Ejercicios shell script.
Administración de Redes Linux

Optativa, Segundo Cuatrimestre.
Iria Estévez Ayres
Universidad Carlos III de Madrid.
Curso 2024–2025

Ejercicio 1 Script para imprimir los argumentos

Escribir un *script* `bash` que imprima por pantalla los argumentos recibidos por la línea de comandos. Deben repetirse en varias líneas, suprimiendo en cada una de ellas el primer argumento de la anterior. El resultado final parecerá un triángulo, tal como se muestra en los siguientes ejemplos:

```
unix$repiteArgs.sh Este script repite los argumentos
Este script repite los argumentos
script repite los argumentos
repite los argumentos
los argumentos
argumentos
unix$
```

```
unix$ repiteArgs.sh otra prueba de ejecucion
otra prueba de ejecucion
prueba de ejecucion
de ejecucion
ejecucion
unix$
```

Notas: Obsérvese que el número de argumentos es variable, el *script* debe ser genérico. Es decir, se debe utilizar un bucle while.

Ejercicio 2 Información sobre el sistema (Entregable)

Se pide escribir un *script* con nombre hello.sh que salude al usuario e imprima la siguiente información en este orden:

```
Hola Nombre_usuario!

Soy Nombre_script, mi PID es Número_proceso
el PID de mi proceso padre es Número_proceso_del_proceso_padre.
Ahora es: día_y_hora
En UTC: día_y_hora_en_utc

Me has invocado con Número_argumentos argumentos, que son:
Todos: todos los argumentos
Argumentos_recibidos cada uno en una línea con el formato Arg[i] = argumento
En orden inverso: todos en orden inverso y como un array separado por comas

Estás en el ordenador  Nombre_ordenador,
con IP dirección_ip_ordenador
que es un Tipo_ordenador
usando el sistema operativo Sistema_operativo,
con núcleo nombre_núcleo
versión del núcleo (versión) version_del_núcleo
(release) release_del_núcleo

Tu identificador es Identificador_usuario
tu directorio personal es Directorio_raíz_usuario
y tu PATH: Variable_PATH_usuario
Estás trabajando en el directorio directorio_en_el_que_se_encuentra
El anterior directorio donde trabajaste es directorio_anterior
```

Ejemplo de ejecución:

```
iria@adagio:~/TRABAJO/DOCENCIA/arlinux/Material/bash/scripts$ ./ej22_2023.sh a b c 1 2 42

Hola iria!

Soy ./hello.sh, mi PID es 13934
el PID de mi proceso padre es 9244
Ahora es:  vie 12 ene 2024 11:32:59 CET
En UTC:  vie 12 ene 2024 10:32:59 UTC

Me has invocado con 6 argumentos, que son:
Todos:  a b c 1 2 42
Arg[1] = a
Arg[2] = b
Arg[3] = c
Arg[4] = 1
Arg[5] = 2
Arg[6] = 42
En orden inverso y como array con comas: {42, 2, 1, c, b, a}

Estás en el ordenador adagio
con IP:  192.110.110.35
que es un x86_64
usando el sistema operativo GNU/Linux
usando un núcleo Linux
```

```
versión del núcleo (version) #101-Ubuntu SMP Tue Nov 14 13:30:08 UTC 2023
(release) 5.15.0-91-generic

Tu identificador es 1001
tu directorio personal es /home/iria
y tu PATH: /home/iria/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:
/usr/local/games:/snap/bin:/snap/bin:/home/iria/.local/bin:/home/iria/.local/bin

Estás trabajando en el directorio /home/iria/TRABAJO/DOCENCIA/arlinux/Material/bash/scripts
El anterior directorio donde trabajaste es /home/iria/TRABAJO/DOCENCIA/arlinux/Material/bash
```

Notas:

- El script entregado (*hello.sh*) debe poder ser ejecutado invocando *bash hello.sh*. Si no es así, se puntuará con un cero.
- Se debe entregar el script y uno o varios pantallazos de ejecución donde se pueda ver el entorno de ejecución y la hora del sistema.
- El *script* debe obtener su información sólo de variables de entorno, variables del *script* y programas auxiliares. Es decir, para imprimir el nombre de usuario no puede utilizar una línea como echo iria.
- Entre otros, pueden usarse los programas: `uname`, `hostname`, `whoami`.
- El *script* debe de imprimir los argumentos del *script* uno a uno, por tanto se debe utilizar un bucle for y la variable del *script* pertinente.
- Para calcular el contador en el bucle de los argumentos, se puede usar una variable que se incremente en el bucle *expr*. También se pueden usar los dobles paréntesis.
- El *script* debe imprimir los argumentos en orden inverso, como un array separados por comas.
- La salida de un comando puede capturarse, de modo que no se imprima directamente por la salida estándar sino que pueda utilizarse en el script. Para ello, se encierra el comando entre acentos graves: `` `comando` ``
- El comando echo tiene una opción para no pasar a la línea siguiente después de imprimir el texto. Puedes consultarlo en el manual si lo necesitas.

Ejercicio 3 Lectura de usuario

Vamos a implementar un juego. Nuestro *script* adivina, que recibirá como parámetro un número (m), calculará UNA SOLA VEZ de forma ALEATORIA otro número entre 1 y m. Una vez calculado, le pedirá al usuario que adivine el número con el comando read. Mientras el usuario no lo adivine le seguirá ofreciendo esa posibilidad de que lo adivine hasta que lo acierte.

El *script* debe producir la siguiente salida:

```
unix$adivina 12
Adivina el numero (entre 1 y 12):12
PRUEBA OTRA VEZ
Adivina el numero (entre 1 y 12):11
PRUEBA OTRA VEZ
Adivina el numero (entre 1 y 12):10
PRUEBA OTRA VEZ
Adivina el numero (entre 1 y 12):5
HAS ACERTADO
El numero era 5
```

En el caso de que el número de argumentos sea distinto de uno, debe presentar el siguiente mensaje de error:

```
unix$adivina
El script debe tener un argumento
unix$adivina 2 3 4
El script debe tener un argumento
```

Para calcular el número aleatoriamente se ha de utilizar la variable RANDOM, que cada vez que accedes a ella genera un valor entre 0 y 32767. Para comprobar esto realizar:

```
unix$ echo $RANDOM
11328
unix$ echo $RANDOM
26812
unix$echo $RANDOM
3651
```

Para que el número se genere entre 1 y máximo se debe implementar la siguiente fórmula: $1 + maximo * RANDOM/32768$. Deberás usar el comando `expr` para evaluar esta expresión

Realizar $num\_aleatorio/32768$ nos da un número entre 0 y 1. Si se quiere más información respecto a fórmulas de generación de números aleatorios, consultar la página de manual de rand.

Ejercicio 4 Test de ficheros en directorios

Se desea diseñar un script, llamado test.sh, que analice los ficheros de un determinado directorio y que nos muestre por pantalla el nombre de los que cumplan unas determinadas características. Una vez hecho esto preguntará al usuario si desea seguir procesando directorios, en caso afirmativo, repetirá el proceso con el nuevo directorio especificado por el usuario.

El *script* recibirá 0 o 1 argumento:

- Recibe 0 argumentos: procesa los ficheros del directorio del usuario

- Recibe 1 argumento: el nombre del directorio en el que debemos procesar los ficheros

La ejecución del programa seguirá los siguientes pasos:

1. Procesar los argumentos (si los hay). Recuerda que si no hay argumentos el directorio por defecto a usar es el home del usuario
2. Entrar en el directorio
3. Imprimir su nombre con el formato indicado
4. Para todos los ficheros del directorio:
   - Comprobar si es un directorio y se puede entrar. En ese caso imprimir su nombre
   - Comprobar si es un fichero y se puede escribir y ejecutar. En ese caso imprimir su nombre
5. Preguntar al usuario si quiere seguir procesando directorios.
   - Si la respuesta es "n" ir a 6
   - Si no, preguntarle el nuevo directorio y volver a 2
6. Finaliza el programa

Ejemplos:

```
unix$./tipo.sh
Procesando el directorio: /home/iria
Directorios en los que se puede entrar:
prueba
Entregas
test
Ficheros que se pueden escribir y ejecutar:
ejecuto
programa
Quiere procesar otro directorio? (s/n) n
unix$
```

Como se puede ver el directorio por defecto es nuestro directorio home. En el caso de tener parámetros el comportamiento es el siguiente:

```
unix$./tipo.sh /home/iria/prueba
Procesando el directorio: /home/iria/prueba
Directorios en los que se puede entrar:
Ficheros que se pueden escribir y ejecutar:
programilla
Quiere procesar otro directorio? (s/n) s
Introduzca el nombre de directorio: /home/iria/test
Procesando el directorio: /home/iria/test
Directorios en los que se puede entrar:
test2
Ficheros que se pueden escribir y ejecutar:
Quiere procesar otro directorio? (s/n) n
unix$
```

Ejercicio 5 Uso del comando sed

El programa `sed` procesa una por una las líneas de un fichero y permite la aplicación de comandos sencillos a cada una de las líneas. Una posible aplicación de este comando es la creación de ficheros a partir de otros a través de la substitución.

El fichero `defectuoso.html` (lo puedes encontrar en Aula Global) no es un fichero html válido, pues nos hemos equivocado al crearlo y en vez de utilizar el tag `p`, que marca inicio y fin de párrafo, hemos usado un tag llamado `parr`.

Utiliza el comando `sed` para reemplazar los tag `<parr>` por los tag `<p>`, y los tag de fin `</parr>` por `</p>`. Para comprobar que se ha hecho bien visualizar el fichero resultante en un navegador.

Nota: para redireccionar la salida estándar del comando `sed` a un fichero se debe utilizar $>$

Ejercicio 6 Seguir los ascendentes de un proceso

Escribir el *script* followppid que dado un entero como único parámetro imprima una línea por cada proceso en la cadena de procesos padre (PID creado por el proceso PPID) hasta llegar al proceso con PID igual a 1. El *script* debe comprobar que recibe un único parámetro.

Debes usar el comando `awk`.

Ejemplo:

```
unix$ followppid
Script debe recibir un parametro (y solo uno): el PID a seguir.
unix$ followppid 2165
 2165 creado por el proceso  2161
 2161 creado por el proceso  2160
 2160 creado por el proceso  2157
 2157 creado por el proceso  2086
 2086 creado por el proceso  2084
 2084 creado por el proceso     1
unix$
```

Nota: Utilizando las opciones apropiadas, el comando ps puede imprimir una única línea que contiene solamente el PID del proceso padre.

Ejercicio 7 Script para buscar archivos y obtener su tamaño

Se desea diseñar un script bash que busque los archivos pertenecientes a un determinado usuario y, para cada uno de ellos, presente su nombre y tamaño. El

script permite especificar, mediante argumentos, el directorio en el que se buscarán los ficheros y el nombre de usuario.

Para cada fichero encontrado se mostrará su nombre y tamaño, según el siguiente formato:

```
fichero tamaño
```

El comando `find` permite especificar un comando a ejecutar sobre cada fichero encontrado. Utiliza esta opción para efectuar un listado largo de cada archivo. La salida puede procesarse, mediante un pipe con el comando `awk`, para presentar los campos pedidos con el formato anterior.

El script debe admitir las siguientes opciones, en cualquier orden:

- -h: Imprime el siguiente mensaje de ayuda y termina (con estado 0):

```
unix$ ./buscar.sh -h
Ayuda:
buscar [-u usuario] [-d directorio] [-h]
```

- -u usuario: Presenta sólo los ficheros cuyo propietario es el usuario especificado. Si no se incluye esta opción, por defecto se presentarán todos los ficheros, independientemente de su propietario.
- -d directorio: Busca los ficheros en el directorio especificado (y recursivamente en sus subdirectorios). Si no se incluye esta opción, se procesará el directorio actual (.).

Control de errores:

- Debe comprobarse que el directorio existe y tenemos permiso de entrada. En caso contrario se presentará un mensaje de error por la salida de error estándar y se terminará con estado 1. Por ejemplo, suponiendo que 'varios' no es un directorio (o no tenemos permiso de entrada), el script escribiría:

```
unix$ ./buscar.sh -d varios
varios no es un directorio o no tiene permiso de entrada
```

- Debe comprobarse que el usuario existe en el sistema. Para ello, debe utilizarse el comando id. Si el usuario no es correcto, debe aparecer el siguiente mensaje y el script terminará con estado 2:

```
unix$./buscar.sh -u yo
id: yo: No existe ese usuario Usuario incorrecto
```

- Si se incluye la opción -h se imprime la ayuda, sin realizar comprobaciones ni realizar la búsqueda.

- Si se incluye algún argumento diferente de los descritos, simplemente se ignora.

Nota: para que los ficheros no aparezcan repetidos, utiliza la opción adecuada para que find no seleccione los directorios (es decir, que procese sólo ficheros regulares y enlaces).

MUY Importante: se ha de mantener el orden de las comprobaciones y el formato de salida.

Ejercicio 8 Contar Procesos en Ejecución

Se desea diseñar un script que permita contar el número de procesos en ejecución.

El script permite contar los procesos, seleccionando todo ellos, o sólo los que pertenezcan a un determinado usuario, o los asociados a un cierto programa, según las opciones utilizadas.

Invocación:

```
unix$contarproc [opciones]
```

El script debe admitir las siguientes opciones:

- Sin argumentos: Cuenta todos los procesos en ejecución.
- -u usuario: Cuenta sólo los procesos pertenecientes al usuario especificado. Presenta un mensaje de error si se utiliza sin especificar ningún nombre de usuario.
- -p programa: Cuenta sólo los procesos asociados al programa especificado. Presenta un mensaje de error si se utiliza sin especificar ningún programa.
- -help: Opción que imprime la ayuda (una breve descripción de las opciones con las que se puede ejecutar el script) y termina.

El script debe utilizar la función cuenta, que calcula el número de líneas de la entrada estándar que contienen la palabra que se pasa como argumento. Para ello usará el comando `wc`.

Una opción para seleccionar los procesos sería comprobar simplemente que la línea contiene el nombre de usuario/programa, con `grep`. Sin embargo, otra solución más precisa sería comprobar exactamente la columna en cuestión (usuario/programa).

La versión mejorada de la función cuenta utiliza un comando de procesamiento de texto (`awk`) para seleccionar la columna deseada antes de relizar el filtrado y contar las líneas. En este caso, los argumentos que recibe la función son dos:

- 1^{er} parámetro: el texto a buscar
- 2^0 parámetro: la columna a seleccionar (la que debe contener el texto buscado)

Notas:

- El script no debe utilizar ficheros temporales.
- Para que el script funcione correctamente, primero deben interpretarse todas las opciones y después realizar las acciones oportunas.
- En caso de error en el uso de las opciones, el programa presentará el mensaje de error, la ayuda y terminará, con estado 1. Si todo funciona correctamente, terminará con estado 0. Si se presenta la ayuda, terminará con estado 1 (aunque la opción se haya utilizado correctamente)

Ejercicio 9 Lectura de usuario mejorada

Vamos a mejorar nuestro juego de adivinación de números.

Nuestro *script* adivinaF, que recibirá como parámetro un número (m), calculará UNA SOLA VEZ de forma ALEATORIA otro número entre 1 y m. Una vez calculado, le pedirá al usuario que adivine el número con el comando read. Mientras el usuario no lo adivine le seguirá ofreciendo esa posibilidad de que lo adivine hasta que lo acierte. Además dispondrá de la opción –help que mostrará la ayuda, y de la opción –pistas que nos dará pistas.
Tanto para presentar la ayuda como para aportar las pistas se han de utilizar funciones.
El *script* debe producir la siguiente salida para la ayuda:

```
unix$adivinaF -help
usage: adivinaF [argumentos] maximo
   Calcula un número entre 1 y maximo, y le pide al usuario
   que lo adivine.
Argumentos:
  -help     Imprime Help (este mensaje) y acaba
  -pistas   Aporta pistas al usuario
```

Con la opción de pistas ha de producir la siguiente salida:

```
unix$adivinaF -pistas 20
Adivina el numero (entre 1 y 20):3
3 es menor que el numero
PRUEBA OTRA VEZ
```

```
Adivina el numero (entre 1 y 20):11
11 es mayor que el numero
PRUEBA OTRA VEZ

Adivina el numero (entre 1 y 20):7
7 es mayor que el numero
PRUEBA OTRA VEZ

Adivina el numero (entre 1 y 20):6
HAS ACERTADO
El numero era 6
```

Sin opciones:

```
unix$adivina 12
Adivina el numero (entre 1 y 12):12
PRUEBA OTRA VEZ

Adivina el numero (entre 1 y 12):11
PRUEBA OTRA VEZ

Adivina el numero (entre 1 y 12):10
PRUEBA OTRA VEZ

Adivina el numero (entre 1 y 12):5
HAS ACERTADO
El numero era 5
```

Los errores serán:

- *script* sin parámetros.
- Utilización de la opción pistas sin indicar el máximo
- Especificar más de un máximo

Se debe mostrar un mensaje de error y la ayuda.
Se recuerda que la fórmula a implementar es: $1+maximo*RANDOM/32768$

Ejercicio 10 Feliz Navidad

Se desea diseñar un *script* que presente un mensaje al usuario cada cierto tiempo. En concreto, una felicitación de Navidad.

El *script* simplemente presenta por pantalla la frase "Feliz Navidad", espera un tiempo determinado, vuelve a presentar el mensaje... y así sucesivamente. Opcionalmente, el *script* permite especificar el nombre del usuario, para que aparezca en el mensaje, el tiempo de espera entre mensajes y el idioma.
Invocación:

```
unix$felizNavidad [opciones]
```

Opciones:

- –t segundos Permite especificar el tiempo de espera entre mensajes, en segundos. Por defecto, si no se utiliza esta opción, son 5 segundos
- –n nombre Permite personalizar el mensaje. En este caso, se presentaría por pantalla: Feliz Navidad nombre Si no se especifica esta opción, no se imprime ningún nombre.
- –i idioma Permite especificar el idioma del mensaje:
  - esp : Español (”Feliz Navidad”)
  - ing : Inglés (”Merry Christmas”)
  - gal : Gallego (”Bo Nadal”)

  Por defecto, si no se utiliza esta opción, se escribe el mensaje en español.
- –help: Opción que imprime la ayuda (una breve descripción de las opciones con las que se puede ejecutar el script) y termina:

```
unix$ FelizNavidad.sh -help
FelizNavidad.sh [-help] [-t segundos] [-n nombre] [-i idioma]
  -help       Ayuda. Muestra este mensaje.
  -t segundos Espera los segundos especificados entre mensajes.
  -n nombre   Incluye el nombre de usuario en el mensaje.
  -i idioma   Idioma del mensaje, por defecto, español.
              Idiomas admitidos:
              - esp : español
              - ing : ingles
              - gal : gallego
```

Para cambiar el mensaje según el idioma, se programará la función seleccionarIdioma:

- Esta función recibe como argumento el idioma deseado y selecciona el mensaje correspondiente.
  Sugerencia: utiliza una sentencia case.
- Si no recibe argumento o no es uno de los idiomas permitidos, la función presenta un mensaje de error (y el programa principal presentará la ayuda y terminará)
  Sugerencia: Utiliza una variable global (mensaje) para almacenar el texto a imprimir. La función seleccionarIdioma simplemente tendrá que modificar esta variable.

Notas:

- Las opciones pueden invocarse en cualquier orden.

- Para que el *script* funcione correctamente, primero deben interpretarse todas las opciones y después realizar las acciones oportunas, tal como se explica más arriba.
- Comprobación de errores: si se produce un error en el uso de las opciones, debe presentarse el mensaje correspondiente, la ayuda y terminar (sin hacer nada más)
- Sólo se comprobará que, si se utilizan, las opciones -t, -n y -i van seguidas de otro argumento, y que el idioma es admitido por el programa (no es necesario comprobar que sea un número en el caso del tiempo...)
- Si todo es correcto, se entra en un bucle infinito que presente el mensaje adecuado periódicamente.
- En caso de error, el programa terminará con estado 1. Si todo funciona correctamente, terminará con estado 0.
- Si se presenta la ayuda, terminará con estado 1 (aunque la opción se haya utilizado correctamente)

Ejercicio 11 Cálculo de los n primeros números primos

Escriba un script que escriba los N primeros números primos, siendo N el único argumento del script. A continuación se detalla un pseudocódigo que puede utilizarse para escribir esos N primeros números primos.

```
function esprimo(n:integer):boolean {
 a=2
 primo=true
 while primo and a*a <= n {
    if n%a = 0 {
       primo=false
    }
    a=a+1
 }
 return primo
}

i=1
n=1
while n <= N {
 if esprimo(i) {
```

```
    print(i)
    n=n+1
 }
 i=i+1
}
```

Ayuda: El módulo entre dos números puede calcularse con el operando `'%'` del comando `expr`.

Ejercicio 12 Una agenda en csv (Entregable)
Crear el script `ej_agenda.sh` que gestione un fichero de agenda.
El fichero de agenda:

- Contendrá una línea por cada registro de evento a anotar.
- El formato de cada línea será dd/mm/yy;Evento;Comentario
  - donde dd/mm/yy es la fecha del evento en formato dia/mes/año
  - `evento` y `comentario` son textos libres
  - todos los campos de la línea están separados por el carácter ;

Ejemplo de fichero agenda

```
11/02/24;Domingo de Carnaval;Recordar comprar disfraz
11/02/24;Domingo de Carnaval;Hacer filloas
10/02/24;Comida con colegas;Mesón Pepe, 14:00
15/02/24;Reunión de contabilidad;Pedir el listado de inventariable
```

Comportamiento del script
El script puede recibir dos parámetros (opciones) y seis acciones diferentes.
Las **opciones** permitidas son:

- `-f fichero` define el fichero a utilizar. Por defecto el fichero es `agenda.txt`.
- `-h` presenta la ayuda del script. Si se presenta esta opción, sólo se presenta la ayuda, no se realizan otras acciones.

Las **acciones** permitidas son:

- **listar**: lista el contenido de la agenda, según el formato
  `n:dd/mm/yy Evento Comentario`
  donde n es el número de registro dentro del fichero. Los campos deberán estar separados por un tabulador.

```
$ ./ej_agenda.sh listar
1:11/02/24 Domingo de Carnaval Recordar comprar disfraz
2:11/02/24 Domingo de Carnaval Hacer filloas
3:10/02/24 Comida con colegas Mesón Pepe, 14:00
4:15/02/24 Reunión de contabilidad Pedir el listado de inventariable
```

- **fecha dd/mm/yy**: lista los eventos de la fecha dada, usando el formato anterior.

```
  $ ./ej_agenda.sh fecha 11/02/24
1:11/02/24 Domingo de Carnaval Recordar comprar disfraz
2:11/02/24 Domingo de Carnaval Hacer filloas
$ ./ej_agenda.sh fecha 10/02/24
3:10/02/24 Comida con colegas Mesón Pepe, 14:00
```

- **agregar dd/mm/yy evento comentario**: añade un nuevo evento con la información dada.

```
$ ./ej_agenda.sh agregar 10/02/24 natación "recordar comprar gorro"
$ cat agenda.txt
11/02/24;Domingo de Carnaval;Recordar comprar disfraz
11/02/24;Domingo de Carnaval;Hacer filloas
10/02/24;Comida con colegas;Mesón Pepe, 14:00
15/02/24;Reunión de contabilidad;Pedir el listado de inventariable
10/02/24;natación;recordar comprar gorro
$ ./ej_agenda.sh listar
1:11/02/24 Domingo de Carnaval Recordar comprar disfraz
2:11/02/24 Domingo de Carnaval Hacer filloas
3:10/02/24 Comida con colegas Mesón Pepe, 14:00
4:15/02/24 Reunión de contabilidad Pedir el listado de inventariable
5:10/02/24 natación recordar comprar gorro
```

- **borrar fecha dd/mm/yy**: elimina todos los registros correspondientes a una fecha.

```
$ ./ej_agenda.sh borrar fecha 10/02/24
$ ./ej_agenda.sh listar
1:11/02/24 Domingo de Carnaval Recordar comprar disfraz
2:11/02/24 Domingo de Carnaval Hacer filloas
3:15/02/24 Reunión de contabilidad Pedir el listado de inventariable
$ cat agenda.txt
11/02/24;Domingo de Carnaval;Recordar comprar disfraz
11/02/24;Domingo de Carnaval;Hacer filloas
15/02/24;Reunión de contabilidad;Pedir el listado de inventariable
```

- **borrar registro nnn**: elimina el registro cuyo numero de registro es el dado en la acción.

```
$ ./ej_agenda.sh borrar registro 2
$ ./ej_agenda.sh listar
1:11/02/24 Domingo de Carnaval Recordar comprar disfraz
2:15/02/24 Reunión de contabilidad Pedir el listado de inventariable
$ cat agenda.txt
11/02/24;Domingo de Carnaval;Recordar comprar disfraz
15/02/24;Reunión de contabilidad;Pedir el listado de inventariable
```

Importante:

- Se recomienda revisar entre otros los comandos `awk`, `sed`, `cut` y `grep`. Esto no quiere decir que la solución haga uso de todos los comandos listados: existen distintas soluciones posibles que hacen uso de conjuntos de comandos diferentes.
- Se deben documentar detalladamente todas las decisiones tomadas, mediante comentarios en el script.
- Si el fichero entregado no ejecuta adecuadamente se considerará no válido y será calificado con un cero.
- Se debe entregar una memoria con pantallazos de la ejecución del script donde se pueda ver el entorno de ejecución, la fecha y en algún sitio el nombre del autor del script.

Ejercicio 13 Comprobación de resúmenes de ficheros (Entregable)

Muchos ataques a sistemas informáticos consisten en la substitución de ficheros (por ejemplo, ejecutables y bibliotecas de funciones) por otros que, aparentemente, hacen lo mismo, pero que incorporan, además, código malintencionado (por ejemplo, un virus).

Para detectar este tipo de ataques, se podría comparar todos los ficheros de la máquina con los originales, periódicamente. Esta solución necesita demasiados recursos, tanto de espacio de almacenamiento como de tiempo de procesado.

Otra opción consiste en no comparar el contenido de los ficheros, sino un resumen de estos. Normalmente, el tamaño del resumen es mucho menor que el del fichero, y se genera con algoritmos especialmente diseñados para que, con una probabilidad muy alta, cualquier cambio que ocurra en un fichero provoque un cambio en su resumen.

Se pide programar un script que compruebe si han cambiado los directorios especificados por el usuario, utilizando su resumen. El resumen se debe generar utilizando el comando `sha256sum` descrito posteriormente. El script debe trabajar en dos modos distintos:

- **Modo creación**: genera un resumen de todos los ficheros regulares contenidos en los directorios especificados, y recursivamente en sus subdirectorios (puede ser útil el comando `find`). Este resumen se almacena en el fichero especificado con la opción `-f`, con una línea por cada fichero sobre el cual se realice el resumen.
- **Modo comprobación**: comprueba si los ficheros han cambiado con respecto al resumen anteriormente generado.
  - Si no hay cambios, debe mostrar el mensaje "No se han detectado cambios."
  - Si los hay: "Se han detectado cambios."

El script recibe como parámetros las siguientes opciones:

- `-n`: establece el modo de creación.
- `-c`: establece el modo de comprobación.
- `-f fichero_de_resumenes`: fichero sobre el cual escribir/leer el resumen de los ficheros.
- `-h`: muestra un mensaje de ayuda, y finaliza la ejecución inmediatamente.
- Sólo en el modo de creación, y a continuación de las opciones anteriores, una lista con uno o más nombres de directorio.

Se han de cumplir las siguientes restricciones:

- El script recibe las opciones en cualquier orden.
- Las opciones `-n` y `-c` son mutuamente excluyentes, y es obligatorio especificar una de ellas. Si se incluyen las dos a la vez, se considera un error.
- La opción -f es obligatoria en ambos modos.
- En el modo de comprobación es necesario verificar que el fichero de resúmenes existe y se puede leer.
- Ante cualquier error, el script finalizará su ejecución, mostrando el mensaje .ºpciones incorrectas.", y el mismo mensaje de ayuda que se obtiene con la opción -h
- El script no puede mostrar por pantalla más mensajes de los especificados en este enunciado.
- El script no debe utilizar ningún fichero adicional.
- Es necesario documentar detalladamente todas las decisiones tomadas.

El comando sha256sum:

- `sha256sum fichero`: vuelca a su salida estándar una línea de texto con el resumen de fichero y su nombre, separados por espacios en blanco.
- *sha256sum -c fichero_de_resumenes*: lee el fichero de resúmenes dado, calcula de nuevo los resúmenes de todos los ficheros a los cuales éste hace referencia, y comprueba que concuerden. Si todos los resúmenes concuerdan (no cambiaron los ficheros), finaliza con status 0. Si alguno no concuerda, finaliza con status distinto de cero.

Ejemplo uso de sha256sum:

```
$ sha256sum /home/iria/bin/clean_latex.sh
c4b1f51f28867e928a44280d2b3915ac497f3177b444762929dff41a10fe15a6  /home/iria/bin/clean_latex.sh
```

Ejemplo de fichero de resúmenes:

```
8ced6b2b38818ec2ed1b48181c97b57f3775359d4b554eeaec7c39749491b629  /home/iria/bin/eduroam_config_file.py
c4b1f51f28867e928a44280d2b3915ac497f3177b444762929dff41a10fe15a6  /home/iria/bin/clean_latex.sh
0a22649f02ebd12a7ff9e88c7666ccbcb357fae2f423626dedbc22319d172b12  /home/iria/bin/iconv_utf8.sh
```