o un nombre de dominio como piluso.rosario-conicet.gov.ar). Al dar enter se te pedirá que introduzcas la contraseña de la cuenta (la que elegiste al crear el par de claves). Al escribirla, no aparece nada en la pantalla, pero anda igual, dar enter al terminar, y si está bien estaremos conectados remotamente a una terminal del servidor.

En resumen, los pasos son:

1. Abrir la terminal.
2. ssh a la cuenta personal, por ejemplo:

   ```
   ssh mprunello.fbioyf@piluso.rosario-conicet.gov.ar
   ```

3. Ingresar contraseña (no se ve nada por cada caracter tipeado) y dar enter.

3.2. Con Windows

Hay que descargar algún programa que permita hacer *ssh* (un *cliente ssh*), como, por ejemplo, Putty, que se descarga de esta página. Ir al cuadro de *Alternative binary files* y descargar el ejecutable que corresponda (32 o 64 bits) debajo del subtítulo *putty.exe*.

Abrir Putty y configurar el tunel de conexión como indica la página del cluster. Esto se hace una sola vez y te queda guardado. Después cuando abrís Putty en el cuadrito *Saved sessions* tiene que aparecer listada, la seleccionás, hacés click en Load y después en Connect. Se tiene que abrir una terminal donde ponés tu clave y te conectás. Toda esta parte de la configuración no la chequeé, la última vez que lo hice en Windows fue en el 2016 y para otro servidor.

4. Transferencia de archivos

4.1. Transferencia de archivos con Linux

Se realiza con el comando `scp`, seguido por el archivo de origen y luego el destino. Por ejemplo, para pasar el archivo `tutorial.txt` desde mi compu (está en una carpeta del escritorio) al servidor (dentro de la carpeta llamada Ejemplo que tengo ahí):

```
scp Desktop/Simulaciones/tutorial.txt mprunello.fbioyf@piluso.ros
ario-conicet.gov.ar:/home/mprunello.fbioyf/Ejemplo
```

Si quiero traer un archivo desde el servidor a la compu, la primera parte es la ubicación del archivo en el servidor, la segunda es en qué carpeta quiero que se guarde en mi compu:

```
scp mprunello.fbioyf@piluso.rosario-conicet.gov.ar:/home/mprun
ello.fbioyf/Ejemplo/correrSimulaciones.R /home/marcos/Documents
```

Puedo traer varios archivos que tengan algún patrón:

```
scp mprunello.fbioyf@piluso.rosario-conicet.gov.ar:/home/mprun
nello.fbioyf/Ejemplo/*.txt /home/marcos/Documents
```

También podría traer toda la carpeta, con todas las cosas que haya en ella. En el destino indicado en mi compu se creará esta misma carpeta:

```
scp -r mprunello.fbioyf@piluso.rosario-conicet.gov.ar:/home/mprun
ello.fbioyf/Ejemplo /home/marcos/Documents
```

Alternativamente, de forma más sencilla con el explorador de archivos de Ubuntu (Nautilus) se puede acceder al server y usar los archivos como estamos acostumbrados (copiar, pegar, eliminar, crear nuevos, etc). Puedo copiar y pegar archivos entre la compu local y el servidor. Para esto, en el explorador de archivos mirar el panel de la izquierda e ir a Other Locations, luego en la barra de abajo ir a Connect to server y poner sftp://mprunello.fbioyf@piluso.rosario-conicet.gov.ar. Me pide la contraseña y me lleva directamente al servidor. Ahí vemos todas las carpetas del servidor, de hecho dentro de home puedo ver la carpeta restringida a cada usuario, puedo entrar sólo a home/mprunello.fbioyf, donde tengo mis archivos. Puedo copiar y pegar como si fuera una carpeta más en mi computadora. Puede ser útil agregarla a la barra de favoritos para entrar directamente.

4.2. Transferencia de archivos con Windows

También hay que instalar un *scp client* para poder hacer transferencia de archivos desde y hacia el servidor en Windows. Una opción es Putty SCP (pscp) pero yo uso WinSCP, que una vez conectado te permite ver en una especie de explorador de archivos como el de siempre de Windows tu computadora en la parte izquierda y el servidor en la parte derecha. Se descarga de acá.

La primera vez tenés que cargar los datos de la sesión, después ya podés dejarlos cargados. En *hostname* ponés el dominio del servidor, por ejemplo, `piluso.rosario-conicet.gov.ar` y completás con tu usuario y contraseña. Para guardar hacés click en *Save* y te queda todo esto registrado (la contraseña opcionalmente). Luego hacer click en *Login*. Si no hiciste que se guarde la contraseña te la pide después de hacer click en *Login*.

5. Usar R en el servidor

5.1. Versiones y paquetes de R en el cluster de Conicet

En general, ejecutás el comando R en la consola y se abre la consola de R, con la versión que esté agregada por default al entorno. Cuando yo me registré (primer cuatrimestre 2017), tenían instalado 3.0.0, pedí que me pongan la última versión y al toque me instalaron 3.4.1. Sin embargo, el que aparecía por default si ejecutaba `R` seguía siendo el 3.0.0. Para ejecutar el 3.4.1 había que buscarlo en `/share/apps/R-3.4.1/bin/R`. Poner exactamente eso si quiero abrir R 3.4.1, e incluso ponerlo así en los scripts .sh con el programa a correr.

Los paquetes instalados los puedo ver así: `ls /share/apps/R-3.4.1/library/` (están los que yo pedí también).

Actualmente, haciendo `ls /share/apps/` veo que está el R 3.5.1. Sin embargo, no tienen los paquetes que yo hice instalar en 3.4.1 (es decir que tendría que volver a pedirlos). Sólo tiene los de R Base: `ls /share/apps/R-3.5.1/lib64/R/library`

5.2. Correr un trabajo de R en el servidor

Voy a tomar de ejemplo unos archivos que hice para correr las simulaciones del proyecto de la cátedra. Necesitaba correr un programa de R que:

- Lea argumentos desde la consola.
- Cargue código de otro script de R con `source()`.
- Use paquetes.
- Corra en paralelo.

- Vaya mostrando el progreso.
- Escriba archivos de salida.

Todo esto está contemplado en la situación de prueba que preparé en la carpeta `Ejemplo`, la cual tiene los siguientes archivos:

- **correrSimulaciones.R**: Este es el que deseo correr para hacer mi análisis. Es el que lee los argumentos pasados a través de la consola por el usuario, corre en paralelo, usa paquetes, escribe archivos y carga un conjunto de funciones que definí yo que uso para el análisis y están guardadas en el archivo funcionesParaSimular.R
- **funcionesParaSimular.R**: archivo auxiliar con funciones que uso en mi análisis. Este archivo es cargado por correrSimulaciones_test.R y es el que tiene el foreach que se encarga de la paralelización.
- **escenarios.Rdata**: un archivo que uso para armar los escenarios de las simulaciones.

Entonces para correr este programa desde la consola en mi sesión actual:

```
module load gcc-6.3.0
cd Ejemplo
/share/apps/R-3.4.1/bin/Rscript correrSimulaciones.R 3 1.5 1 0 lognormal "~/Ejemplo/" 1 3 20
```

La línea de `module` es para cargar un módulo sin el cual no se pueden usar los paquetes de R instalados (ni idea por qué, sin eso da error). Todo lo que está después de `correrSimulaciones.R` son los argumentos que le paso al código (elegidos para que corra rápido en este ejemplo). Cuando termina se habrán generado en esta carpeta los siguientes archivos con resultados:

- corrida1.txt
- esc*...(uno por cada escenario)

Esto funciona bien, pero así no estoy aprovechando el cluster porque sólo estoy corriendo localmente en mi sesión. Para poner en cola mi trabajo y usar más cores tengo que hacer un script de bash como se indica más adelante.

5.3. Monitorear el progreso en R y escribir mensajes

Esto es opcional y lo pongo sólo para registrar algo que me pasó. En un primer intento usé el paquete doParallel para correr en paralelo, que me permite ir guardando en un archivo de texto llamado *mensajes.txt* comentarios para chequear el avance impresos a través de `cat()`. Esto es parte del archivo que mando a correr con Rscript:

```
# Paralelo
library(doParallel)
cl <- makeCluster(ncores, outfile = paste0(pathOutput, "mensajes.txt"))
registerDoParallel(cl)

# Archivo de salida
writeLines("", "mensajes.txt")
sink("mensajes.txt", append = T)

# todo el codigo, con llamadas a cat()... stopCluster(cl)
sink()
```

Sin embargo, cuando mandé a correr en paralelo en cluster, no se me escribía este archivo así que tuve que buscar otra alternativa. Ahí me pasé al paquete doSNOW que permite usar la barra de progreso:

```
# Paralelo
library(doSNOW)
cl <- makeCluster(cores)
registerDoSNOW(cl)
progressBar <- txtProgressBar(max = nsim, style = 3)
```

```
progress <- function(n) setTxtProgressBar(progressBar, n)
opts <- list(progress = progress)

# todo el codigo...

close(progressBar)
stopCluster(cl)
```

6. Usar el sistema de colas para correr trabajos

6.1. Encolar un trabajo

La línea que corre R del ejemplo anterior tiene que estar en un script de bash, que incluya otras opciones para encolar el trabajo. El material de utilización del cluster tiene una explicación muy detallada, yo acá sólo incluyo lo que usé.
Mi archivo de ejemplo se llama `simScript.sh` y su contenido es:

```
#!/bin/bash
#
# Script para aprender a enviar un job.
# Probar la utilizacion de R en paralelo
#
# Opciones SGE
#
# Nombre para este job:
#$ -N simTest
# Para corrida en paralelo con mpi (ni idea])
#$ -pe impi 8
# Para que tome como pwd el que estoy usando ahora
#$ -cwd
# Setear limite de tiempo de corrida, estos es obligatorio, si se alcanza
#ese limite se mata el proceso, formato: hh:mm:ss
#$ -l h_rt=00:10:00
# MPIR_HOME, importo variables de entorno del SGE (ni idea)
#$ -V
# Me dijeron que agregue esta linea para poder usar los paquetes:
module load gcc-6.3.0
# Ahora mi sentencia que corre el programa de R
/share/apps/R-3.4.1/bin/Rscript correrSimulaciones.R 3 1.5 1 0
lognormal "~/Ejemplo/" 1 3 20
```

En ese archivo las líneas se comentan con numeral, pero numeral con signo peso setea opciones para el sistema de colas.
Este trabajo se manda al sistema de colas con:

```
qsub simScript.sh

Your job 916283 ("simBifactorial") has been submitted
```

6.2. Comandos que se pueden utilizar una vez enviado el trabajo

Para cancelar y eliminar un trabajo:

```
qdel 916283
```

Para consultar el estado de un trabajo:

- `qstat`: Muestra el estado de todos los trabajos sin tener en cuenta el estado de las colas.

- `qstat -u nombre_usuario`: Monitorización del estado de los trabajos de un usuario. Ejemplo: `qstat -u mprunello.fbioyf`
- `qstat -f`: Lista toda la información sobre trabajos y colas para todos los usuarios (no lo usé).
- `qstat -F`: Muestra el estado de todos los parámetros de todas las colas, por ejemplo, la carga, uso de memoria, swap, etc (no lo usé).
- `qstat -j id_job` : Da la razón de por qué un trabajo pendiente no ha sido planificado (no lo usé).

Si el trabajo ya no aparece es porque terminó. En `qstat`, los estados pueden ser:

- t: transfer to the system
- r: running
- s: suspended
- S: suspended by the queue. Work is suspended for internal scheduling management, by loading or by priority queue scheduler. The work will continue running when this is possible.
- T: has reached the limit of the tail (Treshold reached).
- w: waiting
- h: hold
- e: error
- q: queued

El comando `qmon` abre una interfaz gráfica. Se necesita tener instalado algo me parece, ahora no lo tengo, no lo puedo chequear.

7. Screen

Screen es una herramienta muy útil que uso todo el tiempo en el servidor del grupo de Stanford. Conectarse por ssh tiene algunas limitaciones, por ejemplo, si querés abrir otra ventana tenés que hacer otra conexión ssh desde otra terminal, o si se desconecta, perdés lo que estabas corriendo (ojo, esto no pasa si el trabaja fue metido en el sistema de colas en el cluster de Conicet, va a seguir corriendo aunque me desconecte). Para ese tipo de casos sirve **Screen**, que es un administrador de la ventana que te permite con una sola conexión ssh trabajar en múltiples ventanas al mismo tiempo. Por ejemplo, en una ventana largo a correr un trabajo de R, en otra ventana voy corriendo otro trabajo (total el servidor tiene muchos cores), en otra ventana voy viendo los archivos que se van generando, en otra ventana puedo estar editando un script, etc, etc.

En general ya está instalado y si no, se instala con: `sudo apt-get install screen`. Para que se inicie el comando es `screen`. La terminal abre UNA sola ventana en el mismo directorio donde estaba. Si estás probando, ejecutar algún comando en esta ventana, por ejemplo, `ls`, para poder diferenciar esta de las otras que vamos a crear.

Todos los comandos para screen se indican con `CTRL + a`. Para crear una segunda ventana, `CTRL + a`, luego `c`. Se abre una nueva ventana, en limpio, la anterior no se ve más (también ejecutar algún comando, puedo ir trabajando de manera independiente en cada una).

Para pasar de una ventana a otra, `CTRL + a + n` (next), va pasando en orden una por una a la siguiente. Para ir a la previa, `CTRL + a + p`. Para cerrar una ventana, tipear `exit` y volvés a la pantalla anterior o a la terminal principal de donde abriste screen si era la última pantalla que te quedaba. También para cerrar una ventana sola: `CTRL + a + k` (kill), te pide confirmar con `y`.

La ventaja de screen es que se puede despegar (detach) una de las sesiones (con todas sus pantallas) para que el proceso siga corriendo en el fondo por más de que salgamos y volvamos a la pantalla principal para hacer otra cosa. El comando es `CTRL + a + d`. O sea que puedo dejar corriendo cosas, desconectarme (a propósito o por un

problema de conexión a internet), conectarme más tarde desde otro lado y todo siguió corriendo.

Si volví a la pantalla principal, puedo ver todas las sesiones de screen con `screen -ls` (porque puedo tener más de un screen, cada uno con múltiples pantallas). Cada sesión de screen tiene un nro de id y también le puedo poner nombre. Puedo retomar una sesión de screen con `screen -r` (resume), con eso alcanza si tengo una sola, si tengo varias sesiones (no lo he usado) poner a continuación el nro de id (o los primeros dígitos únicos). Sólo puedo retomar una sesión si había sido *detached*. Si no lo fue, cuando hago `screen -ls` voy a ver que figura como *attached*. En ese caso primero hay que detach con `screen -d nroID` y luego retomarla con `screen -r nroID`.

Una que usé mucho es `screen -D -r`, para retomar alguna sesión existente o para crear una nueva si no había ninguna.

Hay muchas opciones más, como partir la pantalla para verlas todas juntas. Cualquier cosa buscar referencia en internet, por ejemplo, acá, acá o acá.

8. Dudas y cosas para escribir pendientes

- Cómo mejorar la utilización de los cores para correr en paralelo. En los instructivos de Conicet hay muchas opciones más, si lo necesito debería volver a que me expliquen. Por ahora mis scripts de R determinan el número de cores y se encargan de la paralelización.
- Agregar comandos útiles de bash y vim.