Sistemas Operativos - Práctica de Control II

Sergiy Khudoley

5 de mayo de 2024

Índice

1. Preludio

Estoy iniciando este proyecto con conocimientos muy básicos del lenguaje de scripting **bash**. Los antecedentes que he podido tener de este lenguaje son más bien anecdóticos.

Con esta práctica estoy teniendo mi primer contacto con un script relativamente complejo, y se puede observar en la **inconsistencia del formato** que utilizo para escribir el código. Este ha ido variando bastante desde mis primeros intentos hasta el momento presente.

Apenas estoy aprendiendo la sintaxis básica y las posibles opciones de las que dispongo para atajar diferentes problemas. Viendo las **similitudes** y **diferencias** respecto a otros lenguajes de programación.

Para mis primeros contactos he utilizado de forma exhaustiva **ChatGPT**. Este me ha permitido ganar tracción con el lenguaje de forma extremadamente rápida. Pero pronto he encontrado que tenía un serio problema. Esto me producía **mucha inconsistencia** en el código.

Al no conocer todavía las opciones disponibles, no sabía cual era la forma **óptima** o al menos que estilo me venía mejor en ese momento. Por ello, una vez me sentía más cómodo me empezó a ser más útil referirmente a soluciones encontradas en **Stackoverflow**.

Por último, en la mayor parte del tiempo, simplemente me he referido al guión de la **práctica 6** dada en clase. Ya que habitualmente la mayor parte de dudas solían ser más bien dudas de sintaxis de resolución rápida.

2. Requerimientos

2.1. Referencias

2.1.1. Referenciar script original

Este script debe localizar todos los comentarios existentes de cualquier fichero **.sh** que se encuentre en un determinado directorio. Esto incluye posibles subdirectorios anidados.

El script debe localizar todos estos ficheros y crear una numeración del tipo #**ES_10**, teniendo en cuenta que la referencia se genera con **dos letras** mayúsculas seguido de un _ y la numeración del comentario, comenzando el comentario por 10 e incrementandose de 10 en 10 posteriormente.

2.1.2. Creación de fichero para idiomas

El fichero de script generará otro grupo de ficheros, los que sean necesarios dependiendo de la cantidad de idiomas con los que esté trabajando con el prefijo de cada idioma y acabado en la extensión **.txt**.

2.1.3. Intercambio con comentarios externos

El script puede pedir seleccionar uno de los idiomas disponibles, y debe ser capaz de intercambiar los comentarios guardados en los fircheros **.txt** por sus respectivos comentarios numerados denre del fichero original.

2.1.4. Archivos de log

En caso de existitr algun problema en la sustitución de los comentarios, debe guardarse el comentario a insertar en un archivo **.log** para su posterior comprobación.

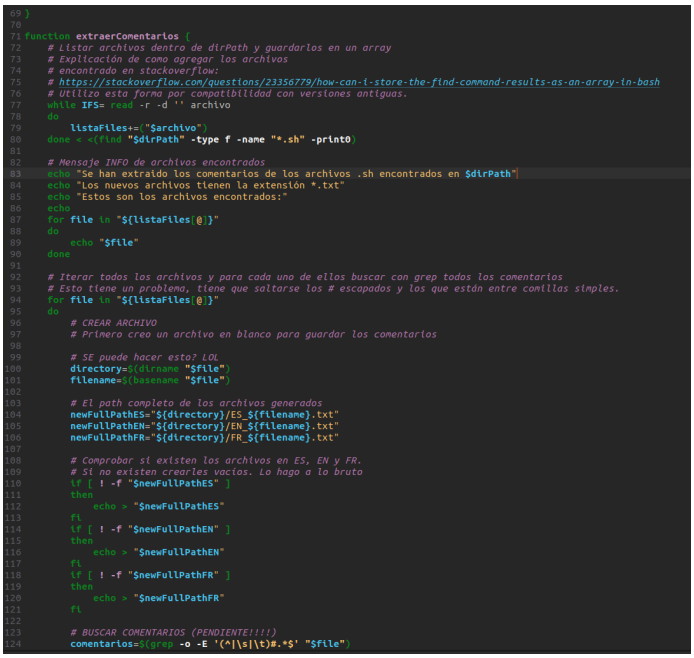
2.2. Idiomas

2.2.1. Trabajando con multiples idiomas

El script genera un archivo oculto llamado **.idiomas** en el mismo directorio donde se guardarán los prefijos/idiomas a utilizar. Estos serán utilizados para pedir al usuario que idioma desea utilizar para el intercambio de comentarios y cuantos ficheros **.txt** debe generar al crear las referencias.

2.2.2. Agregando idiomas adicionales

El script permite también el insertado de nuevos idiomas para poder trabajar con ellos posteriormente.



```
}

function extraerComentarios {
    # Listar archivos dentro de dirPath y guardarlos en un array
    # Explicación de como agregar los archivos
    # encontrado en stackoverflow:
    # https://stackoverflow.com/questions/23356779/how-can-i-store-the-find-command-results-as-an-array-in-bash
    # Utilizo esta forma por compatibilidad con versiones antiguas.
    while IFS= read -r -d '' archivo
    do
        listaFiles+=("$archivo")
    done < <(find "$dirPath" -type f -name "*.sh" -print0)

    # Mensaje INFO de archivos encontrados
    echo "Se han extraido los comentarios de los archivos .sh encontrados en $dirPath"
    echo "Los nuevos archivos tienen la extensión *.txt"
    echo "Estos son los archivos encontrados:"
    echo
    for file in "${listaFiles[@]}"
    do
        echo "$file"
    done

    # Iterar todos los archivos y para cada uno de ellos buscar con grep todos los comentarios
    # Esto tiene un problema, tiene que saltarse los # escapados y los que están entre comillas simples.
    for file in "${listaFiles[@]}"
    do
        # CREAR ARCHIVO
        # Primero creo un archivo en blanco para guardar los comentarios

        # SE puede hacer esto? LOL
        directory=$(dirname "$file")
        filename=$(basename "$file")

        # El path completo de los archivos generados
        newFullPathES="${directory}/ES_${filename}.txt"
        newFullPathEN="${directory}/EN_${filename}.txt"
        newFullPathFR="${directory}/FR_${filename}.txt"

        # Comprobar si existen los archivos en ES, EN y FR.
        # Si no existen crearles vacios. Lo hago a lo bruto
        if [ ! -f "$newFullPathES" ]
        then
            echo > "$newFullPathES"
        fi
        if [ ! -f "$newFullPathEN" ]
        then
            echo > "$newFullPathEN"
        fi
        if [ ! -f "$newFullPathFR" ]
        then
            echo > "$newFullPathFR"
        fi

        # BUSCAR COMENTARIOS (PENDIENTE!!!!)
        comentarios=$(grep -o -E '(^|\s|\t)#.*$' "$file")
```

Figura 1: 1ª versión del script

3. Primera tentativa y proyecto de git

3.1. Inicio del proyecto

Mi primer intento crear el script ha sido más bien erratico.

He empezado a crear los procedimientos que necesitaba de forma procedural para ver que problemas me podría encontrar por el camino. Pronto me topé con el primer problema.Había decidido buscar los comentarios utilizando **grep**.

Empezé buscando diferentes casos que escapaban al alcance del patrón que estaba utilizando. Esto hizo que le dedicara bastante esfuerzo sin realizar ningún avance significativo.

A la vez el código empezó pronto a crecer más de lo que esperaba y a tener la necesidad de modificarlo. Tenía que empezar a plantearme el como estructurarlo.

Aquí decidí empezar a dividir el código en funciones y plantearme como quería organizarlo.

En este momento me sentía con fuerzas para realizar un planteamiento diferente. Ya no estaba tan centrado en el **como** sino en el **qué**. Este fue el punto de inflexión.

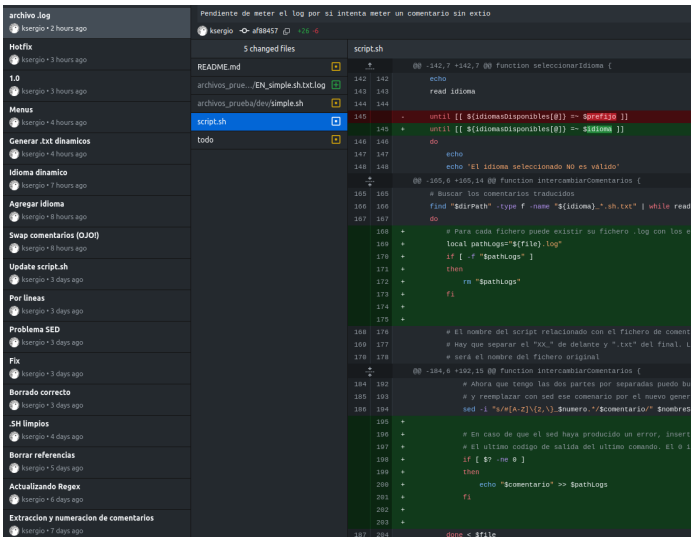


Figura 2: Historial de cambios con git

3.2. Reinicio del proyecto con git

Llegado este punto he decidido crear un listado de requerimientos que necesitaba resolver. Además ahora adobte la mentalidad de resolver un problema, dejar el código funcionando, aunque no estuviese completamente satisfecho. Primero debía r**esolver la mayor cantidad de problemas** posibles, y **pulirlos a posteriori**.

Para facilitarme la trazabilidad decidí iniciar un proyecto con **git**. Además, esto me permitía tener un backup sencillo de los archivos **.sh** que estaba modificando. El proyecto de git tiene dos ramas, una **main** y otra **dev** en la cual realizo todos los cambios.

Para mayor detalle se puede recurrir al historial, así como mis anotaciones en un archivo **README.md** a modo de vitacora. Aquí se puede ver un poco el progreso y en que pensaba cada día. A parte llevaba un archivo **todo** como un recordatorio de que tenía que hacer inmediatamente a continuación en cada momento.

4. Soluciones propuestas

4.1. Método para buscar los comentarios con grep y find

Para solucionar el problema de encontrar todos los ficheros **.sh** simplemente he utilizado el comando:

```
find "$dirPath" -type f -name "*.sh"
```

Habitualmente para ser utilzado a través de un pipe con un bucle **while**. También encontré otra versión de utilizarlo que me pareció un poco menos legible

```
while read linea; do
    # Hacer algo con $linea
done <<< texto.txt
```

Para realizar la busqueda de los comentarios estoy usando un simple **grep** con una expresión regular curiosa que me he encontrado.

Esta expresión busca el incio o el caracter # separado por un espacio o tabuliación. Lo he encontrado bastante efectivo.

```
grep -o -E -n '(^|\s|\t)#[^!#].*$' "$file"
```

La opcion **E** me da acceso a las expresiones extendidas, **n** me permite insertar la linea donde se encuentra y **o** para sacar solo la parte que coincide, no toda la linea.

```
while IFS=: read -r numero_linea comentario
```

Combinando con el separador **IFS=:** tenemos disponibles dos variables, el numero de linea y comentario en el bucle while.

4.2. Creación de un menú interactivo

Para crear el menú he copiado descaradamente los menús utilizados en la práctica de control, con una corección.

Figura 3: Menu interactivo para seleccionar la operación

```
function menuIdiomas {
    local opcion=0

    #validacion
    until ([[ $opcion > 0 && $opcion < 3 ]])
    do
        echo
        echo '---- Idiomas -------------------------'
        echo
        echo '1) Agregar'
        echo '2) Ver disponibles'
        echo

        read opcion
    done

    case "$opcion" in
        '1') agregarIdioma;;
        '2') verIdiomasDisponibles;;
    esac
}
```

En este caso no veo la necesidad de repetir le impresión o pedir la entrada al usuario antes de realizar el bucle. Se puede hacer dentro del bucle como en el ejemplo anterior. De esta forma reduzco duplicación de código.

4.3. Borrado de comentarios en el fichero original

Para borrar las referencias existentes en el archivo original, lo hago usando **sed** y usando un patrón con expreisones regulares.

```
sed -i -e 's/#\([A-Z]\{1,\}_[0-9]*\)*/#/g' $file
```

El formato de comentario una vez generada la referencia es del tipo #**XX_9999**. Dos letras y cualquier cantidad de numeros separados por un guión bajo. Este tiene un tema un poco curioso, y es que durante el desarollo tuve un error en el cual podía crear varias referencisa encadenadas. Por ejemplo; ES_10ES_20_ES_30. Con el patrón que he marcado aquí atrapo el formato todas las veces que se repita.

En la práctia si el referenciado esta bien hecho, no debería ocurrir nunca, ya que me he tomado la molestia de borrar las referencias siempre antes de generarlas.

4.4. Creando ficheros .txt para almacenar datos

Una vez encontrado el **path** del fichero con el que se trabaje puedo generar los archivos necesarios para todos los idiomas. De la siguiente forma extraigo el nombre del archivo del path completo:

```
directorioPadre=$(dirname "$file")
nombreFichero=$(basename "$file")
```

Una vez que tengo el nombre del archivo ya puedo generar el nuevo fichero con el **prefijo** especifico del idioma deseado y **.txt**.

4.5. Intercambiando comentarios

Para intercambiar los comentarios lo que hago es buscar, de nuevo, con expresiones regulares el prefijo y el comentario.

```
's/\(.*\/\)[A-Z]\{2\}_\(.*\)\.txt$/\1\2/')
```

Del prefijo extraigo el numero del comentario y el comentario en si, y a posteriori busco en el archivo donde se tiene que sustituir con la misma numeración.

```
"s/#[A-Z]\{2,\}_$numero.*/$comentario/"
```

Un problema que me dió bastantes problemas, fue el hecho de escapar caracteres en el código. En las páginas de pruebas online donde he realizado las pruebas **regex101.com** no lo necesita, pero mi script si que se debe.

Figura 4: Insertando comentarios de traducción

4.6. Agregando idiomas nuevos y persistencia

Para trabajar con distintos idiomas estoy usando un archivo oculto **.idioma** donde almaceno los idiomas como 2 letras en mayúsculas. Este archivo lo utilizo para poder persistir los datos entre ejecuciones.

Al inicio del código siempre compruebo si existe, en caso contrario creo un archivo por defecto

```
function crearIdiomasStorage {
    local file='./.idiomas'

    # En caso de no existir crearlo de nuevo
    if [ ! -f './.idiomas' ]
    then
        touch .idiomas
        # Idiomas por defecto
        echo 'ES' >> './.idiomas'
        echo 'EN' >> './.idiomas'
    fi
}
```

A continuación lo carga en una array con esto. Esto es un truco curioso, pero no siempre es válido. Existen otras formas de cargar datos en un array más robustos, incluso una nueva forma para **bash >4.4** usando **readarray**.

Yo he decidido usar una forma **curiosa** por que sí.

```
fileItemString=$(cat  './.idiomas' |tr "\n" " ")
idiomasDisponibles=($fileItemString)
```

Por otro lado previo a cualquier ejecución de alguno de las funcionalidades,

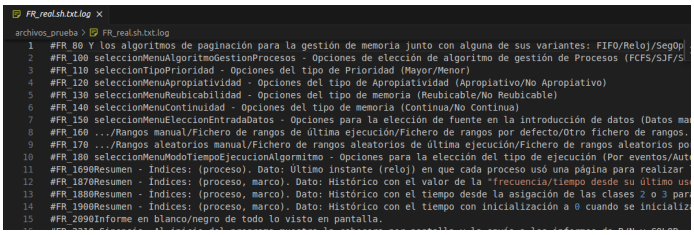


Figura 5: Archivo de log de errores al insertar comentarios

siempre leo este array y solicito al usaurio que me diga con que idioma quiere trabajar. Para ello uso una variable global.

4.7. Agregando ficheros .log para errores de insertado

Esta es la última funcionalidad que he agregado, y no está del todo pulida. Esto queda pendiente de ser extendido, pero por ahora es la forma más básica de atajar el problema de los insertados incorrectos.

```
if [ $? -ne 0 ]
then
    echo "$comentario" >> $pathLogs
fi
```

Con este código compruebo que la ultima ejecución del comando es correcto. Lo uso después de un sed. En caso de dar un error en el insertado sale con un error 1 o 2 que será atrapado en el código.

5. Posibles mejoras

5.1. Patron regex para atrapar comentarios

Respecto a este apartado por ahora he encontrado que es bastante efectivo, pero puede ser que haya algunas lineas de comentario que no atrape. He creado un fichero con **corner cases** o **casos extremos**. Por ejemplo, falla para lineas que hagan echo con comillas simples que tengan un caracter # dentro. Esto **NO** tiene por que ser un comentario.

5.2. Escapado de caracteres con sed

Ahora mismo para el reinsertado de comentarios de un idioma estoy usando un **sed** pero no he encontrado una forma fácil de escapar los posibles caracteres que existan en las variables usadas. Por ello, hay errores con algúnos de los insertados que están pendientes de corregir.

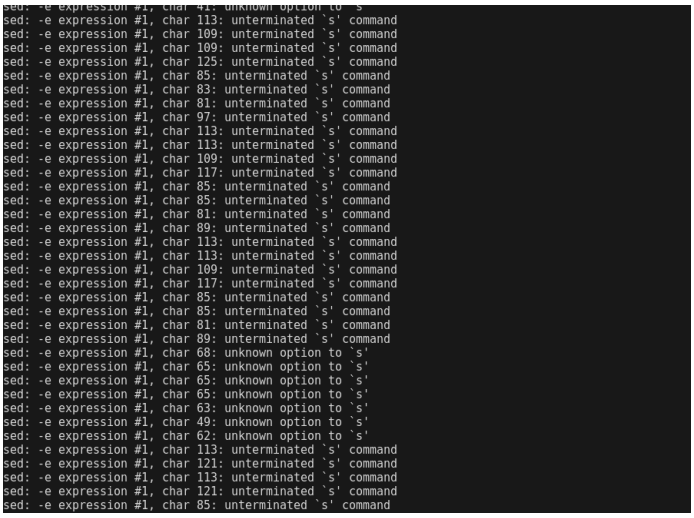


Figura 6: Errores en la inserción con sed

5.3. Abuso del sed

Me ha resultado la forma más fácil de realizar sustituciónes en archivos. También he encontrado la sustitución de parámetros de bash, pero para ello necesito tener las dos cadenas que quiero sustituir localizadas. Para escribir en un archivo en algún lugar concreto la única forma que conozco simple era con un **sed**. Pero esto me lleva a leer el mismo entero archivo multiples veces para sustituir una única línea.

De la misma forma para realizar saltos de linea hago llamadas multiples de **echo**, aunque entiendo que esto no es tan intensivo como la lectura de un archivo completo. En mi caso yo directamente he despreciado posibles penalizaciones por llamarlo multiples veces en vez de hacer un comentario multilinea.

5.4. Insertado de lineas problemáticas en fichero .log

Actualmente el insertado solo se hace si da error el sed. Pero esto no ocurre si la sustitución es vacía o no se realiza porque no se encuentra.

Habría que comprobar si se ha hecho al menos una inserción. Mi idea primera era revisar el archivo antes y después de cada insertado y comprobar si este ha sido modificado con un **diff** por ejemplo. Pero me resulta excesivo hacer una lectura de dos archivos para cada linea.

Esto queda pendiente de encontrar alguna forma eficiente de comprobarse.

6. Conclusión personal

Este trabajo me ha resultado muy útil para conocer más en profundidad los comandos disponibles de **Linux**, sus opciones y peculiaridades.

Siempre he deseado aprender el lenguaje **bash** pero nunca he tenido disponibilidad y he utilizado sustitutos para el scripting como **Python**. Por un lado me parece más legible y obviamente más flexible y potente, pero por otro lado esto supone agregar **complejidad adicional** y alejarme de las herramientas que ya provee el propio **sistema operativo**.

Además usar otro lenguaje implica posibles puntos de fallo adicional, en cambio con bash estamos trabajando directamente con el sistema. No conozco si existe una diferencia de rendimiento importante utilizando estos dos lenguajes, pero entiendo que este debe ser más eficiente.

En definitiva, me alegra el haber tenido la oportunidad de realizar un trabajo para ampliar mi repertorio y forma de pensar.