

Sistemas Operativos

[Práctica 1 – SCRIPTING]

CRISTIAN LAURENTIU SINDILA

Tabla de contenido

[Autores 2](#_Toc126220791)

[Descripción del Código 3](#_Toc126220792)

[Diseño del Código 3](#_Toc126220793)

[Casos de Prueba 3](#_Toc126220794)

[Comentarios Personales 4](#_Toc126220795)

[Problemas Encontrados 4](#_Toc126220796)

[Críticas Constructivas 4](#_Toc126220797)

[Propuesta De Mejoras 4](#_Toc126220798)

[Evaluación Del Tiempo Dedicado 4](#_Toc126220799)

# Autores

**CRISTIAN LAURENTIU SINDILA**

**NIE : Y0347100-K**

**INGENIERÍA DE COMPUTADORES**

# Descripción del Código

## Diseño del Código

**Diagrama UML:**

Diagrama

Descripción generada automáticamente

El objetivo principal del script es buscar información referente a usuarios y realizará diferentes funcionalidades según el número y el tipo de argumentos introducidos durante la llamada del script:

* Si no introducimos **ningún argumento**, aparecerá por pantalla un mensaje de ayuda del script que mostrará de forma esquematizada el correcto uso de éste indicando los tipos de argumentos que acepta. En el código se realiza una llamada a una función llamada “*echoHelp*” que contiene el comando *echo*, el cual imprime las instrucciones por pantalla.
* Si introducimos **un único argumento** tenemos varios casos según la comprobación de éste:
  + Si el argumento es “*—login*”, entonces se mostrará por pantalla:
    - Las 5 últimas conexiones **remotas**. Para ello se hace uso del mandato *grep* para filtrar todas aquellas líneas que concuerdan con la *expresión regular*, escrita en el código, en el directorio predeterminado establecido anteriormente (“*AUTH\_LOG*”). La expresión comienza por “sshd” y termina con “ssh2”. Imprimimos las columnas en el orden deseado mediante *cut* y solo mostraremos las últimas 5 usando *tail*. Y salimos con éxito.
    - Las 5 últimas conexiones **locales**. Estas siguen otra estructura la cual comienza por “*pam\_unix*” y contiene “*session opened for user*” que indica un correcto inicio de sesión. Seleccionamos éstas y descartamos aquellas que contienen *CRON, root* y *sshd* ya que …

Para ello utilizamos igualmente grep y seguimos imprimiendo, usando los mandatos de la misma manera que las conexiones anteriores. Y salimos con éxito.

* + Si el argumento es “*—help*” volvemos a llamar función llamada “*echoHelp*” y salimos retornando un 0 del script.
  + Si el argumento es *“-u”* o *“-g”* el script imprime por pantalla una pequeña ayuda mediante un *echo* diciendo que se debe proporcionar otro argumento como usuario o grupo, saliendo con un error de argumentos.
  + En el resto de los casos (cualquier único argumento) se ofrece la manera de llamar a la ayuda del script y finaliza con errores de argumentos.
* Si introducimos **dos argumentos** se crean tres casos según el tipo de primer argumento:
  + Si el primer argumentoes “*-u*”: Se comprobará en el archivo binario */etc/passwd* si el usuario, introducido como segundo argumento, existe mediante su correspondiente filtrado usando *grep*. Si no existe finalizará el script con un ERROR\_NOT\_FOUND. En cambio, si existe realizará varias acciones:
    - Mostrará si el usuario introducido es el que está **conectado actualmente,** mediante el uso de *who.*
    - **Las 5 últimas conexiones remotas y locales** de ese usuario. Usará el mismo método descrito en el apartado anterior, pero se seleccionarán solo aquellas que correspondan al usuario y se omitirá la visualización del usuario correspondiente a cada conexión.
    - **Los grupos a los que pertenece el usuario**. Para ello se realiza una búsqueda en el archivo */etc/group*. Se crea una lista de strings de la primera columna del archivo binario con una correspondencia entre ésta, la posterior enumeración y el usuario.
    - Mediante el mandato “*du*” se **muestra el espacio ocupado** por la carpeta del usuario, aunque antes se hace una pequeña comprobación de permisos ya que para acceder a estos directorios se necesita permisos del propio o *root*.
    - Por último, se mostrará los archivos superiores a 1MB mediante un listado usando *find.*
  + Si el primer argumento es “-g“: Se comprobará en el archivo binario “*/etc/group*” si el grupo, introducido como segundo argumento, existe mediante *cat* y *grep*  para su filatrado. Se guardará en una lista, la cual se comprobará si está o no vacía. Si lo está finalizará el script con ERROR\_NOT\_FOUND indicando además por pantalla que no existe el grupo. En cambio, si existe se mostrará la lista que contendrá todos los usuarios de ese grupo y saliendo del script con SUCCES.
* Si el número de argumentos es **superior a dos**, el script saldrá con ERROR\_ARGS e imprimirá por pantalla un mensaje de uso excesivo de número de argumentos.
* En **el resto de los casos** finalizará el script con un ERROR\_NO\_GENERIC y un mensaje de error no típico.

## Casos de Prueba

**<< LISTA Y BREVE EXPLICACIÓN DE CASOS DE PRUEBA UTILIZADOS PARA VALIDAR LA PRÁCTICA >>**

# Comentarios Personales

**<< MÁXIMO 2 PÁGINAS >>**

## Problemas Encontrados

**<< LISTA PROBLEMAS ENCONTRADOS A LA HORA DE REALIZAR LA PRÁCTICA >>**

## Críticas Constructivas

**<< CRÍTICAS CONSTRUCTIVAS TANTO SOBRE EL DESARROLLO PROPIO DE LA PRÁCTICA COMO DE LA PRÁCTICA EN SÍ, SI LAS HUBIERA EN CADA CASO >>**

## Propuesta De Mejoras

**<< PROPUESTA DE MEJORAS TANTO DEL DESARROLLO PROPIO DE LA PRÁCTICA COMO DE LA PRÁCTICA EN SÍ, SI LAS HUBIERA EN CADA CASO >>**

## Evaluación Del Tiempo Dedicado

**<< EVALUACIÓN DEL TIEMPO DEDICADO EN HORAS >>**