De 0 al CCAD en 90 min.

Lic. Marcos Mazzini - CCAD / UNC
CPA CONICET

CCAD - UNC

Centro de Computación de Alto Desempeño

Contamos con varios cluster con variadas aplicaciones para simulaciones computacionales:

http://ccad.unc.edu.ar

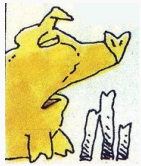
Solicitar acceso en

servicios -> pedido de cuentas

Los clusters

Clusters



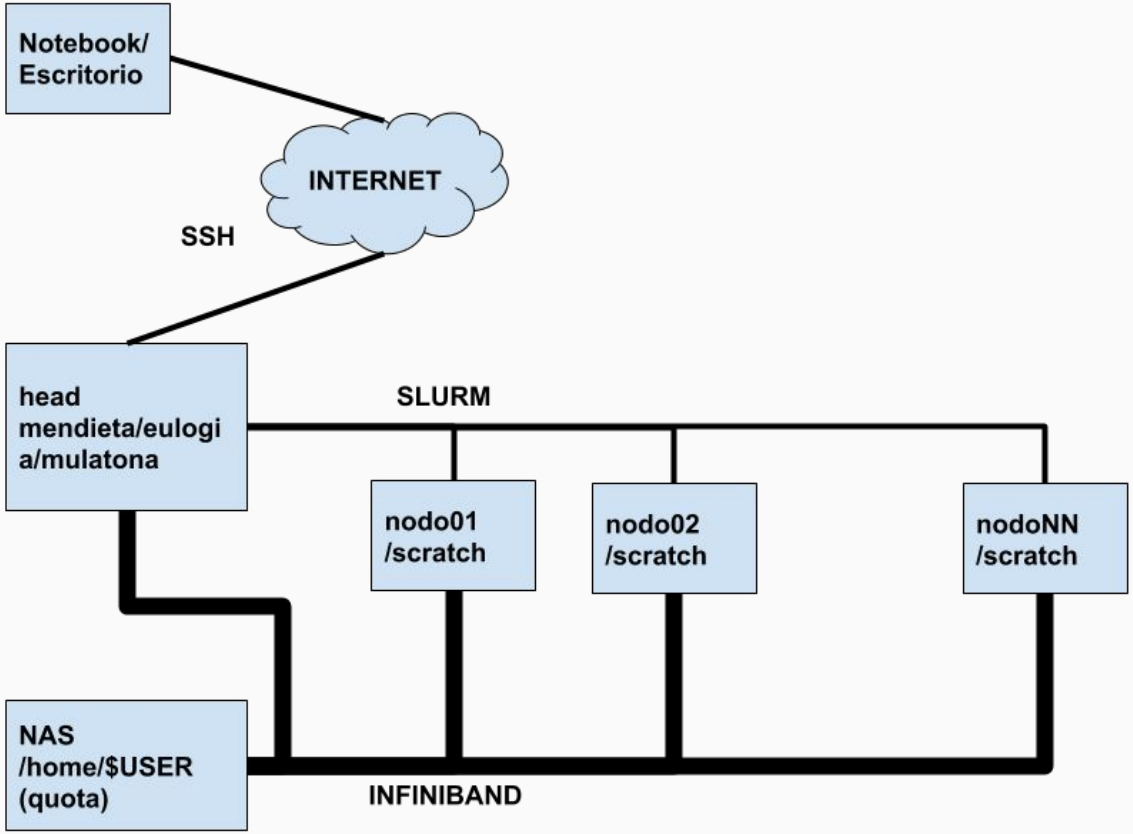



Mendieta: Uso general, el de más trayectoria en el centro

Eulogia: Especializado en cálculos altamente paralelizables

Mulatona: Uso prioritario para el IATE, compartido para el resto

Nabucodonosor: Nodo para Machine Learning



Esquema de conexiones en un cluster

Herramienta fundamental: Shell - Bash

¿Qué es un Shell?

- Es un **programa** especial que permite controlar un equipo mediante texto
- También se conoce como **consola**, **terminal** o **CLI** (command-line-interface)
- Se llama shell porque vendría a ser la última capa alrededor del kernel
- Es un intérprete de comandos REPL
  - Read: leer el comando
  - Eval: evaluar, interpretar, verificar, ejecutar
  - Print: imprimir el resultado (o el error)
  - Loop: repetir

Permite

- Administrar **archivos** (copiar, mover, borrar,editar)
- Administrar **procesos** (ejecutar y cerrar aplicaciones)
- Ejecutar **scripts** (secuencias de comandos guardados en un archivo)
- Monitorear y configurar el Sistema Operativo

Es la forma estándar de interactuar con servidores, sistemas remotos o sin interfaz gráfica

Shells

- La gran mayoría de los SO ofrece una terminal (o varias)
- Hay varias terminales para LINUX, **BASH** es la más extendida

Bash (Bourne-again Shell)

- Es una evolución de **Bourne shell** de UNIX (sh)
- Bash es compatible con sh, cumple con la estandarización POSIX y agrega extensiones

Emuladores de terminal

Como la mayoría de los usuarios usa una interfaz gráfica lo que se termina utilizando es un **emulador de terminal**

- Windows: PuTTY, MoabXterm
- Linux: Terminal, Konsole, XTerm, Terminator
- Mac OS X: Terminal (default), iTerm 2

Terminal Linux

Por medio del menú de aplicaciones puede accederse al emulador, dependiendo de la distribución puede estar ubicado en distintos lugares.

Para una consola modo texto puede usar la combinación Ctrl+Alt+F1 y luego retornar al modo gráfico con Ctrl+Alt+F7 (podrían ser otras “F”)

Iniciando una terminal: El Prompt

Cada vez que abrimos una terminal o nos conectamos a un sistema remoto veremos el motd (message of the day) y el **prompt**

El **prompt** es personalizable, como todo en linux, así que podría no verse igual en todos lados.

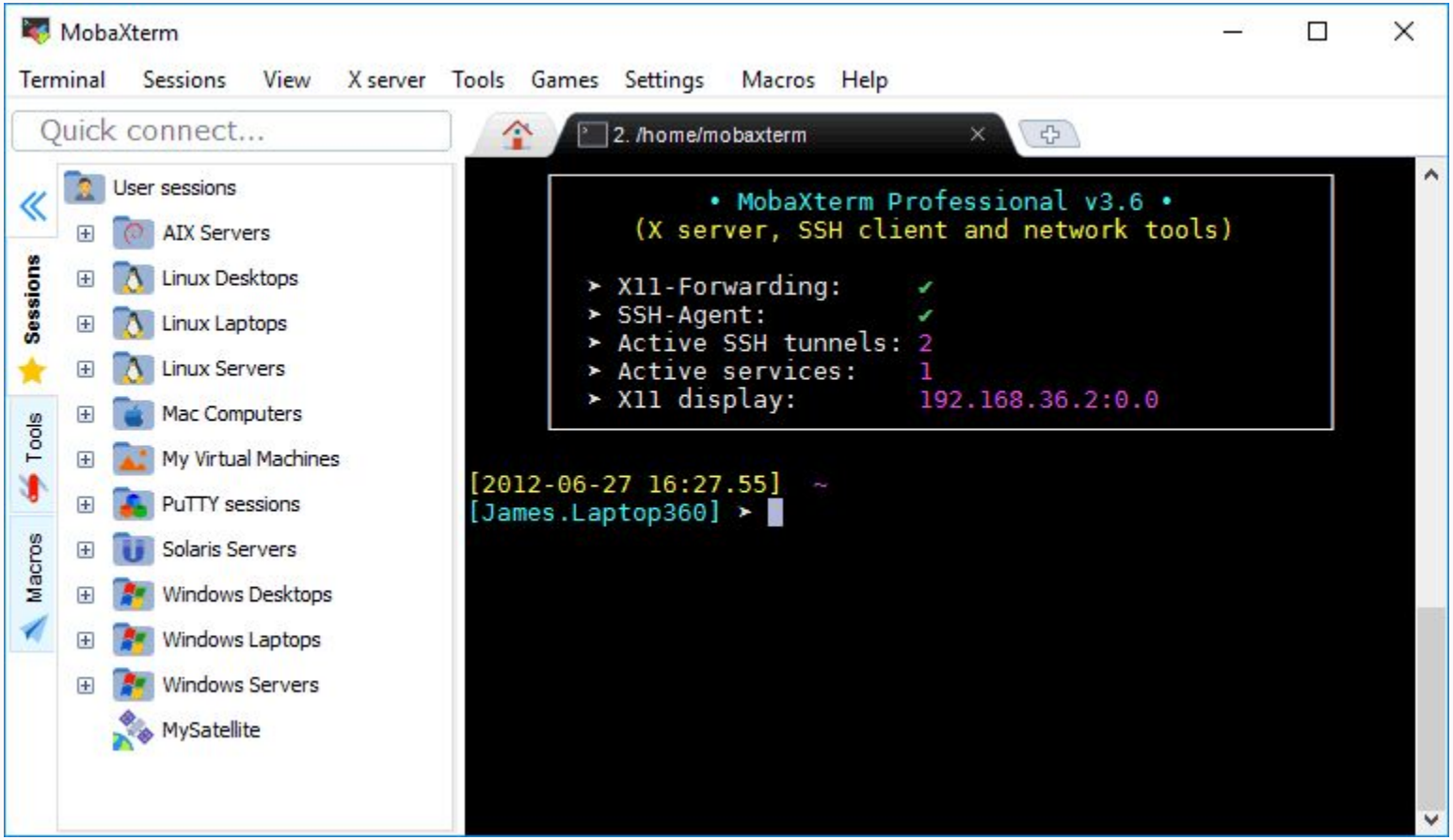
El Prompt

```
usuario@localhost:~$
```

- **usuario**: Nombre de usuario del usuario actual
- **localhost**: nombre del equipo al cual ingresamos
- **~**: carpeta actual, ~ es un atajo a **/home/usuario**
- **$**: fin del prompt, inicio del input

Prompt del usuario “root”, logueado al servidor “server” en la carpeta “/var/log”:

```
root@server:/var/log#
```



Interfaz de MoabXterm

Los Comandos

Los Comandos

- Se ejecutan tipeando el nombre del archivo ejecutable seguido de ENTER
- Puede ser un **binario** o un **script**
- Al comando corriendo se lo conoce como **proceso**
- Por defecto un proceso corre en **foreground** (primer plano), esto implica que hay que esperar a que termine para poder ejecutar el siguiente comando (**Ctrl-c** envía señal de finalización)

Comandos: sintaxis para argumentos

Sin argumento	**$ls**
Con argumento	**$ls /usr/bin**
Con opciones (flags)	**$ls -l** (corta) **$ls --all** (larga) **$ls -la** (combinada ls -l -a) **$ls --sort=size** (larga con parámetro)
Con argumento y opciones	**$ls -la /home**

comandos: ls, echo

```
[usuario@localhost ~]$ ls
curso1  curso2
[usuario@localhost ~]$ ls -l
drwxrwxr-x. 2 usuario usuario 4096 abr 23 10:57 curso1
drwxrwxr-x. 3 usuario usuario 4096 abr 23 11:06 curso2
[usuario@localhost ~]$ ls -la /home
total 12
drwxr-xr-x.  3 root   root 4096 ene 24 11:59 .
dr-xr-xr-x. 19 root   root 4096 abr 23 09:38 ..
drwx------. 29 usuario usuario 4096 abr 23 09:38 usuario
[usuario@localhost ~]$echo una   linea   con   muchos  espacios
una linea con muchos espacios
```

los paths (rutas)

- Se usa **/** (shift-7) como separador, al revés que en windows
- Sirve para referenciar archivos o carpetas en una estructura de árbol
- La raíz es /
- Puede ser **absoluto** (completo) o **relativo** (a partir de la posición actual)

comandos: pwd, cd, cp, ls

```
[usuario@localhost ~]$ pwd
/home/usuario
[usuario@localhost ~]$ cd curso1
[usuario@localhost curso1]$ cd ../curso2
[usuario@localhost curso2]$ cp ../curso1/texto1.txt .
[usuario@localhost curso2]$ ls
texto1.txt
```

Historial y Autocompletado (tab)

Cada línea ingresada queda registrada y es navegable con las flechitas.

Se puede revisar todo con el comando **history** y repetir alguno sin volver a tipear.

La tecla <tab> autocompleta de forma similar al predictivo del celular para:

- comandos
- nombres de archivos
- variables

comandos: history, !

```
[usuario@localhost ~]$history
1000  ls
1001  ls /usr/bin
1002  ls -l
1003  ls --all
1004  ls -la
1005  ls -la /home
1006  history
[usuario@localhost ~]$ !1005
ls -la /home
total 12
drwxr-xr-x.  3 root   root 4096 ene 24 11:59 .
dr-xr-xr-x. 19 root       root 4096 abr 23 09:38 ..
drwx------. 29 usuario usuario 4096 abr 23 09:38 usuario
```

comandos

```
[usuario@localhost ~]$ ls /usr/<tab><tab>
bin/     games/   lib/     libexec/ sbin/    src/
etc/     include/ lib64/   local/   share/   tmp/
[usuario@localhost ~]$ ls /usr/lib<tab><tab>
lib/     lib64/   libexec/
[usuario@localhost ~]$ ls /usr/lib
```

links

Son atajos o accesos directos y se muestran como origen->destino

Lo que más se utiliza es el **soft-link** que puede ser a un archivo o a una carpeta, si el destino se borra el link queda pero roto

El **hard-link** solo puede ser a un archivo y referencia el contenido de este (inodo), es como un nombre alternativo

comandos: ln, rm

```
[usuario@localhost ~]$ cd curso1
[usuario@localhost curso1]$ ln -s ../curso2 c2
[usuario@localhost curso1]$ ll
total 0
lrwxrwxrwx. 1 usuario usuario 9 abr 23 18:16 c2 -> ../curso2
-rw-rw-r--. 1 usuario usuario 0 abr 23 10:35 texto1.txt
-rw-rw-r--. 1 usuario usuario 0 abr 23 10:36 texto2.txt
[usuario@localhost curso1]$ rm c2
[usuario@localhost curso1]$ ll
total 0

-rw-rw-r--. 1 usuario usuario 0 abr 23 10:35 texto1.txt
-rw-rw-r--. 1 usuario usuario 0 abr 23 10:36 texto2.txt
```

Archivos

En LINUX todo se abstrae como un archivo y consecuentemente existen varios comandos para manejar líneas de texto.

El **editor de texto** básico es **nano**.

vi y **emacs** son editores avanzados que una vez que se domina alguno ofrecen muchas ventajas

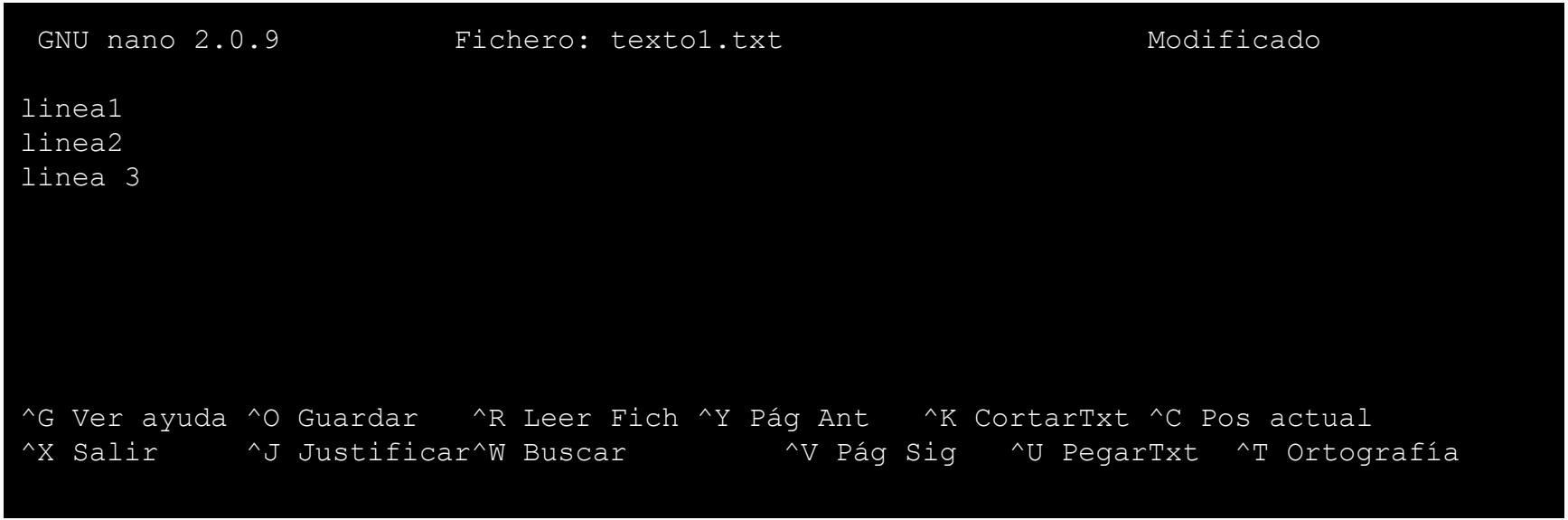
MoabXterm ofrece un editor local compatible con LINUX

comandos: cat, nano

```
[usuario@localhost curso1]$ cat texto1.txt
linea1
linea2
linea 3
[usuario@localhost curso1]$ cat texto1.txt texto2.txt
linea1
linea2
linea3
linea1
linea2
linea 3
[usuario@localhost curso1]$nano texto1.txt
```

comandos : nano

^<tecla> = ctrl+<tecla>



comandos: head, tail, less

```
$ less /usr/share/dict/words
1080
10-point
10th
11-point
12-point
16-point
18-point
1st
2
20-point
2,4,5-t
2,4-d
:
```

<arriba>, <abajo>,pgup,pgdwn, q

```
[usuario@localhost ~]$ head -5 /usr/share/dict/words
1080
10-point
10th
11-point
12-point
[usuario@localhost ~]$ tail -5 /usr/share/dict/words
zyzzyvas
ZZ
Zz
zZt
ZZZ
```

```
$ <comando_salida_extensa> | less
```

comandos: grep, *glob/pathname expansion

```
[usuario@localhost curso1]$ grep ea2 texto1.txt
lin ea2
[usuario@localhost curso1]$ grep "linea 3" texto*
texto1.txt:linea 3
texto2.txt:linea 3
[usuario@localhost curso1]$
```

Sistema de archivos

Las distribuciones LINUX tienen algunas diferencias entre sí pero en general tienen **las mismas carpetas** o directorios. Es porque tratan de cumplir con la estandarización POSIX que define los usos para cada una.

A continuación veremos qué finalidad tiene cada carpeta:

comandos

```
[usuario@localhost /]$ ls -1
bin
boot
dev
etc
home
lib
lib64
lost+found
media
```

```
mnt
opt
proc
root
run
sbin
srv
sys
tmp
usr
var
```

/ (raiz-root)	C: y otras particiones también. No hay nada fuera del raiz
/bin	Binarios esenciales del usario: tienen que estar si o si aun en recovery mode.En /usr/bin van el resto. En /sbin van los de admin
/boot	Archivos necesarios para iniciar el sistema
/cdrom - /media	Punto de montaje para medios removibles (/cdrom es historico)
/dev	Representan dispositivos, particiones (/dev/sda1) o dispositivos virtuales (/dev/random o /dev/null)
/etc	Archivos de configuración de todo el sistema. Son modo texto editables con cualquier editor. Las configuraciones por usuario van en el home de cada uno
/home	Hay una carpeta para cada usuario. Es el único lugar donde cada usuario puede escribir (a no ser que sean root)
/lib, /lib64	Librerías esenciales para los binarios esenciales (/bin). El resto está en /usr/lib

lost+found	Archivos perdidos por un crash del sistema (favorece el recovery)
/mnt	Punto de montaje temporal (igual se puede montar en cualquier lugar)
/proc	Similar a /dev representa información del sistema y de los procesos
/root	Es el home del usuario root. No confundir con /
/run	Archivos temporales, a diferencia de /tmp estos no se borran
/srv	Datos para los servicios provistos por el sistema. Ej. httpd o ftp
/tmp	Archivos temporales, podrían ser borrados en cualquier momento
/usr	Binaros de usuario y datos de solo lectura (/usr/share y /usr/local contienen los que fueron compilados por el usuario)
/var	Es la contraparte escritura de /usr. Los logs van en /var/log

Además en el CCAD:

/opt	Originalmente software opcional, es donde los administradores instalamos los programas y librerías que se ejecutarán en los nodos
/scratch	Espacio temporal de disco local en cada nodo

comandos: mv, mkdir, touch, rm, rmdir

```
[usuario@localhost ~]$ cd curso1
[usuario@localhost curso1]$ cp texto1.txt texto_copia.txt
[usuario@localhost curso1]$ mv texto_copia.txt texto1_copia.txt
[usuario@localhost curso1]$ mkdir extra
[usuario@localhost curso1]$ touch extra/algo
[usuario@localhost curso1]$ ls extra/
total 0
-rw-rw-r--. 1 usuario usuario 0 abr 23 18:57 algo
[usuario@localhost curso1]$ rm extra/algo
[usuario@localhost curso1]$ rmdir extra
```

Usuarios y grupos

Linux es un sistema **multiusuario**, el sistema de por sí cuenta con varios

Cada usuario tiene un nombre y **número** único (UID)

También pertenece a uno o más **grupos** (GID)

root es el nombre de usuario para administración, es superusuario y tiene acceso a todo el sistema (lo tienen también usuarios miembros del grupo wheel o sudoers)

comandos: id, w, su

```
[usuario@localhost ~]$ id usuario
uid=1000(usuario) gid=1000(usuario)
grupos=1000(usuario),10(wheel),984(vboxusers)
[usuario@localhost ~]$ w
 19:01:22 up  9:23,  3 users,  load average: 0,35, 0,57, 0,57
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
usuario  tty1     :0               09:38    9:23m  3:41   1.22s mate-session
usuario  pts/0    :0               10:12    7:32m  0.04s  0.04s /bin/bash
usuario  pts/1    :0               17:15    2.00s  0.19s  0.00s w
[usuario@localhost ~]$ su
Contraseña:
[root@localhost usuario]# cd
[root@localhost ~]# pwd
/root
```

Permisos

Cada archivo y carpeta pertenece a **un usuario** y a **un grupo**. Sobre éstos se disponen permisos de **lectura**, **escritura** y **ejecución** para:

- el propietario (owner)
- el grupo
- otros

Permisos

Hay 3 cosas que se pueden hacer un archivo:

read **w**rite e**x**ecute

```
$ls -l archivo.txt
rw-rw-r--  usuario   grupo    …
```

- el usuario puede leer y escribir
- el grupo puede leer y escribir
- los otros solo pueden leer

Permisos (directorios)

```
$ ls -l
drwxr-xr-x    usuario  usuario  carpeta
```

- **r**: listar
- **w**: crear archivos
- **x**: acceder y modificar archivos

También se pueden ver los permisos en formato binario, permitido (1/0)

rw-	r- -	r - -
110	100	100
6	4	4

comandos: chmod, chown, sudo, exit

```
[usuario@localhost curso2]$ touch nada
[usuario@localhost curso2]$ ll
-rw-rw-r--. 1 usuario usuario 0 abr 23 20:41 nada
[usuario@localhost curso2]$ chmod go-rw nada
[usuario@localhost curso2]$ ll
-rw-------. 1 usuario usuario 0 abr 23 20:41 nada
[usuario@localhost curso2]$ chown root.root nada
chown: cambiando el propietario de «nada»: Operación no permitida
[usuario@localhost curso2]$ sudo chown root.root nada
[usuario@localhost curso2]$ ll
-rw-------. 1 root root 0 abr 23 20:41 nada
[usuario@localhost curso2]$ cat nada
cat: nada: Permiso denegado
[usuario@localhost curso2]$ su
Contraseña:
[root@localhost curso2]# cat nada
[root@localhost curso2]# exit
```

Los Procesos

Procesos

Como dijimos antes un comando ejecutándose se convierte en un **proceso** al que se le asigna un identificador de proceso **PID.**

Un proceso puede crear otros procesos hijos, cuando termina el padre terminan todos los hijos.

Hay procesos que se están ejecutando todo el tiempo **en segundo plano** a los que se los conoce como **demonios** o **daemons.**

Los demonios proveen servicios a los otros procesos.

comandos: ps, pgrep, kill, pkill

```
[usuario@localhost ~]$ ps
  PID TTY          TIME CMD
 4182 pts/1  00:00:00 bash
 5171 pts/1  00:00:00 ps
[usuario@localhost ~]$ pgrep -u usuario
.... <todos los del usuario> ...
[usuario@localhost ~]$ pgrep firefox
2010
[usuario@localhost ~]$ kill -9 2010
[usuario@localhost ~]$ pkill firefox
```

comandos: top

```
[usuario@localhost ~]$ top
top - 19:18:53 up  9:41,  3 users,  load average: 0,41, 0,41, 0,48
Tasks: 155 total,   1 running, 154 sleeping,   0 stopped,   0 zombie
%Cpu(s):  3,2 us,  0,8 sy,  0,0 ni, 94,8 id,  1,1 wa,  0,0 hi,  0,1 si,  0,0 st
KiB Mem :  3323796 total,   494604 free,  2451908 used,   377284 buff/cache
KiB Swap:  7812092 total,  7480368 free,   331724 used.   465384 avail Mem

  PID USER      PR  NI    VIRT  RES   SHR S  %CPU %MEM     TIME+ COMMAND
 2131 usuario   20   0 3447472 1,261g  93932 S   6,0 39,8  85:51.02 Web Content
 1425 root      20   0  355440  83152  68584 S   3,7  2,5   4:06.71 X
 2010 usuario   20   0 9198812 518668 104732 S   2,3 15,6  23:31.93 firefox
 4173 usuario   20   0  643660  14252   6068 S   1,3  0,4   0:07.54 mate-termi+
 5203 usuario   20   0  157716   2228   1544 R   0,3  0,1   0:00.03 top
    1 root      20   0  128164   2320    688 S   0,0  0,1   0:02.16 systemd
    2 root      20   0       0          0     0 S   0,0  0,0   0:00.01 kthreadd
    3 root      20   0       0          0     0 S   0,0  0,0   0:00.34 ksoftirqd/0
    5 root       0 -20       0          0     0 S   0,0  0,0   0:00.00 kworker/0:+
```

comandos: ¿cuántos más hay?

- builtins
  - cd, echo, exit, pwd, history,...
- paquete coreutils (~105) - POSIX
  - https://www.gnu.org/software/coreutils/manual/html_node/index.html
- instalados por la distribución, el admin o el usuario

investigar: find, wget, tar, gunzip

comandos: cómo obtener ayuda

- `$<comando> --help`
- `$<comando> -h` (si -h no se usa para otra cosa)
- `$help <comando>` (para bultins)
- `$man <comando>`
- `$man -k <palabra clave>`
- `$apropos <palabra clave>`

- explainshell
- stackoverflow

SSH: Conectándose al cluster

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 conectarse por ssh se abre un shell y todos los comandos que introducimos se transmiten hacia el servidor por un túnel encriptado.

SSH

La conexión SSH está implementada con un modelo **cliente-servidor**

Para establecer la conexión el **servidor** debe estar ejecutando un proceso (sshd) que escuche por conexiones en un puerto (22 por default). El servidor se encarga de **autenticar la conexión** y **lanzar el entorno** cuando el usuario provee las credenciales correctas.

El **cliente** (máquina local) debe ejecutar un **cliente ssh**, que se encarga de transmitir el nombre de usuario, las credenciales y cada comando que se va ingresando.

SSH

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

En otros sistemas se puede utilizar algún emulador de terminal de los listados antes.

Para windows recomendamos **moabXterm.**

SSH

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

Los mecanismos más extendidos para autenticación son **contraseña** y **llave publico-privada.**

comandos: ssh, w, exit

```
[usuario@localhost ~]$ ssh mmazzini@mendieta.ccad.unc.edu.ar
Last login: Mon Apr 23 11:16:47 2018 from 152.168.141.237
…<motd>
Pedidos al soporte del CCAD soporte@ccad.unc.edu.ar.
**************************
[mmazzini@mendieta ~]$ w
 11:30:36 up 14 days,  3:47, 15 users,  load average: 0,22, 0,27, 0,43
USER     TTY        FROM             LOGIN@     IDLE   JCPU   PCPU WHAT
jborioni pts/1      200.12.138.65    Fri14      2days  0.66s  0.66s -bash
bc       pts/8      :50              16Apr18    4days  0.01s  0.01s bash
gsoldano pts/9      200.16.18.156    10Apr18    6days  1.16s  1.16s -bash
dmasone  pts/10     200.12.137.60    10:11      1:18m  0.05s  0.05s -bash
mmazzini pts/13     152.168.141.237  11:30      0.00s  0.02s  0.01s w
jolmos   pts/15     200.16.18.224    10:48      0.00s  0.29s  0.29s -bash
[mmazzini@mendieta ~]$ exit
logout
Connection to mendieta.ccad.unc.edu.ar closed.
```

SSH

El login por **contraseña** es vulnerable a ataques de fuerza bruta y es más engorroso de administrar.

El login por **llave público-privada** es más cómodo de usar porque permite conectarse sin ingresar la contraseña.

SSH - Llave público-privada

Consiste en un **par de archivos** criptográficos:

1. El correspondiente a la **llave pública** que puede ser distribuido libremente sin preocuparse.
2. El correspondiente a la **llave privada** que debe resguardarse y **nunca exponerlo** a nadie.

SSH - LLave público-privada

Generar el par en la máquina cliente:

$ssh-keygen -t rsa

la llave privada podría a su vez protegerse con una contraseña

En la máquina cliente obtendremos:

/home/$USER/.ssh/id_rsa.pub -> llave **pública**

/home/$USER/.ssh/id_rsa -> llave **privada (RESGUARDAR)**

comandos: ssh-keygen

```
[usuario@localhost ~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/usuario/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/usuario/.ssh/id_rsa.
Your public key has been saved in /home/usuario/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:7qNHM6kHjiQJrKerrTstugFopiOgyJNtTkGiYAiazuY usuario@localhost
```

comandos

```
The key's randomart image is:
+---[RSA 2048]----+
|.                |
|+.               |
|*o .             |
|O.o              |
|*B o    S.       |
|& * o ..=        |
|BE * o +.o       |
|=+* . o.+        |
|XB..  .+..       |
+----[SHA256]-----+
```

comandos

```
[usuario@localhost ~]$ ls -l .ssh
total 28
-rw-------. 1 usuario usuario 1007 abr 14 07:54 authorized_keys
-rw-------. 1 usuario usuario   98 abr 14 07:54 config
-rw-------  1 usuario usuario 1675 abr 23 11:25 id_rsa
-rw-r--r--  1 usuario usuario  414 abr 23 11:25 id_rsa.pub
-rw-r--r--  1 usuario usuario  895 abr 16 20:34 known_hosts
```

SSH - LLave público-privada

Para autenticarse el usuario debe tener **el par** en su computadora **local** y la **llave pública** debe estar copiada en el **servidor remoto** para la cuenta remota en ~/.ssh/authorized_keys

```
[usuario@localhost ~]$ssh-copy-id <usuario_remoto>@<servidor>
```

Hay sistemas en los que el método de **contraseña** directamente **no se utiliza**

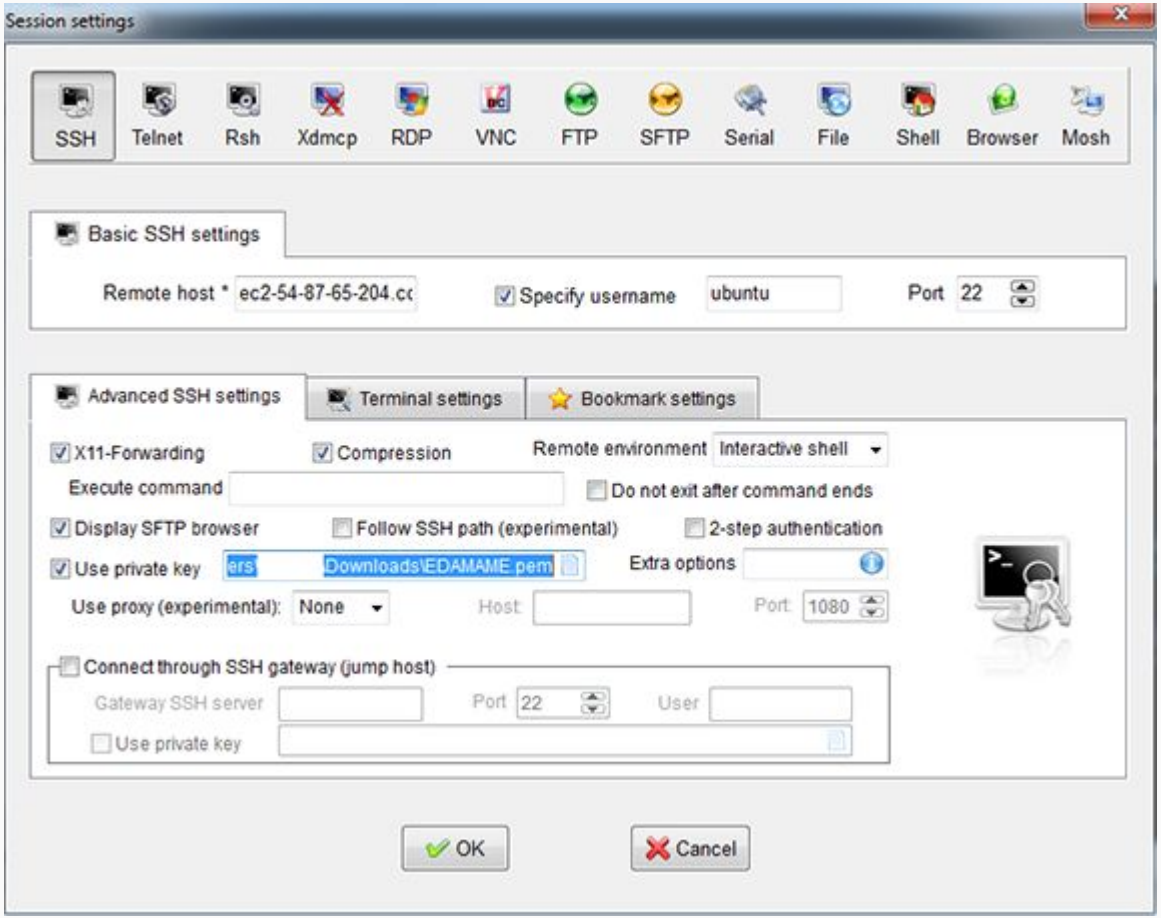
En estos casos “ssh-copy-id” no funciona y se ofrece una **interfaz web** para subir la llave pública o enviarla por **mail al soporte**.

comandos

```
[mmazzini@mendieta ~]$ cat .ssh/authorized_keys
ssh-rsa.AAAB3NzaC1yc2EAAAADAQABAAABAQDPXJ7oVYamyjf2f2CZC5UbFMgCUY4lgpC1l3m09ElNdcAfGEQXZ
FMmx8kGZBSDHn+/7HcpOEGeBGH6UmFJWkezfbgneQRRFfrNSqlLn6sDWZn0cdLOFa7CNByDk6tAy4aQFQEVxn69r
4KNsvooVis5TrSCJH7Yi1BHP5lrSm/xR+EGxr/wxrDh0TqBcoIVeeQDmYZgFBKxBcKywHeQPv1qXB9xBZ+4W4mm1
jote3vo52UB4rv6Tw0npaYFTmiLAnWORbURo/pxx7rENnm6e+7OhUG0pCfQiT+MZ7JU0I9gMfhEQscBHzy7rYZtF
MY3HzjNKwVHtGeVRU8n2aCB+Til marcos@notebook
ssh-rsa.AAAAB3NzaC1yc2EAAAADAQABAAABAQCtxt0y9oenaAlOKIl+1wSr1LtScWWGzh+119YoOUMZ9hsjq2D7
aqGIx4EFH+Y4v86i8YoF7sPmWjwbXVUdg4PDWf9UZAL/dvymaUZ5DsEF+1mKA2nmk1MnUDg6gA+NwuNo8sW94ZJu
Si77qnjdUgwNdrv4l69THFpEvM5xs+MqAvy9dFqf1wFNET3g9tT5ealYLj+oVhuyYdJHQ4N0vG9QGnFiCfByeo4z
k42lcmxzoA4WUz2qTKpeJ3GKfEk5xh4u/tNPlgGcTycwGycVGiGtlkrLh1tL4+6Vt8oajH3g4oZNlC3JeBeIi8iG
wkTqNFdTlX1Fzj7ggJe/TsaEZSyd celu
```

SSH - MoabXterm

MoabXterm implementa un cliente ssh donde la configuración de la conexión se hace mediante la interfaz gráfica.



llave privada para MoabXterm

SSH - Copiando archivos

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.

comandos: scp

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


[usuario@localhost ~]$ scp -r mmazzini@mendieta.ccad.unc.edu.ar:/home/mmazzini/curso .
archivo.txt     100% 0     0.0KB/s   00:00
archivo2        100% 0     0.0KB/s   00:00
archivo3        100% 0     0.0KB/s   00:00
```

Si se va a transferir gran volumen de información conviene comprimir los archivos antes.

Links: Practicar línea de comandos

Practicar comandos básicos resolviendo el misterio

- https://github.com/veltman/clmystery

Desafíos incrementales online para mejorar las habilidades

- https://cmdchallenge.com/

Consola gratuita para usuarios con cuenta google

- https://console.cloud.google.com/cloudshell

Consola Bash nativa para Windows 10

- https://www.howtogeek.com/249966/how-to-install-and-use-the-linux-bash-shell-on-windows-10

Sistema de Colas: SLURM

SLURM

El cluster es un **recurso de cómputo compartido** y por lo tanto no puede atender el uso exclusivo de los procesadores instantáneamente como ocurre en una computadora personal.

Cada usuario **debe describir** qué tareas quiere realizar y el sistema de colas decidirá **cuándo y dónde** realizarlas en base a ésta descripción y al resto de las tareas.

SLURM

El sistema de colas también se encarga de manejar la **salida a pantalla** reenviandola a la terminal del usuario o a un archivo.

El cluster está configurado para que las tareas que se ejecutan en **los nodos** tengan acceso a los archivos del **home del usuario** a través de la red **infiniband** transparentemente (protocolo nfs).

Tener en cuenta que el almacenamiento junto a su red de acceso son también recursos compartidos.

comandos: srun, salloc

```
[mmazzini@mendieta ~]$ srun echo hola
srun: job 73876 queued and waiting for resources
srun: job 73876 has been allocated resources
hola
[mmazzini@mendieta ~]$ salloc
salloc: Pending job allocation 73877
salloc: job 73877 queued and waiting for resources
ssalloc: job 73877 has been allocated resources
salloc: Granted job allocation 73877
[mmazzini@mendieta ~]$ srun hostname
mendieta08.mendieta.ccad.unc.edu.ar
[mmazzini@mendieta ~]$ exit
exit
salloc: Relinquishing job allocation 73877
[mmazzini@mendieta ~]$
```

SLURM

La mayoría de la veces tendremos trabajos no-interactivos que además contarán con más de un comando. Para esto creamos un **script** que contienen la secuencia de comandos y los enviamos con **sbatch**.

```
[mmazzini@mendieta ~]$ cat cinco.sh
#!/bin/bash
date
sleep 300
hostname
[mmazzini@mendieta ~]$ sbatch minuto.sh
Submitted batch job 73878
```

SLURM

- Mientras más información le demos al sistema de colas más rápido se encolará nuestro job.
- Debemos usar comandos de monitoreo para conocer la carga del cluster y cómo van nuestros jobs, no olvidemos que es un recurso compartido.
- Es probable que cometamos errores, en estos casos es importante cancelar los jobs para no sobrecargar el sistema innecesariamente.

SLURM

Para dar más información de nuestro trabajo podemos agregar **opciones** a la línea de comandos o poner un **encabezado** en nuestro script para convertirlo en un **script de lanzamiento**.

Veremos cómo informar una estimación sobre cuánto tardará la ejecución del job.

SLURM

```
[mmazzini@mendieta ~]$ sbatch -t 00:06 cinco.sh
```

```
[mmazzini@mendieta ~]$ cat cinco.sh
#!/bin/bash
###SBATCH --time=dd-hh:mm  <-comentario
#SBATCH --time=00:06

date
sleep 300
hostname
[mmazzini@mendieta ~]$ sbatch cinco.sh
```

SLURM

Para tener un control sobre cómo van nuestros jobs y cual es el estado del cluster se usarán comandos de monitoreo.

Comandos: squeue

```
[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  sbatch 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   sbatch    R         0:02  1   1 (null mendieta08

[mmazzini@mendieta ~]$ cat slurm-73878.out
Sun Sep 30 19:17:03 ART 2018
mendieta08.mendieta.ccad.unc.edu.ar
```

Comandos: scancel, scontrol

Si algo va mal podemos ver la razón y cancelar el job

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

```
[mmazzini@mendieta ~]$ scontrol show job 73878
JobId=73890 JobName=cinco.sh
   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/minuto.sh
   ...
```

Programas y Particiones

Programas / Aplicaciones

- Ya están instaladas (uso extendido)
- Solicitar instalarlas a soporte (conjunto de usuarios)
- Instalarla en el home (usar conda/miniconda si es python)

El mecanismo para usar las aplicaciones o librerías instaladas por soporte es el de módulos de entorno o **environmental modules.**

Este mecanismo permite tener **distintas versiones** del software sin que haya conflicto, dejando en manos del usuario la elección de cuál utilizará.

comandos: module

```
[mmazzini@mendieta ~]$ module list
No Modulefiles Currently Loaded.
[mmazzini@mendieta ~]$ module avail
------------------------------------ /opt/modules/aplicaciones -------------------------------------
R/3.4.1    espresso/5.0.3 gromacs/5.0.2     octave/4.2.1          parallel/20170322   samtools/1.6

------------------------------------ /opt/modules/compiladores -------------------------------------
cuda/8.0(default)    gcc/5(default)     intel/2016(default) jdk/8u121(default)  xeonphi/2016

------------------------------------ /opt/modules/herramientas -------------------------------------
autoconf/2.69(default) binutils/2.27(default) cmake/3.7.1(default)    gdb/7.11(default)
[mmazzini@mendieta ~]$ R --help
-bash: R: no se encontró la orden
[mmazzini@mendieta ~]$ module load R
[mmazzini@mendieta ~]$ R --help

Usage: R [options] [< infile] [> outfile]
   or: R CMD command [arguments]
```

Paralelización

Los programas y librerías instalados están **optimizados** para mejorar el rendimiento con tecnologías de paralelización. Quienes ya saben o planean aprender a programar pueden también utilizarlas al compilar sus propios programas.

- pthreads/OpenMP
- MPI
- GPU/CUDA - PHI

Particiones

SLURM permite agrupar conjuntos de nodos y establecer parámetros por defecto para cada partición.

En el CCAD optamos por agrupar en base a la tecnología de paralelización.

Al enviar un trabajo el usuario debe indicar a qué partición lo enviará:

```
#SBATCH --partition=multi
```

En nuestros clusters contamos con las siguientes particiones:

SLURM: Particiones en el CCAD

- **DEBUG**: Máximo 2 min
- **MONO**: trabajos de 1 nodo, hasta 8 cores por nodo, hasta 7 días de ejecución (pthreads/OpenMP).
- **MULTI**: trabajos de 2 a 8 nodos, hasta 20 cores por nodo, hasta 4 días de ejecución (MPI/MPI+OpenMP).
- **GPU**: trabajos de 1 nodo, hasta 8 cores por nodo, hasta 7 días de ejecución. Acceso a las GPUs del cluster (CUDA).
- **PHI**: trabajos de 1 a 13 nodos, hasta 4 días de ejecución. Acceso a las placas Xeon PHI (compilación específica)

Paralelización

Además de indicar al sistema de colas en qué partición vamos a ejecutar debemos especificar **cuántos procesadores y nodos** solicitamos.

Para correr un job MPI se utiliza la interfaz srun compatible con mpirun

```
srun comando_compilado_para_mpi
```

En caso que combinemos MPI+OpenMP también hay que tener en cuenta cómo se **distribuyen** las tareas.

Cantidad de tareas

- Para indicar cuantas tareas paralelas vamos a correr:

```
#SBATCH --ntasks=P
```

- Para indicar **más precisamente** la distribución deseada (P=N*PPN):

```
#SBATCH --nodes=N
#SBATCH --ntasks-per-node=PPN
```

Cantidad de tareas / GPU

- Para procesos OpenMP, se reserva un core por tarea (opcional)

```
#SBATCH --cpus-per-task=H
```

- Para solicitar GPU (hasta 3 **por nodo**)

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

```
#!/bin/bash

#SBATCH --job-name=espresso
#SBATCH --partition=multi
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=20
#SBATCH --cpus-per-task=1
#SBATCH --gres=gpu:0
#SBATCH --time 4-0:00

#-------------------------------------------------
# No tocar
. /etc/profile

# Configurar OpenMP y otras bibliotecas que usan threads

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
export MKL_NUM_THREADS=$SLURM_CPUS_PER_TASK

# Cargar los módulos para la tarea
module load espresso/5.4.0

# Lanzar el programa
srun pw.x -nk 2 -inp input > output
```

Ejemplo de script de submit completo

Soporte CCAD

wiki.ccad.unc.edu.ar

mailto: soporte@ccad.unc.edu.ar