

Práctica Control Sistemas Operativos - Bash

Sergiy Khudoley

1 de junio de 2024

Script con de internacionalización
de comentarios en lenguaje Bash

Índice

1. Preludio	4
2. Punto de partida del proyecto	5
2.1. Motivación y objetivos	5
2.2. Conocimiento inicial del lenguaje Bash	6
2.3. Herramientas y recursos usados	6
3. Funcionamiento y principales características	8
3.1. Idiomas	8
3.1.1. Ver idiomas disponibles	8
3.1.2. Agregar nuevo idioma	8
3.1.3. Borrar idioma existente	9
3.2. Referencias	9
3.2.1. Generar nuevas referencias	9
3.2.2. Intercambiar idiomas de referencias	10
3.2.3. Agregar referencias adicionales	11
3.2.4. Borrar referencias	11
3.2.5. Reenumerar referencias existentes	12
4. Desarrollo del script	12
4.1. Menús de navegación	12
4.2. Trabajando con distintos idiomas	12
4.3. Búsqueda archivos .sh	14
4.4. Búsqueda de comentarios	14
4.5. Usando sed para susituciones	15
4.6. Nuevas referencias y ficheros de traducción	16
4.7. Intercambiando idiomas	16
4.8. Agregar referencias adicionales	16
4.9. Reenuemeración de referencias existentes	17
5. Mejoras a implementar	17
5.1. Refactorizado general	17
5.2. Expresiones regulares y atrapado de comentarios	18
5.3. Escpado de caracteres en el uso de sed	18
5.4. Acotar y/o limitar el uso de sed	19
5.5. Agregar opciones adicionales	19
5.6. Comprobación de existencia de archivos	19
6. Conclusiones personales	19
6.1. Problemas encontrados	19
6.1.1. Escape de caracteres	19
6.1.2. Rellenar array con while y pipe	19
6.1.3. Declare -a dentro de una funcion	20
6.1.4. Iterar arrays de cadenas de texto	20

Práctica Control - Bash

6.2. Sobre el uso de Chat GPT	20
6.3. Habilidades adquiridas	21

1. Preludio

En total se registraron más de 62.000 inmersiones, revelando que los pingüinos juveniles se movieron inicialmente hacia el norte para alcanzar áreas de aguas abiertas y aguas más cálidas. .Este es el momento en que esencialmente están aprendiendo a nadar”. ”No es algo que sus padres les enseñen. Cuando entran al agua por primera vez, son muy torpes e inseguros de sí mismos. No son los nadadores rápidos y elegantes en los que los más afortunados se convertirán”.

La dura infancia del pingüino emperador - National Geographic España

2. Punto de partida del proyecto

2.1. Motivación y objetivos

Este proyecto trata sobre la creación de un script en lenguaje **bash** para el tratamiento de comentarios de ficheros de tipo **.sh** generando unos ficheros de traducción.

En concreto el script extrae a un fichero de texto plano todos los comentarios, los marca con el prefijo del idioma en el que se encuentran y los indexa según una numeración secuencial que irá en incrementos de 10 unidades.

Adicionalmente genera ficheros para todos los idiomas que vayamos a trabajar, de manera que tendremos:

1. Un fichero con el idioma actual y todos los comentarios numerados
2. Un fichero por cada idioma adicional con las referencias sin texto

De tal manera que podemos rellenar los ficheros de los idiomas con las traducciones correspondientes y pedirle al script que intercambie los nuevos comentarios traducidos por los contenidos en el fichero de script original.

De esta forma conseguimos traducir unicamente las partes relacionadas con el comentarios dejando intacto el resto del código.

Ahora podemos rellenar nuestros ficheros de traducciones con los correspondientes comentarios traducidos y utilizar una opción del script para intercambiar cada comentario referenciado de un idioma a otro deseado.

Por otro lado, el script nos permite agregar comentarios que hayamos creado adicionalmente a posteriori. Lo único que tendríamos que hacer es insertar una referencia con una numeración intermedia entre otros dos comentarios.

Esto ultimo generaría una nueva numeración para las referencias. El script también nos permite volver a reenumerar todas las referencias para tener una numeración nueva constante respetando el orden anterior.

El objetivo en última instancia es **familiarizarse** con el lenguaje de scripting de bash creando esta herramienta de utilidad reusable, así como una breve introducción al pensamiento lógico para iniciarnos en la programación.

2.2. Conocimiento inicial del lenguaje Bash

Esta prácticas las empiezo con nula experiencia en el lenguaje bash. Sí, he interactuado con la **consola** con sistemas **linux** para realizar tareas y gestiones básicas, pero nunca he tenido la necesidad de crear ningún script de bash. Para cualquier tarea que ha requerido de una complejidad mayor he utilizado **Python** ya que a día de hoy está presente por defecto en muchas distribuciones de linux y se integra con suma facilidad.

En esta ocasión vamos a utilizar bash ya que, por otro lado, esté es inherente a los sistemas linux. Python necesita ser agregado *a posteriori*, pero bash ya está presente y es parte de la misma plataforma y reduce la complejidad de agregar otra parte móvil, que puede ser motivos de fallo, al sistema.

2.3. Herramientas y recursos usados

Teniendo en cuenta que no había trabajado antes con bash, salvo la **práctica del tema 6** que me ayudo a tener un primer contacto a las situaciones más comunes que se me podían presentar, para la primera tentativa para empezar a construir el script me apoye mucho en el uso de **Chat GPT**. Esta fué la forma más rápida de ganar tracción con el lenguaje, ya que me proporcionaba conceptos y respuestas *“válidas”* para los problemas que me iba topando.

En un principio no tenía una estructura clara de como debería lucir por ello el código era bastante caótico y pronto se volvió un auténtico caos. En este momento era conciente que este no sería el proyecto final, pero aún así decidí seguir para usarlo como un borrador en el cual **experimentar** con bash.

En el momeneto que me sentí cómodo y con un poco de seguridad decidí volver a reempezar el proyecto esta vez usando **git** como gestor de versionados. Reutilizé algunas partes del borrador para empezar a construir el proyecto real.

A partir de aquí empecé a utilizar y apoyarme más sobre **Stackoverflow** y los aportados en la asignatura. Stackoverflow para localizar preguntas complejas y específicas, y los PDF de la asignatura a modo de índice rápido para consultas de ^osin^otaxis y ejemplos sencillos, que en la práctica era a lo cual más recurriría.

Más adelante en la sección de conclusión personal hablo más sobre el uso de **Chat GPT**, así como donde y como me ha ayudado, y donde resultaba mejor ignorarlo.

En el proyecto además del propio historial de **git**, tengo un archivo **README.md** en formato markdown con una breve explicación y un historial donde se ve el proceso de la escritura del script y mis anotaciones.

```
echo
echo "Vamos a insertar los comentarios en $idiomaSeleccionado"

archivoOriginal='inter.sh'
archivoInsertar="${idiomaSeleccionado}_inter.sh.txt"

# Saco el numero de comentarios en el script
numComOrig=$( grep -E '(\s|\t)#+.*$' $archivoOriginal | wc -l)
# Saco el numero de comentarios en el archivo a insertar
numComNuev=$( wc -l "$archivoInsertar" )

# Deben coincidir los valores del ultimo comentario a insertar con los comentarios totales * 10
# PJ: Si hay 50 comentarios, el ultimo deberá ser 500 (o intermedio con el siguiente 505)
# Nunca puede ser el 510, en ese caso estoy insertando comentarios sobre cosas que no existen.
# IMPORTANTE! Tener en cuenta que al hacer append se pone en una línea nueva, por lo cual siempre
# marcará que hay una línea a mayores en el $numComNuev.

# Voy a mirar el ultimo numero de comentario a insertar, No debe exceder el numero de líneas*10+9
numComMax=$((numComOrig*10+9))
echo "Comentario max: -> $numComMax"

# Buscar la numeracion máxima de los comentarios nuevos que voy a introducir
# Presupongo que es el que aparece al principio de la ultima línea.
# Un poco chapuzillas, pero estoy aprendiendo.
numComNuevoMax=$( tail -n 1 "$archivoInsertar" )
numeroMayorInsertar=$(grep -o '^[0-9]*' <<< $numComNuevoMax)
echo "Numero máx comentario a insertar: $numeroMayorInsertar"

if [ $numeroMayorInsertar -gt $numComMax ]
then
    echo 'La numeración de los comentarios a insertar es '
    echo 'superior a los comentarios existentes. Si quieres'
    echo 'insertar comentarios adicionales deben de estar'
    echo 'entre los ya existentes. Max: $numComMax'
fi

function menuInicio {
    echo 'MENU INICIO'
    echo '1) Saludar'
    echo '2) Extraer comentarios'
    echo '3) Intercambiar comentarios'
    read seleccionMenuInicio
    until ([ $seleccionMenuInicio > 0 && $seleccionMenuInicio < 4 ])
    do
        echo "Error en la elección de una opción válida"
        echo '1) Saludar'ine.
        # Un poco chapuzillas, pero estoy aprendiendo.
        numComNuevoMax=$( tail -n 1 "$archivoInsertar" )
        numeroMayorInsertar=$(grep -o '^[0-9]*' <<< $numComNuevoMax)
        echo "Numero máx comentario a insertar: $numeroMayorInsertar"

        if [ $numeroMayorInsertar -gt $numComMax ]
        then
            echo 'La numeración de los comentarios a insertar es '
            echo 'superior a los comentarios existentes. Si quieres'
            echo 'insertar comentarios adicionales deben de estar'
```

Figura 1: Primera tentativa del script desechada

```
898
899 #####
900 # Inicio de ejecución del script
901 #####
902
903 # Ejecución
904 cargarIdiomasDisponibles
905 cabecera
906 menuInicio
907
908
909
910 #####
911 ## Idiomas disponibles
912 ## Se puede pero NO DEBE agregar y quitar desde el mismo script.
913 ## NO insertar manualmente! Porque NO generaría los
914 ## ficheros de traduccion necesarios.
915 ## DEBE EXISTIR UN SALTO DE LINEA AL FINAL DEL ULTIMO IDIOMA, SI NO, NO FUNCIONA.
916 #####
917 #EN-Ingles
918 #ES-Español
919
```

Figura 2: Persistencia de datos de idiomas de forma dinámica en el archivo

3. Funcionamiento y principales características

3.1. Idiomas

El proposito del script es la traducción de los comentarios de ficheros .sh, para ello necesitamos especificar con que idiomas vamos a trabajar. Estos idiomas se guardan sobre el mismo archivo, por lo cual no tiene dependencia de archivos adicionales. Si fuese necesario se puede modificar sobre el mismo script. El apartado de la persistencia de los idiomas se encuentra al final del archivo, pero lo idoneo es trabajar directamente con el menú dedicado desde el script ya que este agrega funcionalidades extra descritos más adelante.

3.1.1. Ver idiomas disponibles

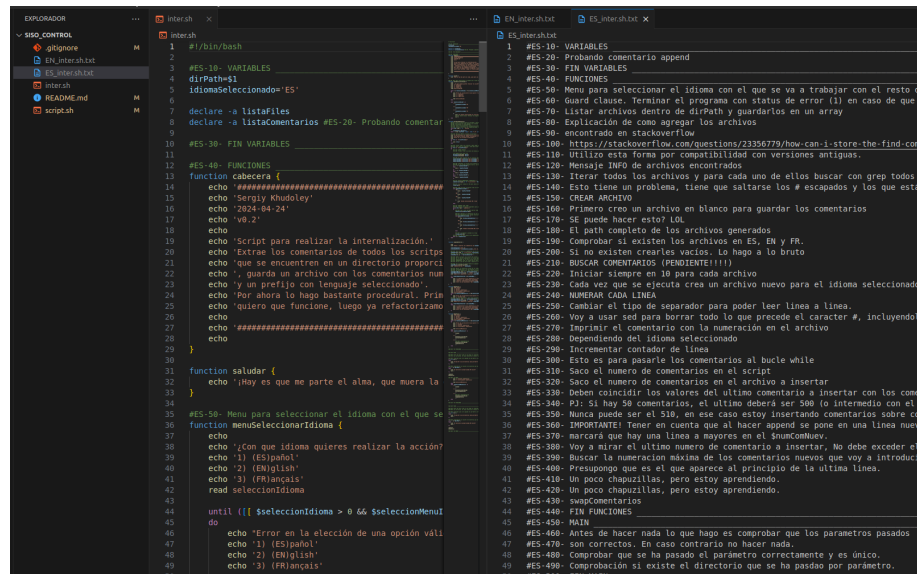
Esta opción nos lista los idiomas disponibles. Cada uno de estos idiomas será usado para las opciones de intercambio de comentarios (cambiar los comentarios de un idioma a otro) y para crear los fichers de traducción específicos.

3.1.2. Agregar nuevo idioma

Esta opción nos guarda en el listado de idiomas un idioma adicional. Nos pide por un lado el prefijo que se usará y el nombre completo para tener una mejor referencia.

Además, en el caso de que ya existan ficheros de traducción generados, si agregamos un idioma, este creará el fichero de traducción específico adicional y lo

Práctica Control - Bash



```
1 #!/bin/bash
2
3 #ES-10- VARIABLES
4 dirPath=1
5 idiomaSeleccionado= 'ES'
6
7 declare -a listafiles
8 declare -a listaComentarios #ES-20- Probando comentar
9
10 #ES-30- FIN VARIABLES
11
12 #ES-40- FUNCIONES
13 function cabecera {
14     echo "#####"
15     echo "Sergiy Khudoley"
16     echo "2024-04-24"
17     echo "v0.2"
18     echo "Script para realizar la internalización."
19     echo "Extrae los comentarios de todos los scripts"
20     echo "que se encuentran en un directorio proporc"
21     echo "y guarda un archivo con los comentarios num"
22     echo "y un prefijo con lenguaje seleccionado."
23     echo "Por ahora lo hago bastante procedural. Prip"
24     echo "quiero que funcione, luego ya refactorizamo"
25     echo "#####"
26 }
27
28 function saludar {
29     echo "¡Hay es que me parte el alma, que muera la"
30 }
31
32 #ES-50- Menu para seleccionar el idioma con el que se
33 function menuSeleccionarIdioma {
34     echo "Con que idioma quieres realizar la acción?"
35     echo "1) (ES)pañol"
36     echo "2) (EN)glish"
37     echo "3) (FR)ançais"
38     read seleccionIdioma
39
40     until { [[ $seleccionIdioma > 0 && $seleccionMenu
41     do
42         echo "Error en la elección de una opción vál"
43         echo "1) (ES)pañol"
44         echo "2) (EN)glish"
45         echo "3) (FR)ançais"
46     }
47
48 #ES-60- MAIN
49 #ES-70- son correctos. En caso contrario no hacer nada.
50 #ES-80- Comprobar que se ha pasado el parametro correctamente y es único.
51 #ES-90- Comprobar si existe el directorio que se ha pasado por parametro.
52 #ES-100- crear menu
```

Figura 3: Extracción y referenciado de comentarios de un script

rellenará con las referencias que existan en el script en ese momento. No modifica los otros ficheros de traducción existentes.

3.1.3. Borrar idioma existente

Borra un idioma del listado de idiomas disponibles. A diferente del agregado, el borrado **no** borra el fichero de traducción para poder mantener las referencias existencias para su futuro tratamiento o reuso.

3.2. Referencias

Esta es el apartado de la funcionalidades principales del script. Veamos a continuación que podemos hacer.

3.2.1. Generar nuevas referencias

Primero nos solicita con que idioma es el que vamos a considerar que estamos trabajando en el script.

Esta opción nos pedirá una confirmación primero, ya que lo que hace es **sobre-escribir** todos los ficheros de traducción existentes.

Lo primero que se realiza es la posible eliminación de referencias anteriores que existan con el mismo patrón que vayamos a usar para referenciar. Esto quita

```
# El path completo de los archivos generados para cada idioma
pathTraduccion="${directorioPadre}/${i}_${nombreFichero}.txt"

# Compruebo si existe la numeracion en los ficheros de traduccion
referencia=$(grep -E "#${i}-${numero}" "$pathTraduccion" | head -n 1)

if [ -z "$referencia" ]
then
    # Lo que voy a hacer es buscar el numero inmediatamente anterior e insertar este comentario justo delante en los archivos de traduccion.
    # Para encontrar el anterior voy a ir decrementando el numero hasta que coincida con algo.

    # Inicializamos un contador para decrementar el número
    num_anterior=$((numero - 1))

    ##### POR AQUI ANDA LA MOVIOINHA #####
    # Buscamos la referencia anterior con un bucle while
    while [ $num_anterior -ge 0 ]
    do
        referencia_anterior=$(grep -E "#${i}-${num_anterior}" "$pathTraduccion" | head -n 1)

        # Si encontramos una referencia anterior, insertamos el nuevo comentario justo después de ella
        if [ -n "$referencia_anterior" ]
        then
            # Comprobar si el idioma del comentario en el script coincide con el archivo. En ese caso tiene
            # que insertar el comentario completo
            if [ $prefijo = $i ]
            then
                sed -i -E "/#${i}-${num_anterior}/a\\${comentario}" "$pathTraduccion"
                break
            # En caso contrario simplemente inserta la referencia sin el texto
            else
                sed -i -E "/#${i}-${num_anterior}/a\\#${i}-${numero}" "$pathTraduccion"
                break
            fi
        fi
        # Decrementamos el contador para buscar la siguiente referencia anterior
        num_anterior=$((num_anterior - 1))
    done
fi
```

Figura 4: Agregando nuevos comentarios a fichero de traducción existente

referencias antiguas.

Luego extrae todos los comentarios existentes y los numera de forma secuencial, de 10 en 10.

Por último para cada fichero de traducción inserta el prefijo seguido del numero de cada comentario. Para el idioma seleccionado, además, agregará el texto de los comentarios.

3.2.2. Intercambiar idiomas de referencias

Esta opción buscará el fichero de traducción del idioma seleccionado e introducirá los comentarios según coincidencia de referencias en el script original.

En caso de que un comentario no exista en el fichero de traducción **se dejará en blanco**.

Para el caso de que exista en el de traducción, pero no en el original, **se omitirá** por completo.

```
# Este es el caso de que en numero de referencia != numeracionBucle
# Hay que actualizar la numeracion en el script al numero nuevo.
# Tambien el los ficheros de traduccion.

# 1- Modificar en el script original el numero antiguo por el nuevo que llevo en la variable
sed -i "${numLinea}s/${prefijo}-${numero}-/${prefijo}-${numeracionBucle}-/" $file

# Buscar todos los ficheros de traduccion que tenga esa numeración
for i in "${idiomasDisponibles[@]}"
do
    # Pillar prefijo
    i=${i:0:2}

    # Para trabajar con los paths
    directorioPadre=$(dirname "$file")
    nombreFichero=$(basename "$file")
    pathTraduccion="${directorioPadre}/${i}_${nombreFichero}.txt"

    # Primero localizo el numero de linea en la que está dicha referencia.
    # Esto es necesario para no hacer trabajar a sed sobre todo el fichero de traducción
    # si no solo sobre una única línea. De otra forma cada referencia que queramos editar tendría que leer
    # el archivo de traducción completo una y otra vez.

    # Por otro lado, también puede darse el caso que cambiada una línea, la siguiente tenga la misma numeración.
    # Por ejemplo. la línea 15 pasa a ser la 20 y la siguiente es la 20. Especificando la línea evito este problema.

    # Para resolver este problema voy a escoger el numero que coincida empezando desde atrás.

    numLineaUltima=$(grep -n "${i}-${numero}-" "$pathTraduccion" | tac | head -n 1)
    numLineaUltima=${numLineaUltima%:*}

    # Si se encontró una línea, modificar esa línea específica.
    if [[ -n $numLineaUltima ]]
    then
        sed -i "${numLineaUltima}s/${i}-${numero}-/${i}-${numeracionBucle}-/" "$pathTraduccion"
    fi
done
```

Figura 5: Reenumerando los comentarios

3.2.3. Agregar referencias adicionales

Esta opción nos permite trabajar sobre el fichero de script.

Si decidimos insertar nuevos comentarios sobre el un script, lo único que hay que respetar es el patron de la referencia. Debemos introducir el prefijo del idioma en el que se encuentra y agregar un número intermedio entre otros dos.

Para el orden de inserción se tendrá en cuenta esta numeración y no lo posición. Es decir, si insertamos al final un comentario del tipo: *#ES-11-Este comentario va al final*, este se insertará entre el primer y segundo comentario en el fichero de traducción. Ignora la posición del comentario en el script.

3.2.4. Borrar referencias

La opción de borrar referencia busca todas las coincidencias del patrón usado para crear la referencia y la elimina de los archivos .sh existentes.

3.2.5. Reenumerar referencias existentes

La última opción es precisamente para poder volver a reenumerar las referencias de nuevo.

Si insertamos muchas referencias intermedias podría darse el caso que empecemos a *ensuciar* la numeración de estas. Para evitar esto, simplemente reenumeramos las referencias de los comentarios para volver a enumerarlos de 10 en 10.

Las posibles referencias que **no encontradas** se conservarán para su tratamiento.

4. Desarrollo del script

En esta sección explicaré brevemente que estrategias he seguido para los pasos más importantes y alguna otra indicación que me parece interesante.

4.1. Menús de navegación

Para la creación de menús he necesitado implementar un sistema de validación. La forma más sencilla que me he encontrado es crear una variable que contenga la respuesta seleccionada y acto seguido solicito mediante un **read** al usuario la opción.

Posteriormente en un bucle de tipo **until** comprobamos que la opción está dentro de los parametros que hayamos decidido. Por ejemplo, entre un rango de números o incluso que coincida con un patrón. Si no coincide, le solicitamos con **read** otra vez la entrada. También podemos agregar algún mensaje para avisar del error en la entrada.

Además en la mayoría de menús agrego un **clear -x** para limpiar la pantalla. Con el flag **-x** hacemos scroll en vez de limpiar la consola, conservando la salida de la consola si hacemos scroll hacia atrás.

4.2. Trabajando con distintos idiomas

Los idiomas los guardo al final del archivo con el siguiente formato; **XX-Nombre**

Las dos primeras letras son el **prefijo** que usaré para los nombres de los ficheros de traducción y el inicio de la referencia.

El nombre simplemente es para **evitar ambigüedades** entre distintos prefijos de idiomas.

```
# MENUS #####

function menuReferencias {
    clear -x

    local opcion=0

    #validacion
    until ([ $opcion > 0 && $opcion < 7 ])
    do
        echo
        echo '---- Referencias -----'
        echo '1) Nuevos ficheros de traducción'
        echo '2) Intercambiar por otro idioma'
        echo '3) Agregar comentarios adicionales a fichero de traducción'
        echo '4) Borrar referencias'
        echo '5) Re-enumerar'
        echo '6) Atras'
        echo

        read opcion
    done

    case "$opcion" in
        '1') crearReferencias;;
        '2') intercambiarComentarios;;
        '3') agregarReferenciasAdicionales;;
        '4') borrarReferencias;;
        '5') reenumerarReferencias;;
        '6')
            clear -x
            menuInicio
        ;;
    esac
}
```

Figura 6: Vista de menús del script

Para trabajar con ellos los cargo en una variable de tipo array llamada **idiomasDisponibles**. La forma que tengo de acceder a ellos es usando el comando **tac** (el inverso de **cat**). Leo cada línea como un idioma hasta que me tope con algo que no coincida con el patrón deseado. Aquí simplemente se que la siguiente línea tendrá el comentario **##** pero lo idoneo sería comprobar si el patrón no es el mismo ,para no depender de ese comentario concreto.

Para seleccionar un idioma simplemente tengo otra variable llamada **idiomaSeleccionado** que asigno a un idioma concreto antes de usar otra utilidad cualquiera.

Si necesito trabajar con los idiomas me basta simplemente con iterarlos con un simple **bucle for**.

4.3. Búsqueda archivos .sh

Para trabajar con **todos** los archivos de tipo **.sh** en el mismo directorio, lo he decidido hacer buscandolos con un **find**. De esta forma me aseguro de que si existen directorios anidados también los tenga en cuenta. El comando **find** me resultó la forma más sencilla de implementarlo. Lo único que tenía que agregar era la opción de ignorarse a si mismo.

Una vez leído con **find** lo que hago es rellenar con los datos un array llamado **ficherosScript**. ¡Cuidado! El comando **readarray** se puede usar a partir de la **versión 4.0** de **bash** lanzada en **2009**.

```
local selfName=$(basename $scriptSelfName)

readarray -d '' ficherosScript <<(
    find "$dirPath" \
    -type f \
    -name "*.sh" \
    ! -name "$selfName" \
    -print0
)
```

A partir de aquí para usar los ficheros simplemente podemos trabajar iterando sobre el array.

4.4. Búsqueda de comentarios

Para trabajar buscando los comentarios he decidido utilizar **grep**. En la práctica lo he usado en conjunto con el flag **-n** para extraer el número de línea en donde

Práctica Control - Bash

se encontraba. Además por comodidad he usado el flag **-o** para que grep solo me devolviese el comentario y no toda la línea (esto era útil en comentarios que iban detrás de una línea de código).

Para atrapar los comentarios simplemente he buscado con la siguiente expresión regular y el flag **-E** para habilitar el uso de expresiones regulares extendidas:

```
(^\s|\t)#\[^\].*$
```

Cualquier comentario siempre empieza por el simbolod **#** y existen tres posibles casos:

1. Un comentario de línea. Por lo cual empieza desde el inicio de línea.
2. Un comentario detrás de una línea de código. Por lo cual va precedido por un espacio o tabulación.

También ignoro aquellos en los que **#** va seguido del caracter **!** ya que se trata del **shebang**. Todo el resto hasta el final de línea será nuestro comentario.

Si necesitáramos atrapar solo aquellas que tengan una referencia asignada le tenemos que cambiar el patrón para que coincida con el prefijo, numeración y los guiones adicionales:

```
(^\s|\t)#\[^\][A-Z]{,2}-[0-9]*-.*$
```

Ahora todo esto lo podemos guardar en un array para trabajar con ello, también podemos pasarlo con un pipe al siguiente comando. ¡Cuidado con esto! En problemas encontrados he tenido problemas al pasar datos por el pipe.

4.5. Usando sed para susituciones

Durante todo el script he utilizado el comando **sed** de forma extensiva, quizás hasta he hecho un abuso de él. En muchas situaciones es simplemente más fácil usar un **bash paramter substitution** para trabajar sobre el mismo archivo, pero el sed me ha dado muchas facilidades.

El principal problema que me he encontrado es la necesidad de escapar las variables con las que vayamos a trabajar. He tenido que tener mucha precacuión

con esto y donde he utilizado el comando con el flag **-E** para escapar según cual caracter. En definitiva, esto es un tema escabroso que hay que ver caso a caso.

Una utilidad que me proporcionaba era realizar sustituciones directamente sobre patrones en vez de cadenas exactas.

En la mayor parte de los casos he necesitado simplemente intercambiar una cadena por otra y no varias a la vez, para evitar que sed trabajara con los archivos completos le he pasado solo la línea concreta sobre la que debía realizarlo.

Para la inserción de referencias nuevas también he usado sed para la inserción detrás de una línea concreta.

4.6. Nuevas referencias y ficheros de traducción

Para las nuevas referencias lo que hago es eliminar todos los archivos de traducción de cada script encontrado y generarlos de nuevo.

Buscamos todos los scripts del directorio, luego buscamos todos los comentarios para cada uno de ellos y por ultimo iteramos cada uno de estos comentarios generandole su referencia para cada idioma disponible. Después esta referencia se inserta en su fichero de traducción correspondiente y si el idioma seleccionado para trabajar con el fichero sobre el cual se está escribiendo también agregamos el comentario completo.

4.7. Intercambiando idiomas

Para el intercambio de idiomas, primero solicitamos al usuario introducir el idioma que quiere intercambiar. Posteriormente se busca cada comentario en el script y para referencia se busca la misma numeración en el fichero de traducción.

1. Si existe, entonces el comentario del script es reemplazado por el del mismo número del fichero de traducción
2. Si NO existe, entonces el comentario del script se deja en blanco.

4.8. Agregar referencias adicionales

Para agregar comentarios adicionales lo que hacemos es buscar cada script y cada comentario de estos. Ahora para cada comentario comprobamos si existe en el archivo de traducción. En caso de que no exista debemos introducirlo, pero para saber en que posición debe insertarse según el número.

Para insertarlo en la posición correcta en el fichero de traducción, lo que hago es restarle al número 1 unidad hasta toparme con el primer comentario. Esto hace que encuentre el comentario justamente anterior.

Por ejemplo, si tengo el comentario 16 y en mi fichero de traducción está el 10-13-17-20 lo que haría sería buscar el 15, luego el 14 y por último encontrar la coincidencia del 13. Entonces sabría que hay que introducirlo después de este.

Aquí vuelvo a usar `sed` para hacer una inserción con un *append* justo después del 13.

De esta manera quedarían colocados.

4.9. Reenumeración de referencias existentes

Si repetimos el proceso del apartado anterior e insertamos muchas referencias quizás nuestra numeración pueda *ensuciarse* con mucha numeración errática.

Para poder volver a reenumerarlo de 10 en 10 no puedo borrar las referencias y generarlas de nuevo porque perdería las posibles referencias antiguas de los archivos.

Lo que voy a hacer es iterar cada comentario con referencia del script y llevar un contador en paralelo con la nueva numeración. Si la numeración coincide significa que no hay que modificar nada, en caso contrario implica que **esa** referencia debe ser buscada en el fichero de traducción y ser reemplazada por la **nueva numeración**.

5. Mejoras a implementar

5.1. Refactorizado general

El código se ha ido generando desde cero y sin tener conocimiento previo, por lo cual hay muchas partes que están hechas de una forma innecesariamente larga, con repeticiones, o mediante procesos innecesarios.

Hay funciones que hacen partes parecidas que podría extraer y reutilizar en cada una de ellas. Es el caso de buscar ficheros y comentarios, que a pesar de haber sido implementado no se ha modificado para ser usado de nuevo. Pero si el código siguiera creciendo podríamos empezar a utilizarlas y sustituirlas en las antiguas para tener una reducción de código.

```
### UN AUTENTICO MADMAN #####
# Tengo que escapar los caracteres especiales de bash para poder usarlos en la expresion de sed. Si no, interpreta cosas
# y no funciona como se espera.

# Uso doble // para que sean sustituidos todas las coincidencias, no solo uno
# Los backslash \ por \\
comentarioEscapado=${comentario//\\/\}
comentarioConReferencia=${traduccion//\\/\}
# Los | por \|
comentarioEscapado=${comentarioEscapado//\|/\|}
comentarioConReferenciaEscapado=${comentarioConReferencia//\|/\|}
# Los $ por \$
comentarioEscapado=${comentarioEscapado//\$/\$}
comentarioConReferenciaEscapado=${comentarioConReferencia//\$/\$}
# Los # por \#
comentarioEscapado=${comentarioEscapado//\#/\#}
comentarioConReferenciaEscapado=${comentarioConReferencia//\#/\#}
# Los ! por \!
comentarioEscapado=${comentarioEscapado//\!/\!}
comentarioConReferenciaEscapado=${comentarioConReferencia//\!/\!}
# Los / por \/
comentarioEscapado=${comentarioEscapado//\/\}
comentarioConReferenciaEscapado=${comentarioConReferencia//\/\}
# Los ] por \]
comentarioEscapado=${comentarioEscapado//\]/\]}
comentarioConReferenciaEscapado=${comentarioConReferencia//\]/\]}
# Los * por \*
comentarioEscapado=${comentarioEscapado//\*/\*}
comentarioConReferenciaEscapado=${comentarioConReferencia//\*/\*}
# Los . por \.
comentarioEscapado=${comentarioEscapado//\./\}
comentarioConReferenciaEscapado=${comentarioConReferencia//\./\}
# Esto se podrá hacer todo en uno pero ya veremos más adelante si eso.
# Y AUN ASI SIGUE FALLANDO!?!

### BASTA YA #####
```

Figura 7: Pendiente mejora en el escado de caracteres

5.2. Expresiones regulares y atrapado de comentarios

Podríamos seguir experimentando con la expresión regular para atrapar comentarios para buscar otros *corner cases* que no hayamos tenido en cuenta.

También se ve que en el atrapado de comentarios suelo atrapar los comentarios con el espacio o tabulación de delante. Para trabajar después hago un truco quitando con sed separandolo en dos grupos el del espacio/tabulación y el resto. Podría simplemente comprobar si hay tabulación o espacio delante para quitarlo.

Pero en general repito el truco de arriba varias veces en el código, ensuciandolo y siendo más difícil de entender a primera vista.

5.3. Escapado de caracteres en el uso de sed

Como comenté, abuso bastante del uso de **sed** y tengo que escapar continuamente las variables que paso para utilizar dentro.

A lo largo del código se ven varios bloques en los que reemplazo uno a uno todos los caracteres que me han ido dando problemas. Esto se podría haber hecho de una forma más compacta.

5.4. Acotar y/o limitar el uso de sed

He utilizado sed en algunos lugares donde no eran necesarios y simplemente se podrían haber utilizado otras técnicas como **sustitución de parametros de bash**.

También conservo algunos sed que realizan reemplazos de una única línea, pero lo hago sobre todo un fichero. Cuando están en bucles y se tiene que hacer varias veces, sed itera todo el archivo para localizar esta sustitución. Se podría sustituir una única vez y salir de la comprobación o simplemente especificar la línea en vez del archivo concreto.

5.5. Agregar opciones adicionales

Esta versión ha sido revisada varias veces. En cada vez hemos encontrado diferentes casos y fallos que no contemplaba que rompían por completo el funcionamiento.

Sería interesante seguir buscando nuevos casos que nos dieran pistas de que otras funcionalidades deberíamos de implementar.

5.6. Comprobación de existencia de archivos

Uso tanto el sed como el grep sobre ficheros de traducción, pero casi en ningún momento realizo comprobaciones de si estos archivos existen. Si fallan simplemente dejo el **mensaje de error** por defecto del sed o grep.

6. Conclusiones personales

6.1. Problemas encontrados

6.1.1. Escape de caracteres

Excluyendo el tiempo dedicado en aprender el lenguaje y descubrir ciertas peculiaridades, la mayor parte de mi tiempo se ha ido en pelearme con los posibles valores que tenía que escapar para ser utilizados en sed.

6.1.2. Rellenar array con while y pipe

El problema que encuentre se debía a que el bucle while que utilizas para leer las líneas del archivo se estaban ejecutando en **subprocesos** debido al uso del **pipe**. Cuando se ejecuta en un subproceso, las modificaciones del array no se reflejarán fuera de ese subproceso, que explica por que el array no tenía nuevos elementos terminado el bucle.

```
#####
# Declarando las variables globales a usar
#####

scriptSelfName=$0 # El nombre de ESTE script para ignorarlo en búsquedas de ficheros .sh
dirPath='./' # Donde buscar los ficheros. Modificado!! Ahora trabaja directamente sobre la ruta donde esta el ESTE script.
idioma='ES' # Idioma por defecto

# Esto son arrays que uso como contenedores en las ejecuciones de algunas funciones de abajo.
# Me da bastante rabia que las funciones no puedan retornar valores fuera de los numericos y que me obligue a trabajar con variables globales.
declare -a idiomasDisponibles
declare -a ficherosScript
declare -a ficherosTraduccion
declare -a comentariosEncontrados
```

Figura 8: Variables y arrays usados de forma global

6.1.3. Declare -a dentro de una funcion

He querido sobrescribir una variable de tipo array dentro de una función, al hacerlo inicializandola con declare -a tuve problemas. Me dí cuenta que esto la convertía en una variable de tipo local de forma implícita.

6.1.4. Iterar arrays de cadenas de texto

He estado largo y tendido con el mismo problema varias veces. No se puede iterar sobre `${array[@]}` se debe iterar sobre `"${array[@]}"`. De otra forma itera directamente por caracteres, es decir saca las letras.

6.2. Sobre el uso de Chat GPT

Siendo algo tan disruptivo he decidido dedicarle una sección aparte.

Mi experiencia con **Chat GPT** e incluso otras inteligencias artificiales generativas, aquí incuyo algunas que me han comentado compañeros e incluso otras específicas enfocadas a código como **GitHub Copilot**, es más bien agrisulce.

En esta ocasión el mayor beneficio que me ha aportado ha sido el poder acelerar mi entrada al lenguaje. Me ha dado una variedad de respuestas sobre casos concretos de forma inmediata, esto me ha ayudado a encontrar diferentes aproximaciones a un problema con herramientas proporcionadas por bash que ni siquiera conocía. A esto me refería con ganar tracción con el lenguaje.

Otra ventaja innegable es poder pasarle un fragmento de código y preguntarle; **¿Donde está el problema?**. En el caso de que fueran problemas de sintaxis o problemas muy simples me ahorra mucho tiempo en encontrar problemas **"tonto"** que de otra forma me hubieran retrasado. El típico error de sintaxis en el cual gastas media tarde.

Fuera de estas dos principales bazas he encontrado más inconvenientes que ventajas.

En primer lugar, no tiene un contexto lo suficientemente específico de mi proyecto. Intenta promediar mi proyecto con lo que sea que utilice para generar su respuesta. Esto hace que me **rompa el estilo** que sigo. No solamente en un sentido estrictamente estético, si no que además me cuesta leerlo a posteriori.

A veces mezcla diferentes formas de hacer la misma cosa. A veces durante el proyecto necesito hacer cosas relativamente parecidas pero lo suficientemente diferentes para querer reescribirlas dos veces. Deberían verse relativamente parecidos, pero con ChatGPT puede generarte dos formas lo suficientemente diferentes de hacer esto que te cueste pasar de leer uno a otra. Esto te obliga a hacer un esfuerzo mental extra para entender que ocurre en el código, cuando , si mantuviera el mismo estilo, debería ser muy sencillo de llevar.

Otro aspecto negativo es que te vuelve más **perezoso y descuidado**. Bajas la guardia y, en parte, empiezas a aceptar cualquier cosa que te de. Esto puede hacer que empieces a **desvariar** en el código. Muchas veces la solución era mucho más simple, y es por ello que a lo largo de la práctica según avanzaba más y más he recurrido más a **stackoverflow** o **las practicas** para resolver dudas concretas. Esto ha resultado más efectivo cuando sabía que buscaba y sabía identificar la respuesta correcta.

Esto último me gustaría recalcarlo, si tenía dudas sobre la respuesta recibida es imposible resolverla con ChatGPT sin fiarse de su respuesta, que por supuesto puede o no ser correcta.

Al fin y al cabo el **NLP** actual solo es un juego estadístico para precedir que el carácter será el siguiente en ser escrito, un promedio.

Yo no quiero el promedio, quiero asomarme y mirar que hay en el borde.

6.3. Habilidades adquiridas

Esta práctica me ha ayudado a ganar confianza con **shell** y comandos de linux como **sed**, **grep**, **find**. Así como las **bases** y **peculiaridades** del scripting en bash.

También en menor medida me ha ayudado con **lógica** básica de programación.