Proyecto de BASH: Información básica de las sesiones de los procesos.

Es hora de aprender a manejar los conceptos vistos para crear scripts por nosotros mismos que automaticen las tareas más comunes y que además nos permita profundizar en aspectos importantes del sistema operativo.

Es muy importante que entiendas que este proyecto es estrictamente individual y debe ser realizado sin ayuda de sistemas de Inteligencia Artificial como Copilot, Chat GPT o cualquier otro.

Este ejercicio está pensado para que te enfrentes a los problemas inherentes de utilizar la shell de un sistema operativo para administrar y automatizar determinadas tareas. Esto solo es posible si realizas la práctica de forma individual. Los proyectos copiados recibirán un 0 sin menoscabo de otras posibles acciones disciplinarias. Tampoco se permiten las prácticas realizadas en grupo.

Importante: no expongas tu código en un repositorio público.

Por los mismos motivos no se permite el uso de asistentes de inteligencia artificial de cualquier tipo.

1. Recomendaciones generales.

Cumplir o no estas recomendaciones tiene influencia en la calificación del trabajo, como se reflejará en la rúbrica de la tarea.

- Recuerda facilitar la lectura de tu código:
    - Usa comentarios en los elementos más complejos del programa, para que te sea más sencillo entenderlo cuando haya pasado un tiempo y no te acuerdes.
    - Usa funciones para compartimentar el programa, así como variables y constantes (variables en mayúsculas) que hagan tu código más legible.
    - Incluye código para procesar la línea de comandos.
- Incluir la ayuda del comando:
    - Se debe indicar el error y mostrar ayuda sobre el uso si el usuario emplea una opción no soportada
- Maneja adecuadamente los errores:

    - En caso de error muestra un mensaje y sal con código de salida distinto de 0. Recuerda la función `error_exit` de ejercicios anteriores y, si lo crees conveniente, reutiliza o haz la tuya propia.
    - Trata como un error que el usuario emplee opciones no soportadas.
    - Haz lo mismo con las otras posibles condiciones de error que se te ocurran, ¿has probado a invocar tu programa con opciones absurdas a ver si lo haces fallar?
- Ojo con la sustitución de variables y las comillas. En caso de problemas, piensa en cómo quedarían las sentencias si las variables no valieran nada, ¿tendría sentido para BASH el comando a ejecutar?
- Cuidado al pasar argumentos a las funciones siendo estos argumentos variables. Realiza el paso de argumentos usando comillas alrededor de la expansión de variables. Por ejemplo, si haces esta llamada:
  var1=
  var2=”Hola, mundo”
  Mifuncion $var1 $var2

  Al entrar en Mifuncion el parámetro 1 será “Hola,” y el parámetro 2 será “mundo”, dada la forma del comando que queda después de la expansión. Sin embargo si haces la llamada así:

  Mifuncion “$var1” “$var2”

  Al entrar en Mifuncion, el parámetro 1 es la cadena vacía y el parámetro 2 será “Hola, mundo”, ya que las comillas también sirven para delimitar los argumentos de los comandos y funciones.

2. Tareas a realizar.

Tendrás que desarrollar el script **infosession.sh** para la shell Bash que incluya las siguientes funcionalidades. Utiliza un sistema Linux. Ten en cuenta que el ejercicio se corregirá en los ordenadores del Centro de Cálculo de la ESIT.

Ten en cuenta que salvo que se diga expresamente lo contrario, todas las opciones son combinables en cualquier orden.

1. Funcionamiento básico

En este apartado consideraremos la siguiente sintaxis admisible, que irá cambiando a medida que avanzamos:

```
infosession.sh  [-h] [-z] [-u user ]
```

a) Si no se especifica ninguna de las opciones se deberán mostrar la información de los procesos cuyo usuario efectivo sea el usuario de bash, incluyendo los siguientes campos del proceso:

1. Identificador de sesión (sid)
2. Identificador del grupo de procesos al que pertenece (pgid)
3. Identificador de proceso (pid)
4. Nombre de usuario efectivo del proceso (user)
5. Identificador de la terminal controladora o ? si no tiene ninguna (tty)
6. Porcentaje de memoria consumida por el proceso (%mem)
7. Comando que lanzó el proceso (cmd)

En esta tabla **no** se deben mostrar los procesos cuya sesión tenga identificador 0. Además, la información debe mostrarse ordenada por nombre de usuario, en una tabla con encabezado con un formato cuidado.

Para implementar esta funcionalidad, y de cara al resto del proyecto, te recomiendo que primero realices una función que produzca en la salida estándar una tabla con formato simple (campos separados por un espacio, por ejemplo) sin encabezado. Trata que el tipo de ordenamiento pueda ser un parámetro de la función puesto que en varias partes del proyecto se precisan ordenamientos diferentes.

Importante: La información necesaria se puede obtener con el comando ps. C**onstruye tu script de modo que solamente se llame al comando ps una vez.** Esta llamada a ps te servirá para obtener una "fotografía" del estado de los procesos en el momento en que haces la llamada. En esta llamada tendrás que recopilar toda la información que necesites. La razón es que el estado de los procesos en el sistema es dinámico y por tanto si intentas obtener la información necesaria a trozos, en diferentes momentos, no tendrás una imagen coherente del estofado de los procesos.

b) La opción -h debe simplemente mostrar la ayuda del comando y terminar con estado 0.

c) Si se utiliza la opción -z, la tabla mostrará también los procesos cuyo identificador de sesión sea 0.

d) La opción -u deberá ir acompañada de un nombre de usuario. Si se utiliza, se mostrarán los procesos cuyo usuario efectivo sea el especificado.

Observa que aquí y en el resto del proyecto todas las opciones se pueden combinar y especificar en cualquier orden. Por ejemplo, si se combina -z con -u user, se mostrarán los procesos cuyo identificador de sesión es el 0 solo si su usuario efectivo es el especificado. Si se combina -h con cualquier opción, se mostrará la ayuda y no se hará nada más.

e) Si se introducen opciones incorrectas o mal formadas (por ejemplo, se especifica -u y a continuación no hay un nombre de usuario, el script debe detenerse, dando cuenta del error con un mensaje y con estado de salida diferente de 0.

2. Selección por múltiples usuarios y por procesos buscados con lsof

Ampliaremos el funcionamiento básico de modo que ahora las opciones admisibles son:

```
infosession.sh  [-h] [-z] [-u user1 ... ] [ -d dir ] [-t ]
```

a) Ahora la opción -u admite una lista de longitud arbitraria de nombres de usuario, de modo que los procesos se seleccionan de forma que su usuario efectivo sea cualquiera de los usuarios en la lista.
b) La opción -d toma como parámetro obligatorio una ruta de directorio. Con esta opción el script debe mostrar solo procesos que tengan abiertos archivos en el directorio especificado por dir. Para ello usaremos el comando lsof.

   El comando lsof permite obtener los archivos abiertos por un proceso. Es un comando muy importante para la administración de sistemas y tiene infinidad de opciones. Aquí lo vamos a utilizar en una forma básica:

   ```
   lsof +d ruta-de-directorio
   ```

   que nos permite obtener información de los procesos que han abierto alguno de los archivos dentro del directorio especificado (sin entrar en los directorios subsiguientes). Por ejemplo

   lsof +d /opt/docencia/ssoo2425

   ```
   COMMAND     PID  USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
   bash       3054 nacho  cwd  DIR  8,21     4096 50479834 /opt/docencia/ssoo2425
   vim       14290 nacho  cwd  DIR  8,21     4096 50479834 /opt/docencia/ssoo2425
   vim       14290 nacho    4u  REG  8,21    28672 50479836 opt/docencia/ssoo2425/.infoproc.sh.swp
   bash     105276 nacho  cwd  DIR  8,21     4096 50479834 /opt/docencia/ssoo2425
   lsof     112577 nacho  cwd  DIR  8,21     4096 50479834 /opt/docencia/ssoo2425
   lsof     112578 nacho  cwd  DIR  8,21     4096 50479834 /opt/docencia/ssoo2425
   ```

   La primera columna es el nombre del comando del proceso que tiene abierto el archivo, mientras que la segunda columna es el pid del mismo. En la última columna está la ruta al archivo abierto.

   La opción -d se puede combinar con cualquiera de las demás opciones. Por ejemplo, si también se utiliza la opción -u, los procesos en la lista deberán tener como usuario

efectivo alguno de los señalados por -u y tener abierto alguno de los archivos en el directorio dado por -d.

Importante: Recuerda que el comando ps utiliza un sistema aditivo para combinar las opciones. Por ejemplo, si escribes
ps -e -u luis
Se seleccionarán todos los procesos del sistema (opción -e=, no solo los procesos del usuario luis (opción -u luis).
El comportamiento que se pide en este script es diferente: el resultado de la selección de los procesos debe cumplir todas las condiciones impuestas al mismo tiempo.

c) Si añades la opción -t los procesos seleccionados tendrán que tener forzosamente una terminal controladora asociada.

3. Información por sesión y ordenamientos

Llegamos al funcionamiento final del script. El formato final es:

```
infosession.sh [-h] [-e ] [-z] [-u user1 ... ] [ -d dir ] [-t ] [-sm] [-r]
infosession.sh [-h] [-z] [-u user1 ... ] [ -d dir ] [-t ] [-sg] [-r]
```

a) El comportamiento descrito hasta ahora, es decir, mostrar la lista de procesos, solo se activará si se establece la opción -e. Cuando esta opción no se utiliza, el script pasa al modo “tabla de sesiones”, donde la tabla incluirá los siguientes campos:

1. Identificador de sesión
2. Total de grupos de procesos diferentes de esa sesión con algún proceso seleccionado por el resto de opciones.
3. Total del porcentaje de memoria consumida por todos los procesos de la sesión seleccionados en esa sesión.
4. Identificador de proceso del proceso líder de la sesión.
5. Usuario efectivo del proceso líder de la sesión.
6. Terminal controladora de la sesión (si la tuviera).
7. Comando del proceso líder de la sesión.

Es posible que el proceso líder de la sesión no cumpla con los criterios de selección establecidos en las opciones. En ese caso, en los campos relativos al proceso líder de sesión se pondrá el símbolo ?

La realización de esta parte del proyecto es más sencilla si partes de la tabla de datos en crudo mencionada en el apartado 1.a del guión con las selecciones aplicadas. Dependiendo de cómo vayas a resolver los totales te puede interesar un ordenamiento concreto, por ejemplo por sesión en primer lugar y por grupo de procesos en segundo lugar.

Recuerda que el pid del proceso líder de la sesión coincide con el identificador de sesión (sid). Por otra parte, el identificador de grupo (pgid) coincide con el pid del proceso líder del grupo. El proceso líder de la sesión será líder de uno de los grupos de procesos que componen la sesión.

b) El ordenamiento por defecto de esta tabla será por nombre de usuario. Sin embargo, si se usa la opción -sm, el ordenamiento será por el total de memoria consumida por los procesos de la sesión (de menor a mayor). Si se usa -sm con la opción -e, el ordenamiento será por el porcentaje de memoria consumida por el proceso.

c) La opción -sg ordenará la tabla "resumen" por el número de grupos de procesos en cada sesión (de menos a más). La opción -sg es incompatible tanto con la opción -e como con la opción -sm. Si se usan conjuntamente se detectará en el script y se terminará con un error.

d) La opción -r invertirá cualquiera de los ordenamientos, incluido el ordenamiento por nombre de usuario.

Evaluación:
Nota: las fechas que se mencionan aquí podrían sufrir cambios si se produjeran situaciones no previstas en la planificación.

Entregas parciales

Está prevista la evaluación de dos entregas parciales del proyecto antes de la entrega final. En esas entregas parciales se pedirá en la hora de prácticas realizar una modificación sobre el código, y el resultado de la evaluación dependerá en gran medida de esta modificación. Las fechas previstas para estas entregas parciales son las del 28 de octubre al 1 de noviembre (primera entrega parcial) y la del 4 al 8 de noviembre (segunda entrega parcial). El contenido a tener terminado en cada entrega parcial se describirá en la actividad que se abrirá para cada tarea en el aula virtual. Estas dos entregas parciales puntúan en el apartado de "Ejercicios de Bash a realizar en clase" de la guía docente.

Entrega final

Fecha prevista de entrega del proyecto final: semana del 11 al 15 de noviembre. Tomaremos para la corrección tanto la hora previa de Estructuras de Computadores como la nuestra, en tu horario de prácticas. Asignaremos franjas horarias a cada uno.

El script realizado se subirá tras la corrección del profesor cuando éste dé el visto bueno.

Este script se revisará en clase por el profesor de prácticas, el alumno deberá comentar el código y contestar a las preguntas que plantee el profesor.

Recibirá una nota entre 0 y 10 según el grado de completitud y lo bien acabado que esté el script conforme a la rúbrica de esta tarea.

4. Notas técnicas

En este apartado iremos añadiendo algunas observaciones para facilitarte la resolución de algunos problemas.

4.1 Utilización de la propiedad local para variables en funciones.

Es probable que tengas que usar unos cuantos nombres de variables en tu script, y si no tienes cuidado acabarás con nombres repetidos en variables “globales” del script y en variables que quieres considerar locales en funciones. Esto es una fuente de error a veces difícil de encontrar. Para evitarlo puedes marcar las variables que consideras locales a las funciones como “local”. Si lo haces así, esa variable “local” dentro de la función será considerada como una variable diferente. Mira este código de la referencia de GNU Bash, para entender el efecto de marcar una variable como local en una función, y como se “arrastra” hasta funciones que llamas desde la primera.

```
func1()
{
   local var='func1 local'
   func2
}

func2()
{
   echo "In func2, var = $var"
}

var=global
func1
echo “In main script, var=$var”
```

El resultado que se va a mostrar es:
In func2, var = func1 local
In main script, var=global

En resumen, puedes evitarte problemas si marcas las variables que consideras locales a tus funciones con local.

4.2 Formato decimal del porcentaje de memoria utilizada

El campo %mem que se puede obtener con ps es el porcentaje de memoria utilizada por el proceso. Es un campo que no da un valor entero sino decimal. Por ejemplo: 3.2.

Este tipo de valores puede ser problemático por dos razones. La primera es que al tratarse de un número decimal puede requerir sus propias reglas, por ejemplo de cara a la ordenación. Es el caso de sort por ejemplo que requiere de la opción -g para este tipo de datos.

Otro efecto menos evidente se refiere a la localización. En determinadas localizaciones, el punto decimal se expresa como una , (3,2) mientras que en otras es un . (3.2). Es el caso de awk, donde el problema estriba en la conversión de la cadena “3.2” al valor binario, realizada por una función de librería interna que sí tiene en cuenta la localización. Por ejemplo:

```
echo "3.2" | awk '{ print 2*$1 }'
```

Produce en mi máquina

```
6
```

en lugar de 6.4. La razón es que el campo 1 leído por awk es “3.2”, pero la librería de conversión de cadenas a números que utiliza esperaría “3,2” puesto que aplica la localización de mi máquina. Si hago:

```
echo "3.2" | awk '{ print 2*$1 }'
```

Se obtiene el resultado

```
6,4
```

que es el esperado. Para solucionarlo se recomienda establecer la localización estándar básica de C, mediante la variable de entorno LC_ALL asignada para la ejecución del comando:

```
echo "3.2" | LC_ALL=C awk '{ print 2*$1 }'
```

De esta manera el resultado es

```
6.4
```

También es conveniente usar este sistema para las llamadas a sort.