* 1)
  1. **Shell Scripting:** Es la práctica de escribir secuencias de comandos o scripts que están destinados a ser ejecutados por un intérprete de comandos/Shell. Los scripts de Shell están compuestos de comandos que normalmente se ejecutarían desde una Shell de forma interactiva, pero se agrupan en un archivo para automatizar tareas o secuencias de tareas más complejas.

Tipos de tareas a las que están orientados los Scripts:

* Automatización de Tareas del Sistema, de Desarrollo, Tareas Repetitivas, de Seguridad.
* Procesamiento de Archivos y Datos.
* Gestión de Usuarios y Permisos.
* Interacción con Redes y Servidores, Bases de Datos, APIs y Servicios Web.
* Generación de Informes y Logs.
* Configuración de Ambientes de Desarrollo.

Los scripts no deben compilarse, estos son interpretados por la Shell en tiempo de ejecución, es decir, el código es leído línea por línea y ejecutado de inmediato por la Shell, las razones por las cuales no necesitan ser compilados son:

* **Interpretación en tiempo real:** El Shell los interpreta en tiempo real permitiendo una rápida iteración y prueba de código.
* **Portabilidad:** Los scripts suelen ser independientes del hardware y el sistema operativo ya que el Shell se encarga de adaptarlos a cada entorno.
* **Facilidad de Edición:** Se puede modificar un script directamente en un editor de texto y ejecutarlo sin pasar por un proceso de compilación.
* **Menor sobrecarga de Desarrollo:** Al no ser necesaria la compilación y enlazamiento del código antes de ejecutarlo, el proceso de desarrollo y prueba se vuelve más ágil.
* 2)
  + Comandos:

“echo”: Su función principal es imprimir texto o variables en la salida estándar del sistema.

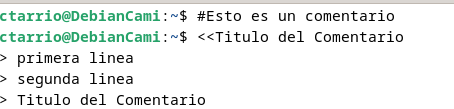
Parámetros:

* “-n”: Evita que se añada un salto de línea automático al final del texto.
* “-e”: Habilita la interpretación de escape o saltos de línea usando “\n (salto de línea) ó \t (tabulación)”.
* “-E”: Deshabilita la interpretación de escape o saltos de línea.
* Es común usar este comando con Redirecciones de Salida (> ó >>).

“read”: Se utiliza para leer entradas del usuario desde la línea de comandos directamente desde el teclado mientras se ejecuta un script, la entrada se almacena en una variable.

“read [opciones] variable”.

Parámetros:

* “-p [prompt]”: Muestra un mensaje antes de solicitar la entrada.
* “-s”: Habilita el modo silencioso, es decir, la entrada del usuario no se va a mostrar en la pantalla.
* “-[n]”: Especifica el número máximo de caracteres, ese número se tiene que especificar en n.
* “-t [n]”: Especifica un tiempo límite de espera en segundos para ingresar la entrada. Se debe especificar la cantidad de segundos en n.
* “-r”: Desactiva la interpretación de barras invertidas \.
  + Hay dos formas:
* 
  + Formas de nombrar Variables:
* Deben comenzar con una letra o un guión bajo.
* Pueden contener letras, números y guiones bajos.
* Los nombres de las variables son case sensitive.

Declaración:

* Para la declaración se utilizan las normas de nombramiento anteriores y la forma de darles valor es la siguiente: mi\_variable=valor. No se deben dejar espacios entre el signo “=”.

Referencia:

* Para hacer referencia a una variables y obtener su valor se debe utilizar el símbolo “$” seguido del nombre de la variable, por ejemplo, $mi\_variable.
* 3)
  + Comandos:

“mkdir practica-shell-script”

“cd /practica-shell-script”

“touch mostrar.sh”

“vi mostrar.sh” (Agrego todo el texto indicado)

**Texto**:

*#!/bin/bash*

*# Comentarios acerca de lo que hace el script*

*# Siempre comento mis scripts, si no hoy lo hago*

*# y mañana ya no me acuerdo de lo que quise hacer*

*echo "Introduzca su nombre y apellido:"*

*read nombre apellido*

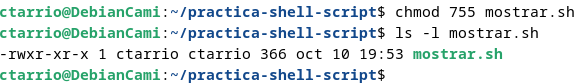
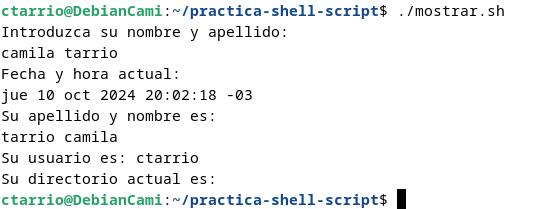
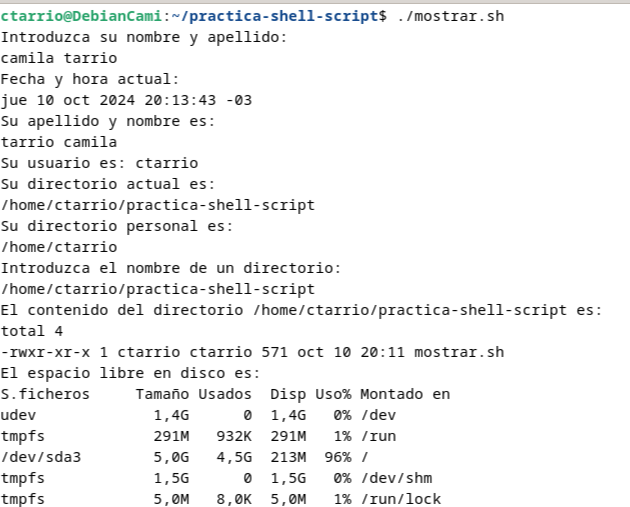
*echo "Fecha y hora actual:"*

*date*

*echo "Su apellido y nombre es: “*

*echo "$apellido $nombre" echo "Su usuario es: `whoami`"*

*echo "Su directorio actual es:"*

* + 
  + 
  + En el contexto del comando “whoami” las backquotes se utilizan para ejecutar un comando dentro de otro comando y luego sustituir la salida del comando interno en el comando externo aunque ahora se recomienda utilizar “$(comando)”.
  + 
* 4)
  + Cuando se acceden a los parámetros enviados al script al momento de su invocación se hace a través de variables especiales, estas suelen estar predefinidas y contienen la información de los argumentos proporcionados al script, estos parámetros se almacenan en las variables “$1, $2, $3, …” al ejecutar el script con argumentos se debe hacer de la siguiente manera “./script.sh parámetro\_1 parámetro\_2”.

**Variables:**

* “$#”: Contiene el número de argumentos que se pasaron al script.
* “$\*”: Representa todos los argumentos en una sola cadena de texto, esta variable no preserva los espacios en blanco entre los argumentos.
* “$?”: Contiene el código de salida del último comando ejecutado. Si el comando se ejecuta con éxito, “$? = 0”. Si hay algún error, será un valor diferente de 0. Esto es útil para verificar si un comando fue exitoso y tomar decisiones basadas en eso.
* “$HOME”: Contiene la ruta al directorio raíz del usuario actual.
* 5)
  + “exit”: Se utiliza para finalizar la ejecución de un script o programa en una Shell. Puede recibir un valor como argumento que indica el código de salida que el script o programa devolverá al sistema operativo.
* Sin argumentos: Sin argumento alguno, el comando “exit” devuelve el código de salida predeterminado, que es 0. Un código de salida igual a 0 generalmente indica que el programa se ejecutó con éxito y sin errores.
* Con argumentos: Si se proporciona un argumento, este será el código de salida que se devolverá al sistema operativo, estos códigos no deben ser negativos y generalmente se utilizan para indicar si el programa se ejecutó con éxito o si ocurrió algún tipo de error. Es común usar códigos de salida diferentes a cero para indicar diferentes tipos de errores.
* 6)
  1. “expr”: Herramienta de Shell que permite la evaluación de expresiones en scripts de Shell. Las expresiones pueden incluir:

Operadores Aritméticos:

* Suma (+): Suma dos números “expr 5 + 3”.
* Resta (-): Resta dos números “expr 8 – 2”.
* Multiplicación (\*): Multiplica dos números “expr 4 \* 3”
* División (/): Divide un número por otro “expr 10 / 2”.
* Módulo (%): Devuelve el resto de la división de dos números “expr 7 % 3”.

Operadores de Comparación:

* Igual (=): Compara si dos cadenas son iguales “expr “hola” = “hola””.
* Diferente (!=): Compara si dos cadenas son diferentes “expr “hola” != “chau””.

Operadores Lógicos:

* AND (&&): Operación lógica “y”; “expr 1 && 0”.
* OR (||): Operación lógica “o”; “expr 1 || 0”.

Operadores Relacionales:

* Menor que (<): Compara si el primer argumento es menor que el segundo “expr 5 < 10”.
* Mayor que (>): Compara si el primer argumento es mayor que el segundo “expr 8 > 3”.
* Menor o igual que (<=): Compara si el primer argumento es menor o igual que el segundo “expr 5 <= 5”.
* Mayor o igual que (>=): Compara si el primer argumento es mayor o igual al segundo “expr 10 >= 8”.

Otros operadores:

* Longitud de Cadena (length): Devuelve la longitud de una cadena “expr length hola”.
* 7)
  + “test”: Herramienta utilizada para evaluar expresiones y devolver un valor de éxito o fracaso (true o false). Normalmente se utiliza como [expresión].

Evaluación de Archivos:

* “-e archivo”: Comprueba si el archivo existe.
* “-f archivo”: Comprueba si es un archivo regular y no un directorio o dispositivo.
* “-d directorio”: Comprueba si el directorio existe.
* “-s archivo”: Comprueba si el archivo tiene un tamaño mayor que cero.
* “-r archivo”: Comprueba si el archivo tiene permisos de lectura.
* “-w archivo”: Comprueba si el archivo tiene permisos de escritura.
* “-x archivo”: Comprueba si el archivo tiene permisos de ejecución.

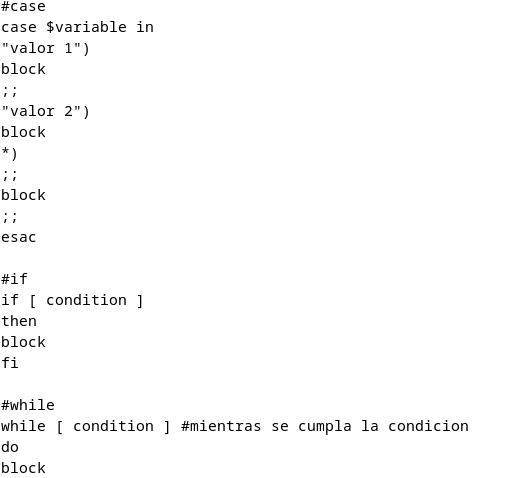
Evaluación de Cadenas de Caracteres:

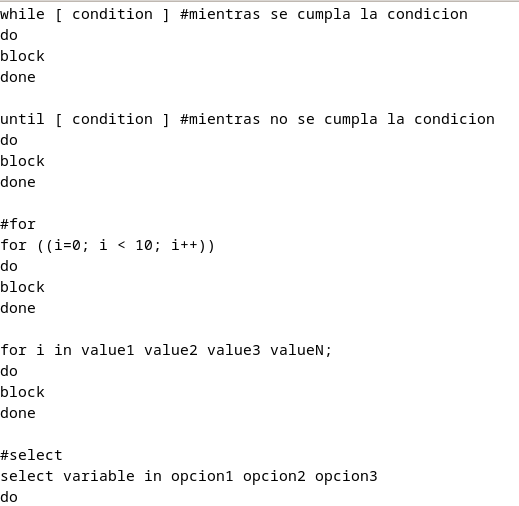
* “-z cadena”: Comprueba si la cadena es vacía (length = 0).
* “-n cadena”: Comprueba si la cadena no es vacía (length > 0).
* “cadena1 = cadena2”: Comprueba si las cadenas son iguales.
* “cadena1 != cadena2”: Comprueba si las cadenas son distintas.

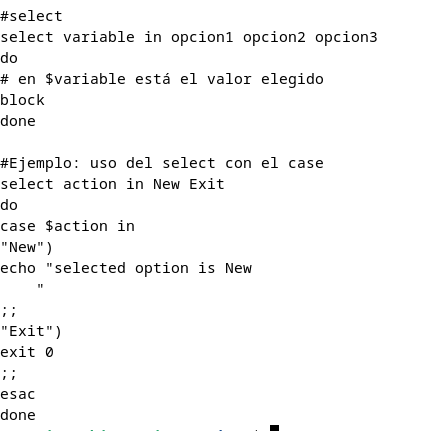
Evaluaciones Numéricas:

* “num1 –eq num2”: Comprueba si los números son iguales.
* “num1 –ne num2”: Comprueba si los números son distintos.
* “num1 –lt num2”: Comprueba si el primer número es menor que el segundo.
* “num1 –le num2”: Comprueba si el primer número es menor o igual que el segundo.
* “num1 –gt num2”: Comprueba si el primer número es mayor que el segundo.
* “num1 –ge num2”: Comprueba si el primer número es mayor o igual que el segundo.

Otros operadores:

* “! [Expresión]”: Negación.
* “expresión1 –a expresión2”: AND lógico.
* “expresión1 -0 expresión2”: OR lógico.
* 8)
  + 





* 9)
  + Break: Se utiliza para salir inmediatamente de un bucle for, while, o until antes de que se complete su iteración normal.

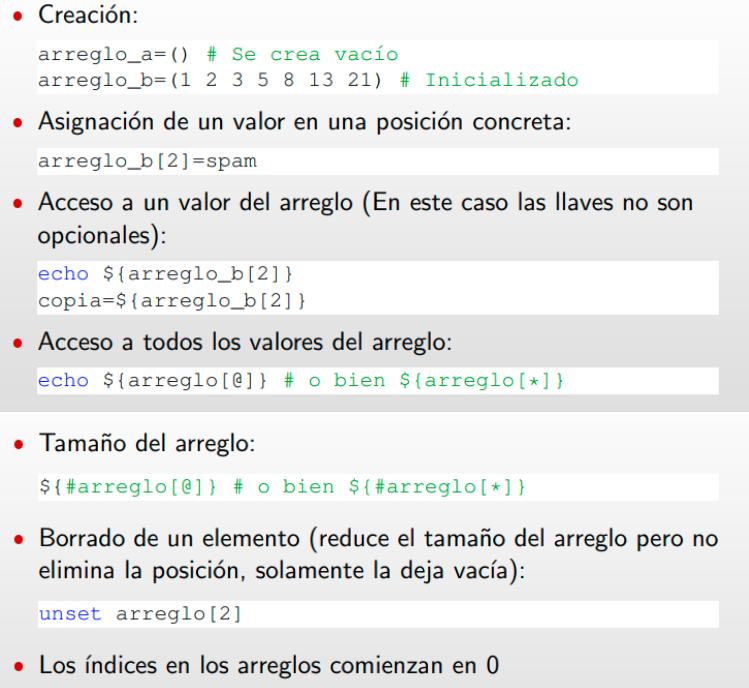
Parámetros:

* “break [n]”: “n” Es un número entero opcional que indica cuántos niveles de bucles se deben salir/saltear. Si no se proporciona valor alguno, sale del bucle actual.

Continue: Se utiliza para pasar a la siguiente iteración del bucle sin ejecutar el código que sigue a continuación en el bucle, es decir, salta la parte restante del bucle para la iteración actual y pasa a la siguiente iteración.

Parámetros:

* “continue [n]”: “n” Es un número entero opcional que indica cuántos niveles de bucles se deben saltear. Si no se proporciona valor alguno, saltea el bucle actual.
* 10)
  1. En Shell scripting las variables solo pueden ser strings y arrays. En cuanto a si Shell script es fuertemente tipado, esto no es cierto, no es necesario especificar el tipo de datos al declarar una variable y las mismas pueden almacenar diferentes tipos de datos en diferentes momentos.



* 11)
  + Si, se pueden definir funciones dentro de un script, esto nos ayuda a modularizar el comportamiento de los scripts. Se pueden declarar de 2 formas:
* function nombre { block }
* nombre() { block }

Return:

Utilizando la sentencia “return” se retorna un valor entre 0 y

255. Este valor se puede evaluar mediante la variable “$?” y

las funciones reciben argumentos en las variables “$1, $2, etc”.

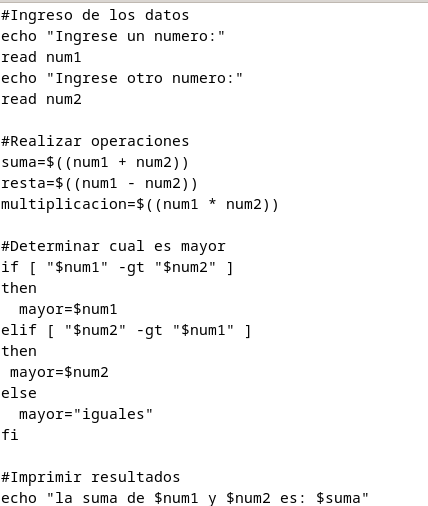
Variables:

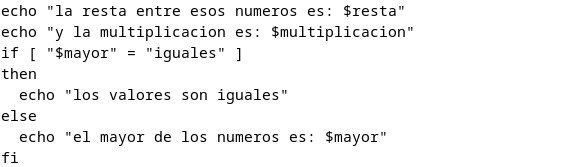
Las variables que se usarán sólo dentro de la función se definen

con la sentencia “local variable” ya que estas por defecto son globales. Las variables que no son inicializadas se reemplazan por un valor nulo o 0. Todas las variables de entorno son heredadas por los procesos hijos y para exponer una variable global a los procesos hijos se usa el comando “export VARIABLE\_GLOBAL”.

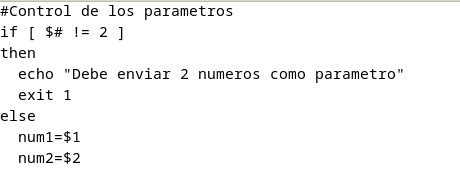
Invocación:

Para la invocación de una función en el código se debe hacer “nombre\_de\_la\_función argumento1 argumento2 argumentoN”

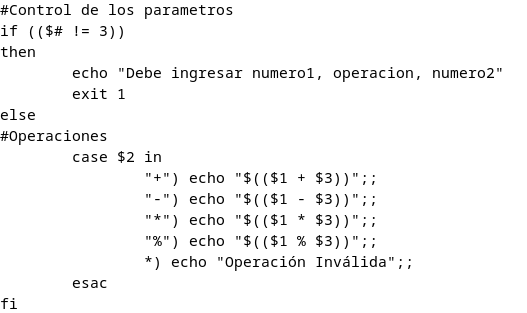
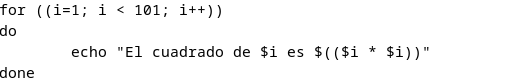
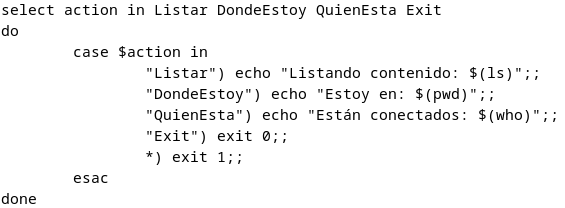
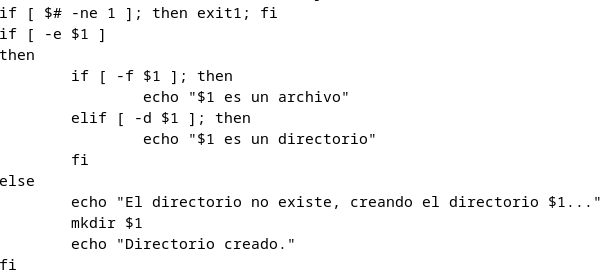
* 12)
  + 

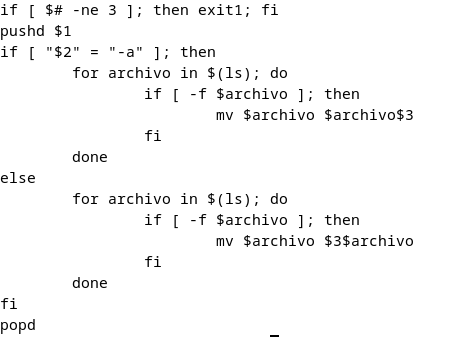


* + Cambios realizados antes de las operaciones:



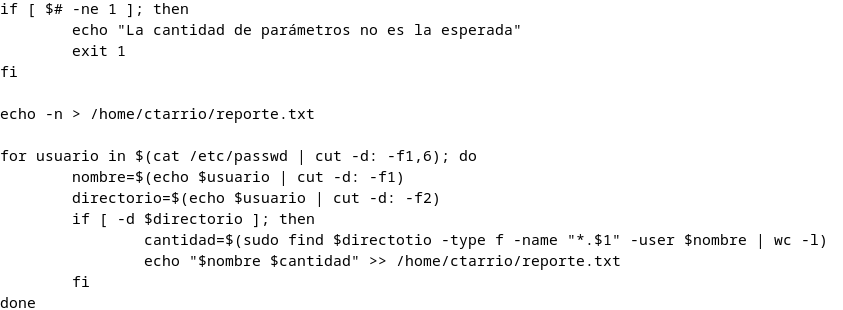
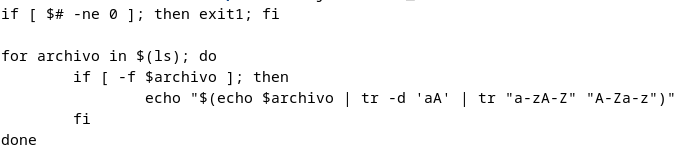
Luego se ejecuta el script enviándole como parámetro 2 números:

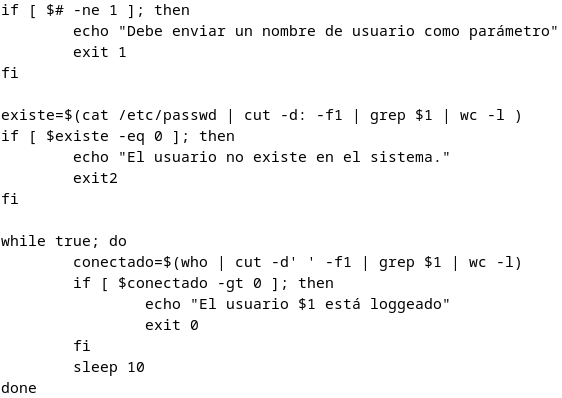
* ./Ejercicio12.b\_TP3 10 5
  + 
* 13)
  + 
  + 
  + 
* 14)
  + Comandos:
* “pushd [ruta]”: Cambia el directorio actual y guarda el directorio anterior en la pila de directorios, si no se especifica una ruta intercambia el directorio actual con el último guardado en la pila.
* “popd”: Cambia el directorio actual con el último guardado en la pila y lo retira.

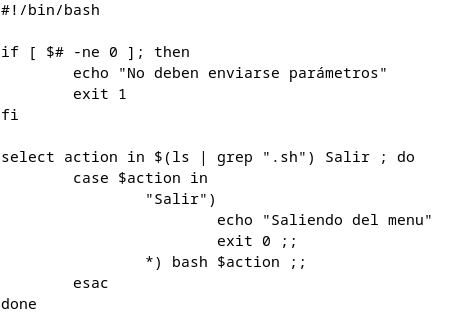
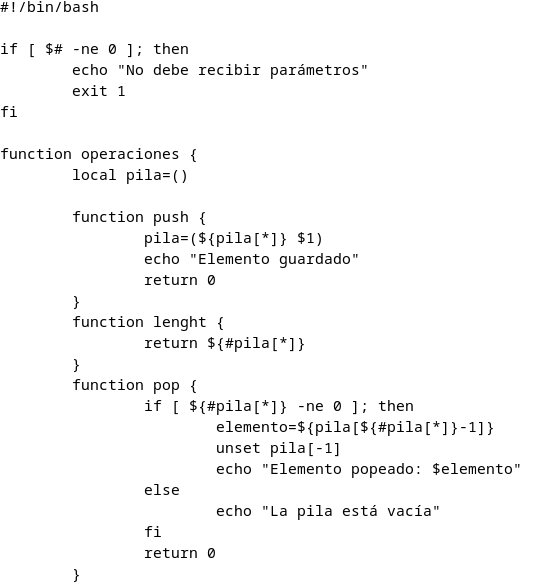


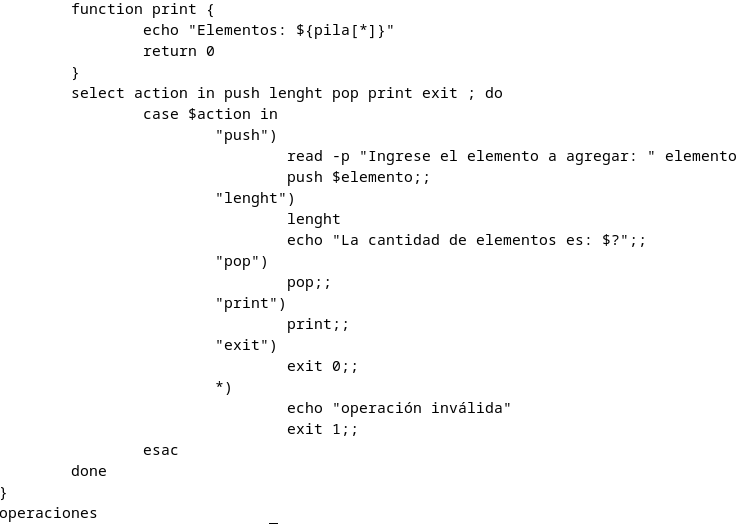
* 15)
  + “cut”: Comando que nos permite extraer secciones de líneas de texto de archivos o de la entrada estándar para luego imprimir el resultado en la salida estándar, normalmente es usado para extraer columnas específicas de un archivo de texto delimitado por caracteres, además permite especificar un determinador personalizado en lugar del tabulador “\t” que es el predeterminado.

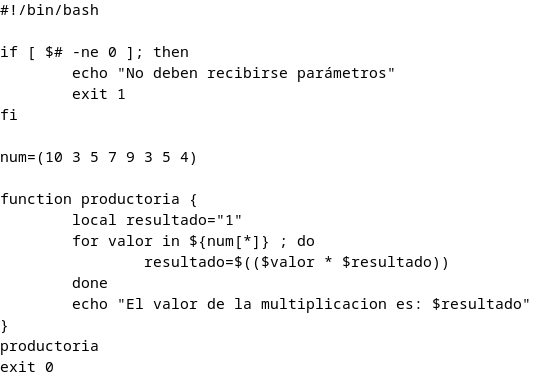
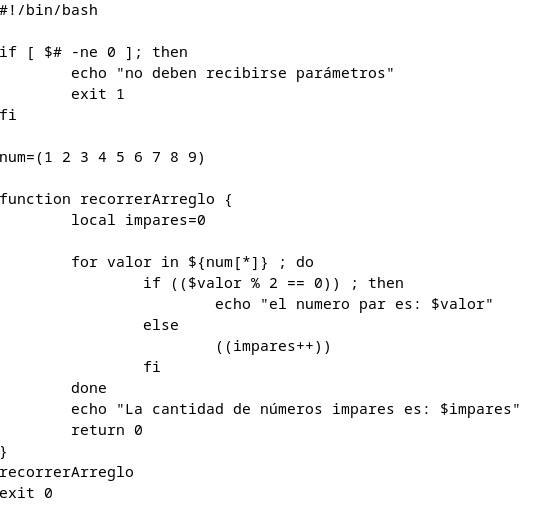
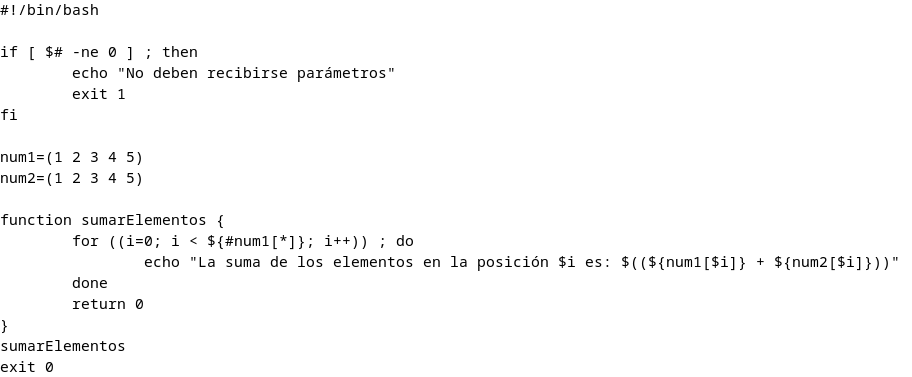
Parámetros:

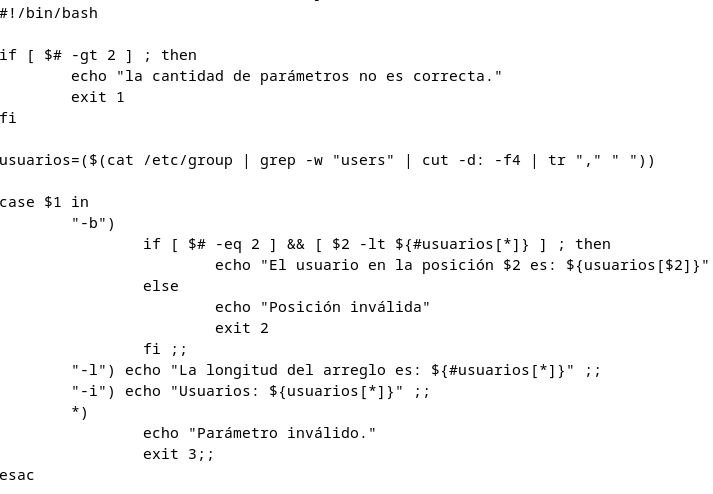
* “-d [delimitador]”: Especifica el delimitador, por defecto, el tabulador.
* “-f [num\_campo] [archivo]”: Especifica los campos que se quieren extraer.
* “-c [rango] [archivo]”: Extraer un rango de caracteres específicos de una línea de texto en lugar de columnas basadas en un delimitador.
* “-b [rango] [archivo]”: Trabaja igual que el parámetro “-c” pero en vez de caracteres usa bytes.
* 16)
  + 
* 17)
  + 
* 18)
  + Explicación de la línea cat /etc/passwd | cut -d: -f1 | grep usuario | wc -l
* **cat /etc/passwd**: Muestra el contenido del archivo /etc/passwd. Este archivo contiene información sobre los usuarios del sistema, con cada línea representando un usuario. Las diferentes partes de cada línea están separadas por dos puntos (:)
* **cut -d: -f1**: Usa cut para extraer el primer campo de cada línea del archivo /etc/passwd, delimitando los campos con el carácter :.
* **grep $1**: Busca la cadena que está almacenada en el primer argumento del script.
* **wc -l**: Cuenta el número de líneas que resultaron de la búsqueda realizada por grep. Este comando devuelve el número de coincidencias encontradas.

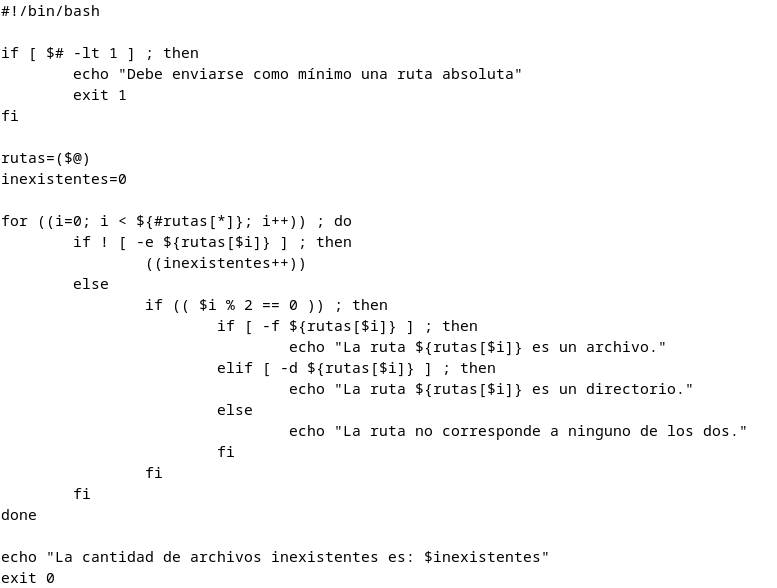


* 19)
  + 
* 20)
  + 

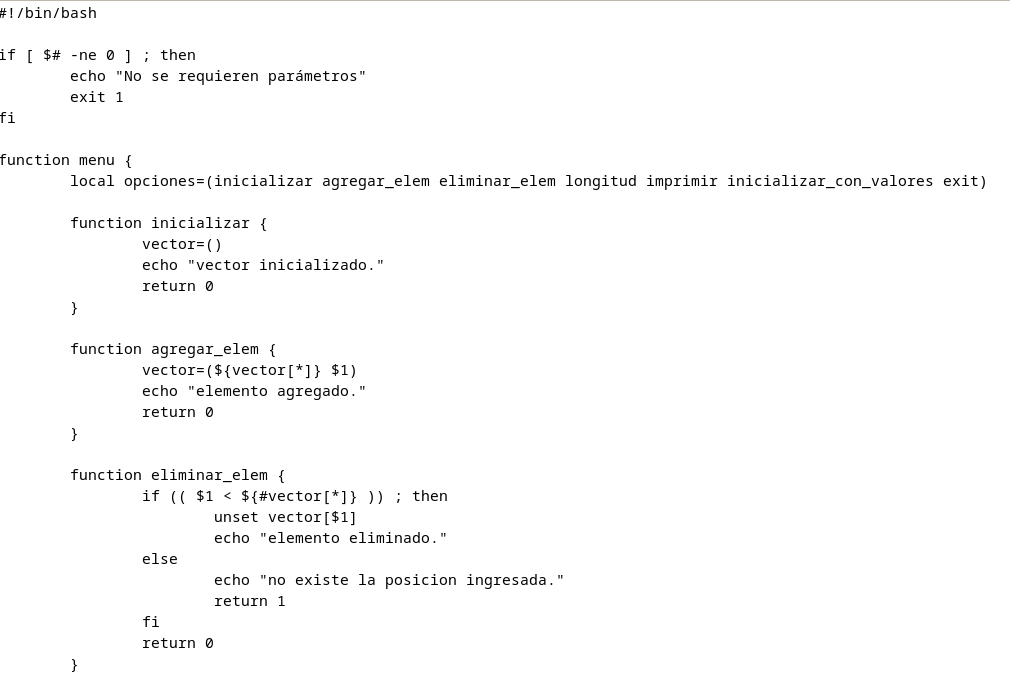


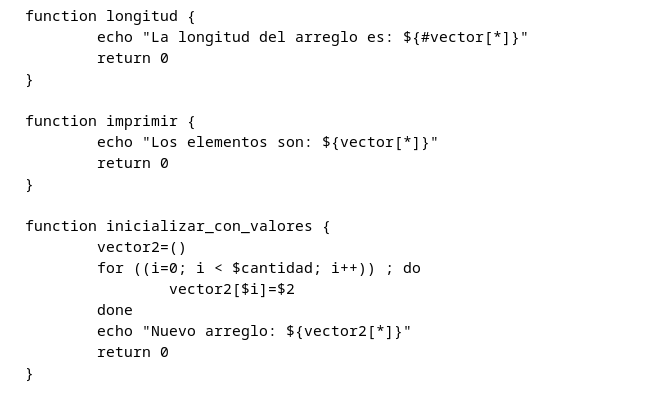
* 21)
  + Probar el código.
* 22)
  + 
* 23)
  + 
* 24)
  + 
* 25)
  + Explicación de la sentencia usuarios=($(cat /etc/group | grep -w "users" | cut -d: -f4 | tr "," " "))
* **cat /etc/group**: Muestra el contenido del archivo /etc/group, que contiene información sobre los grupos del sistema y sus usuarios.
* **grep -w "users"**: Filtra las líneas que contienen exactamente la palabra users como nombre de grupo (la opción -w asegura que coincida la palabra completa).
* **cut -d: -f4**: Divide la línea seleccionada por : y toma el cuarto campo (donde están listados los usuarios del grupo, separados por comas).
* **tr "," " "**: Reemplaza las comas por espacios para que los usuarios queden separados correctamente.
* **usuarios=($(...))**: Asigna los resultados obtenidos en un array llamado usuarios. Los nombres de los usuarios del grupo "users" estarán disponibles en este array, donde cada usuario es un elemento separado.

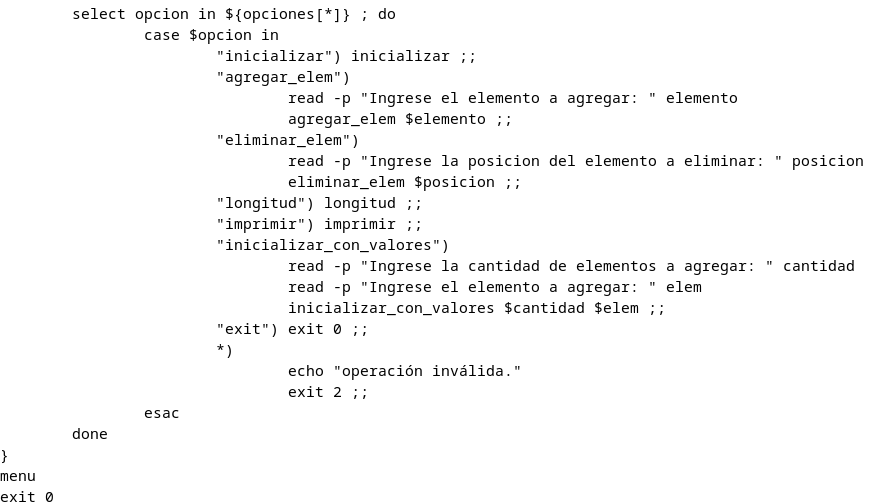


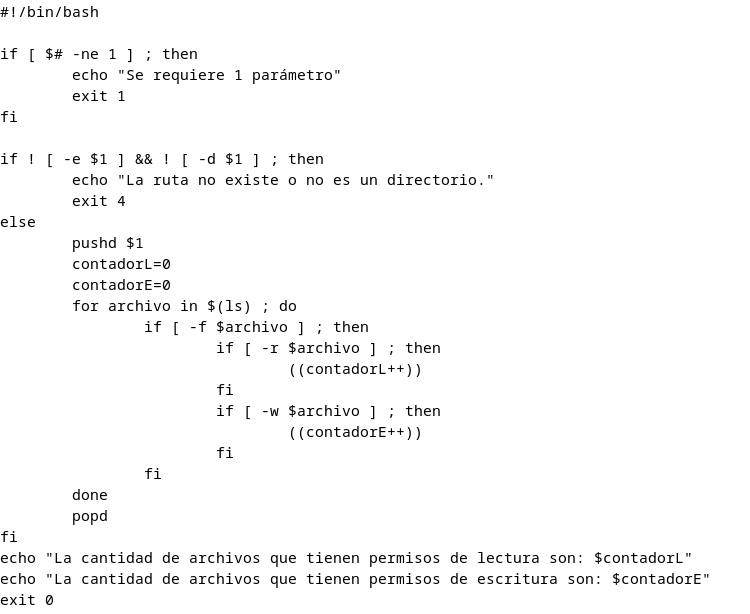
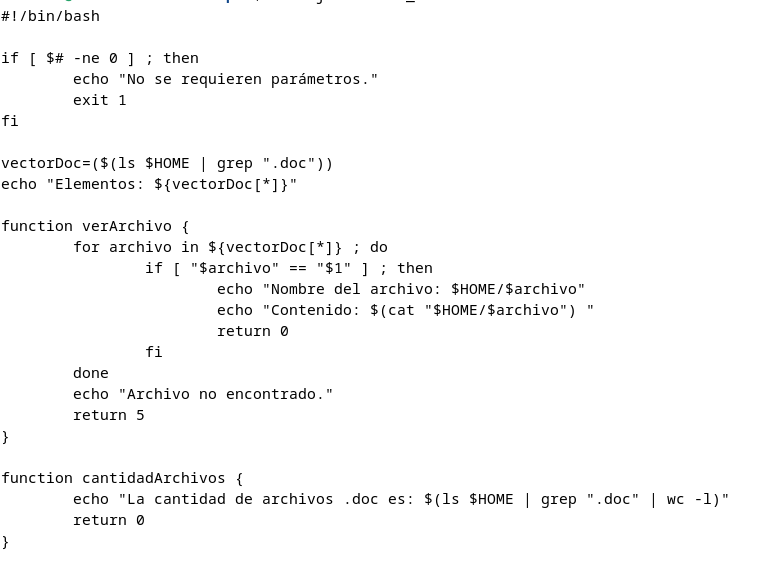
* 26)
  + 

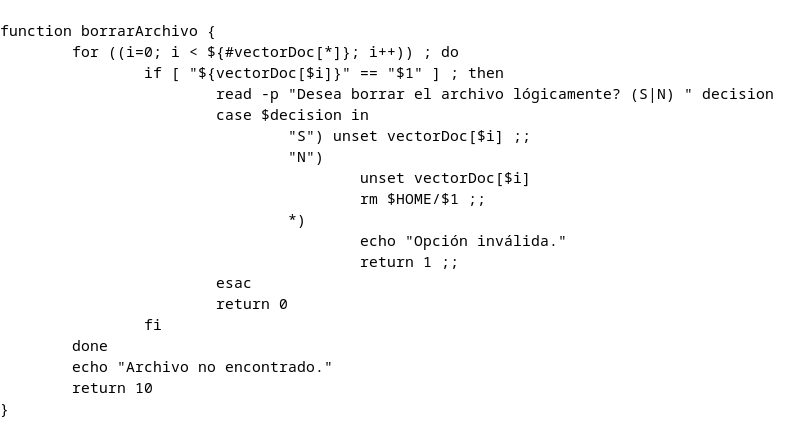
Aclaración: Cuando controlo si “i” es par o impar la condición queda “al revés” (o sea entra cuando es par) ya que el arreglo inicia en 0 pero los parámetros en 1, de esta forma va a analizar sólo los parámetros impares.

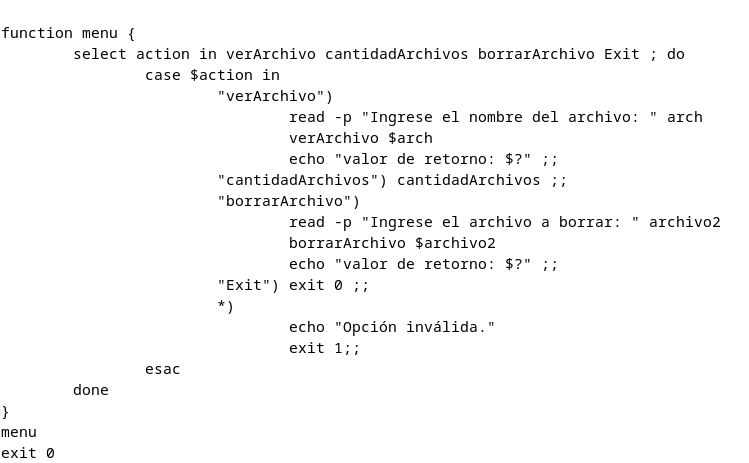
* 27)
  + 





* 28)
  + 
* 29)
  + 





* 30)