Preguntas

1. **Investigar y contestar las siguientes preguntas. ¿Que es un usuario root en Linux?**

En Linux el usuario root es aquel que tiene todos los permisos en el sistema operativo, es decir, es el súper administrador. Puede acceder a cualquier archivo y también ejecutar cualquier comando, incluidos los que nunca deberías ejecutar.

Debido al gran poder y peligro que tiene el usuario root, en Ubuntu y otras distribuciones basadas en Ubuntu viene deshabilitado por defecto. Si quisiéramos instalar cualquier programa o hacer algo que requiere más permisos de los que tiene nuestro usuario podríamos utilizar el comando ***sudo***. Si introducimos ***sudo*** nos pedirá nuestra contraseña. Para hacer el proceso más cómodo, podemos hacer login directamente como root con el comando ***sudo -i***.

2. **¿Por qué ubuntu no me deja establecer la contraseña durante la instalación?**

El instalador le solicita que cree una contraseña al escribir el nombre de cuenta de usuario y si no ingresa una contraseña y solo ingresa el nombre de la cuenta de usuario, termina diciendo que cree una contraseña.

"Debe conocer su contraseña para instalar el software en Ubuntu. Esta es una característica de seguridad de Ubuntu y otras distribuciones de Linux".

Esto evita que personas no autorizadas alteren la configuración del sistema, también es una "red de seguridad" para que confirmes que realmente quieres hacer cambios en la configuración.

3. ¿Cuáles son los procesos típicos de Linux? ¿Cómo identificarlos?

COMANDO ps:

El comando ps devuelve el estado de los procesos. Su framework es el directorio /proc; lo que significa que lee directamente la información de archivos que se encuentran en este directorio.

COMANDO pstree:

Muestra los procesos en forma de árbol, pstree --help te da las opciones más comunes.
Es recomendable usarlo con la opción -A y -G para que te un árbol con líneas con líneas estilo ASCII y de terminal VT100 respectivamente, puedes añadir también -u para mostrar entre paréntesis al usuario propietario del proceso:

#> pstree -AGu

COMANDO kill:

El comando kill, que literalmente quiere decir matar, sirve no solo para matar o terminar procesos sino principalmente para enviar señales (signals) a los procesos. La señal por default (cuando no se indica ninguna es terminar o matar el proceso), y la sintaxis es kill PID, siendo PID el número de ID del proceso. Así por ejemplo, es posible enviar una señal de STOP al proceso y se detendrá su ejecución, después cuando se quiera mandar una señal de CONTinuar y el proceso continuará desde donde se quedó.

#> kill -l

La lista que devuelve el comando contiene todas las posibles señales que pueden mandarse a un proceso y estas pueden ser invocadas a través del número de la señal o de su código, por ejemplo:

#> kill -9 11428 (termina, mata un proceso completamente)

#> kill -SIGKILL 11428 (Lo mismo que lo anterior)

Las señales más comunes son la 19 y 20 que detienen momentáneamente la ejecución de un proceso o programa, 18 la continua, 1 qué es la señal de hang up que obliga al proceso a releer sus archivos de configuración estando en ejecución y 9 que termina rotundamente un proceso.

COMANDO killall:

El comando killall, que funciona de manera similar a kill, pero con la diferencia de que en vez de pasarle un PID se indica el nombre del programa, lo que afectará a todos los procesos que tengan ese nombre. Así por ejemplo si se tienen varias instancias ejecutándose del proxy server squid, con killall squid eliminará todos los procesos que se estén ejecutando con el nombre 'squid'

#> killall -l (lista de posibles señales)

#> killall -HUP httpd (manda una señal de "colgar", detenerse
releer sus archivos de configuración y
reiniciar)

#> killall -KILL -i squid (manda señal de matar a todos los
procesos squid pero pide confirmación)

COMANDO nice:

Permite cambiar la prioridad de un proceso. Por defecto, todos los procesos tienen una prioridad igual ante el CPU que es de 0. Con nice es posible iniciar un programa (proceso) con la prioridad modificada, más alta o más baja según se requiera. Las prioridades van de -20 (la más alta) a 19 la más baja. Solo root puede establecer prioridades negativas que son más altas. Con la opción ps -l es posible observar la columna NI que muestra este valor.

#> nice (sin argumentos, devuelve la prioridad por defecto)

#> nice -n -5 comando (inicia comando con una prioridad de -5, lo que le da más tiempo de cpu)

COMMANDO renice:

Así como nice establece la prioridad de un proceso cuando se inicia su ejecución, renice permite alterarla en tiempo real, sin necesidad de detener el proceso.

#> nice -n -5 yes (se ejecuta el programa 'yes' con prioridad -5)

- Dejemos ejecutando yes y en otra terminal consultemos ps

  #> ps -el

- Ahora cambiemos su prioridad:

  #> renice 7 12826 (ojo:usar el id del proceso)

- Vemos que la prioridad antigua era -5, y la nueva prioridad que pedimos es 7, volvemos a checar la prioridad

  #> ps -el

COMMANDO nohup y &:

Cuando se trata ejecutar procesos en background (segundo plano) se utiliza el comando nohup o el operador . Aunque realizan una función similar, no son lo mismo.
Si se desea liberar la terminal de un programa que se espera durará un tiempo considerable ejecutándose, entonces se usa . Esto funciona mejor cuando el resultado del proceso no es necesario mandarlo a la salida estándar (stdin), como por ejemplo cuando se ejecuta un respaldo o se abre un programa Xwindow desde la consola o terminal. Para lograr esto basta con escribir el comando en cuestión y agregar al final el símbolo &.

$> nohup yes > /dev/null &

$> nohup czf respaldo /documentos/* > /dev/null/

$> nohup konqueror

Así se evita que el proceso se "cuelgue" al cerrar la consola.

- Otra forma de usar nohup es para evitar que un programa que ejecutó el usuario no se corte o se termine abruptamente si este hace logout:

  # nohup miprograma &
  # exit

COMMANDO jobs:

Si por ejemplo, se tiene acceso a una única consola o terminal, y se tienen que ejecutar varios comandos que se ejecutarán por largo tiempo, se pueden entonces como ya se vió previamente con nohup y el operador '&' mandarlos a segundo plano o background con el objeto de liberar la terminal y continuar trabajando.

Pero si solo se está en una terminal esto puede ser difícil de controlar, y para esos casos tenemos el comando jobs que lista los procesos actuales en ejecución:

#> yes > /dev/null &
[1] 26837

#> ls -laR > archivos.txt &
[2] 26854

#> jobs
[1]- Running yes >/dev/null &
[2]+ Running ls --color=tty -laR / >archivos.txt &

COMMANDO top:

Es una utilería muy útil para el monitoreo en tiempo real del estado de los procesos y de otras variantes del sistema; se ejecuta desde la línea de comandos, es interactivo y por defecto se actualiza cada 3 segundos.

```
$> top
top - 13:07:30 up 8 days,  6:44,  4 users,  load average: 0.11, 0.08, 0.08
Tasks: 133 total,   1 running, 131 sleeping,   0 stopped,   1 zombie
Cpu(s):  0.0%us,  0.2%sy,  0.0%ni, 99.7%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:    497356k total,   472352k used,    25004k free,    21500k buffers
Swap:  1156640k total,   257088k used,   899552k free,    60420k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
26156 sergon    15   0  2160 1016  784 R    1  0.2   0:00.93 top
    1 root      15   0  2012  616  584 S    0  0.1   0:00.98 init
    2 root      RT   0     0    0    0 S    0  0.0   0:00.29 migration/0
    3 root      34  19     0    0    0 S    0  0.0   0:00.00 ksoftirqd/0
    4 root      RT   0     0    0    0 S    0  0.0   0:00.00 watchdog/0
    5 root      RT   0     0    0    0 S    0  0.0   0:00.38 migration/1
```

Estando dentro de la aplicación, presionando 'h' muestra una ayuda de los posibles comandos que permiten configurar top, por ejemplo, al presionar 's' pregunta por el tiempo en segundos de actualización, etc.

COMMANDO fork:

Es un mecanismo de control de procesos e hilos. Dos conceptos relacionados, pero con sutiles diferencias.
Partiendo de la idea posible de "ejecutar varias cosas a la vez" surgen los hilos o threads, pero su aspecto más relevante es la ejecución fuera del contexto de una terminal o una ventana, evitando en todo momento el cuello de botella con la interfaz de usuario y liberando éstos recursos para la aplicación en primer plano.

Si queremos que nuestro programa empiece a ejecutar varias cosas "a la vez", tenemos dos opciones.
Podemos crear un nuevo proceso o crear un nuevo hilo de ejecución (un thread).

Procesos y su relación con fork
En un sistema Linux, (multihilo), se pueden estar ejecutando distintas acciones a la par, y cada acción es un
proceso que consta de uno o más hilos, memoria de trabajo compartida por todos los hilos e información de
planificación. Cada hilo consta de instrucciones y estado de ejecución.

Cuando ejecutamos un comando en el shell, sus instrucciones se copian en algún sitio de la memoria RAM del
sistema para ser ejecutadas. Cuando las instrucciones ya cumplieron su cometido, el programa es borrado de la
memoria del sistema, dejándola libre para que más programas se puedan ejecutar a la vez. Por tanto cada uno de estos programas son procesos.

Los procesos son creados y destruidos por el sistema operativo, pero lo hace a petición de otros procesos. El
mecanismo por el cual un proceso crea otro proceso se denomina bifurcación (fork). Los nuevos procesos son
independientes y no comparten memoria (es decir, información) con el proceso que los ha creado.
En definitiva, es posible crear tanto hilos como procesos. La diferencia estriba en que un proceso solamente puede
crear hilos para sí mismo y en que dichos hilos comparten toda la memoria reservada para el proceso.

COMMANDO Hilos:

Los hilos son similares a los procesos ya que ambos representan una secuencia simple de instrucciones
ejecutada en paralelo con otras secuencias. Los hilos son una forma de dividir un programa en dos o más
tareas que corren simultáneamente, compitiendo, en algunos casos, por la CPU.

La diferencia más significativa entre los procesos y los hilos, es que los primeros son típicamente independientes,
llevan bastante información de estados, e interactúan sólo a través de mecanismos de comunicación dados por el sistema. Por otra parte, los hilos generalmente comparten la memoria, es decir, acceden a las mismas variables
globales o dinámicas, por lo que no necesitan costosos mecanismos de comunicación para sincronizarse. Por
ejemplo un hilo podría encargarse de la interfaz gráfica (iconos, botones, ventanas), mientras que otro hace una
larga operación internamente. De esta manera el programa responde más ágilmente a la interacción con el
usuario.

En sistemas operativos que proveen facilidades para los hilos, es más rápido cambiar de un hilo a otro dentro del
mismo proceso, que cambiar de un proceso a otro.
Es posible que los hilos requieran de operaciones atómicas para impedir que los datos comunes sean cambiados
o leídos mientras estén siendo modificados. El descuido de esto puede generar estancamiento.

La tabla siguiente resume algunas diferencias entre procesos e hilos, y la figura 1 muestra una representación de
los conceptos proceso e hilo. Nótese cómo en la figura 1 se puede apreciar que los procesos son entidades
independientes, mientras que los hilos son entidades relacionadas por la sección de datos en el interior del
proceso que los contiene.

RELACION PROCESOS-HILOS

Creación de procesos: fork y clone
A la hora de crear procesos linux provee de dos funciones para dicho cometido, la función clone() y la función
fork(). Ambas crean un nuevo proceso a partir del proceso padre pero de una manera distinta.
Cuando utilizamos la llamada al sistema fork, el proceso hijo creado es una copia exacta del padre (salvo por el
PID y la memoria que ocupa). Al proceso hijo se le facilita una copia de las variables del proceso padre y de
los descriptores de archivo. Es importante destacar que las variables del proceso hijo son una copia de las del
padre (no se refieren físicamente a la misma variable), por lo que modificar una variable en uno de los procesos
no se refleja en el otro.

La llamada al sistema clone es mucho más genérica y flexible que el fork, ya que nos permite definir qué van a
compartir los procesos padre e hijo. La tabla 2 resume las diferencias entre las llamadas al sistema fork y clone.

Las llamadas al sistema fork y clone tienen la misma funcionalidad, pero distintas características:

COMANDO **fork:**

En el momento de la llamada a fork el proceso hijo:

- Es una copia exacta del padre excepto el PID.
- Tiene las mismas variables y archivos abiertos.
- Las variables son independientes (padre e hijo tienen distintas memorias).
- Los archivos son compartidos (heredan el descriptor).

COMANDO **clone:**

Permite especificar qué queremos que compartan padre e hijo

- Espacio de direccionamiento
- Información de control del sistema de archivos (file system)
- Descriptores de archivos abiertos.
- gestores de señales o PID.

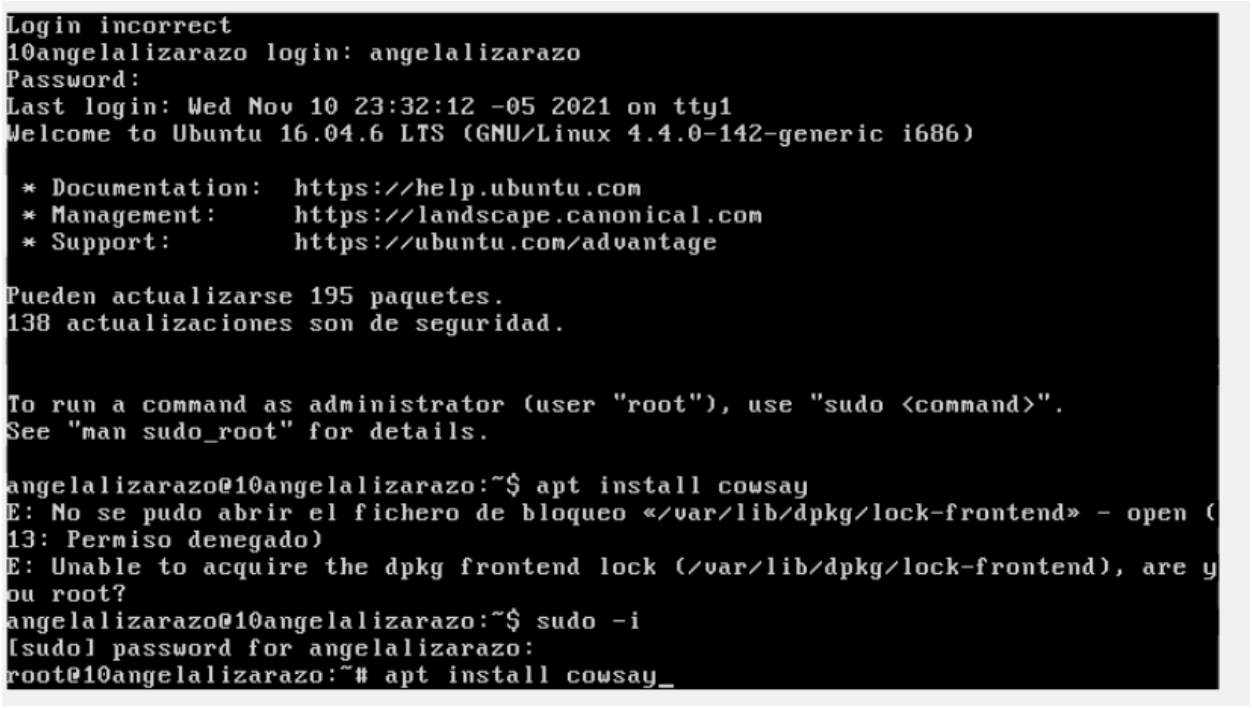
4. Investigar y establecer una contraseña para el usuario root.

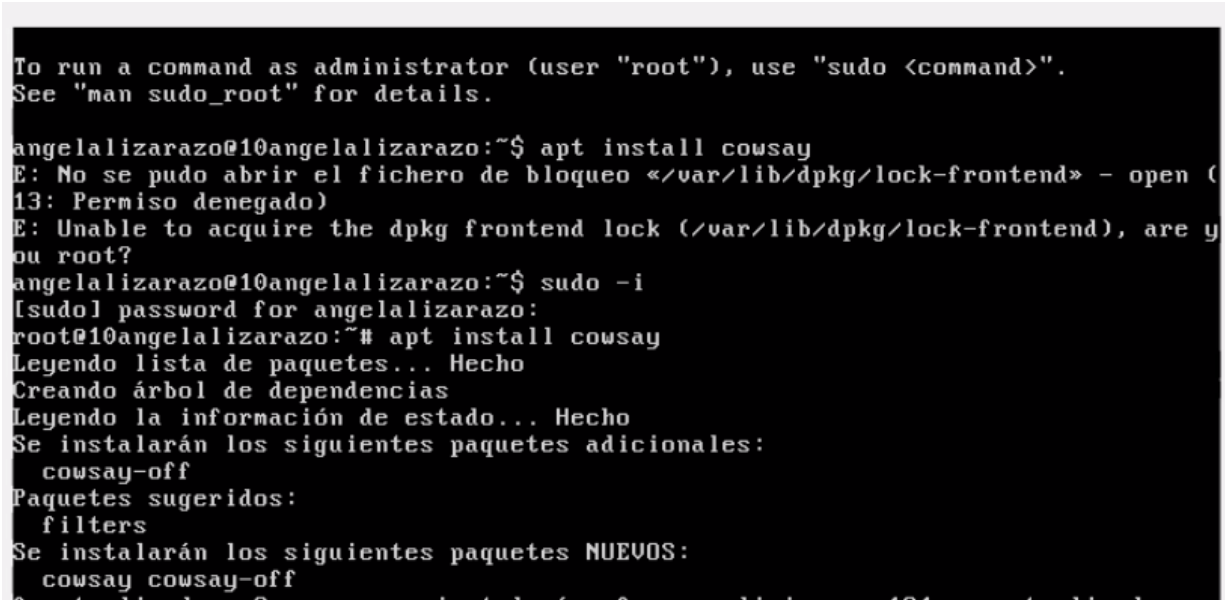
Para habilitar root simplemente tenemos que asignarle una contraseña con el siguiente comando **(no olvidar la contraseña ya que de olvidarse, se perderá completamente el acceso).**

```
sudo passwd root
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Opcional:

- Escribir en la terminal el comando apt install cowsay.
- Escribir en la terminal el comando cowsay “ Hola mundo “.
- Escribir en la terminal el comando sudo apt install fortune
- Escribir en la terminal fortune.
- fortune | cowsay






Preparando para desempaquetar .../cowsay_3.03+dfsg1-15_all.deb ...
Desempaquetando cowsay (3.03+dfsg1-15) ...
Seleccionando el paquete cowsay-off previamente no seleccionado.
Preparando para desempaquetar .../cowsay-off_3.03+dfsg1-15_all.deb ...
Desempaquetando cowsay-off (3.03+dfsg1-15) ...
Procesando disparadores para man-db (2.7.5-1) ...
Configurando cowsay (3.03+dfsg1-15) ...
Configurando cowsay-off (3.03+dfsg1-15) ...
root@10angelalizarazo:~# cowsay "Hola mundo"
La orden «cowsay» está disponible en «/usr/games/cowsay»
La orden no se pudo encontrar porque «/usr/games» no se ha incluido en la variab
le de entorno PATH.
cowsay: no se encontró la orden
root@10angelalizarazo:~# exit
logout
angelalizarazo@10angelalizarazo:~$ cowsay "Hola Mundo"
< Hola Mundo >
angelalizarazo@10angelalizarazo:~$

```
Procesando disparadores para libc-bin (2.23-0ubuntu11) ...
angelalizarazo@10angelalizarazo:~$ fortune
Like an expensive sports car, fine-tuned and well-built, Portia was sleek,
shapely, and gorgeous, her red jumpsuit moulding her body, which was as warm
as seatcovers in July, her hair as dark as new tires, her eyes flashing like
bright hubcaps, and her lips as dewy as the beads of fresh rain on the hood;
she was a woman driven -- fueled by a single accelerant -- and she needed a
man, a man who wouldn't shift from his views, a man to steer her along the
right road: a man like Alf Romeo.
                -- Rachel Sheeley, winner

The hair ball blocking the drain of the shower reminded Laura she would never
see her little dog Pritzi again.
                -- Claudia Fields, runner-up

It could have been an organically based disturbance of the brain -- perhaps a
tumor or a metabolic deficiency -- but after a thorough neurological exam it
was determined that Byron was simply a jerk.
                -- Jeff Jahnke, runner-up

Winners in the 7th Annual Bulwer-Lytton Bad Writing Contest.  The contest is
named after the author of the immortal lines:  "It was a dark and stormy
night."  The object of the contest is to write the opening sentence of the
worst possible novel.
angelalizarazo@10angelalizarazo:~$ _
```

```
The hair ball blocking the drain of the shower reminded Laura she would never
see her little dog Pritzi again.
                -- Claudia Fields, runner-up

It could have been an organically based disturbance of the brain -- perhaps a
tumor or a metabolic deficiency -- but after a thorough neurological exam it
was determined that Byron was simply a jerk.
                -- Jeff Jahnke, runner-up

Winners in the 7th Annual Bulwer-Lytton Bad Writing Contest.  The contest is
named after the author of the immortal lines:  "It was a dark and stormy
night."  The object of the contest is to write the opening sentence of the
worst possible novel.
angelalizarazo@10angelalizarazo:~$ fortune | cowsay
 _______________________________________
/ You have a reputation for being       \
| thoroughly reliable and trustworthy. A |
\ pity that it's totally undeserved.     /
 ---------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
angelalizarazo@10angelalizarazo:~$
```