



TALLER DE TECNOLOGÍAS

OBLIGATORIO 1 – UNIVERSIDAD ORT

328226 - Manuel Pallares

143403 - Sebastian Mendez

GRUPO N1B

Tabla de contenido

**Presentación**3

Objetivo3

Integrantes3

**Estructura del proyecto**4

**Ejecución del programa**5

**Documentación del código**6

main.sh6

seguridad.sh7

configuracion.sh9

menu.sh11

consultas.sh11

algoritmos.sh14

Presentación

Objetivo

La entrega comprende la realización del primer obligatorio de la materia, en el cual se aborda el aprendizaje y uso de los principales comandos Linux. Para ello se solicita la realización de un script de Bash el cual simula una serie de acciones en un sistema Linux.

Integrantes



328226 – Manuel Pallares 143403 - Sebastián Méndez

Estructura del proyecto

El proyecto consta de varios archivos .sh, los cuales han sido nombrados según su responsabilidad en el programa.

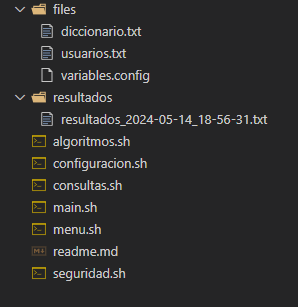
Se decidió separarlos para hacer más mantenible el código y que sea más fácil realizar cambios al estar separadas las diferentes funciones.

Archivos:

* main.sh: Es el punto de entrada a la aplicación.
* algoritmos.sh: Funciones que son utilizadas para las funcionalidades Algoritmo 1 y Algoritmo 2.
* configuración.sh: Funciones encargadas de guardar y leer las diferentes variables que serán utilizadas.
* consultas.sh: Funciones encargadas de realizar las diferentes consultas al diccionario.
* menu.sh: Funciones encargadas de mostrar el menú y su controlador correspondiente.
* seguridad.sh: Funciones encargadas de login y de interactuar con el archivo de usuarios.

Carpetas

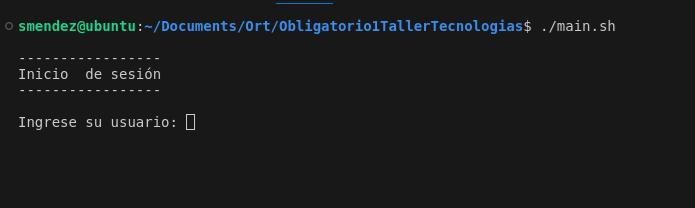
* resultados: Se guardan los resultados de las diferentes consultas realizadas sobre el diccionario.
* files: Archivos de configuración, variables y diccionario.



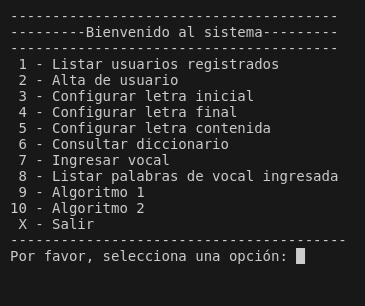
Estructura del proyecto

Ejecución del programa

Para la ejecución, mediante la consola el usuario debe de posicionarse en la raíz del proyecto, y ejecutar el script ./main.sh.



El usuario por defecto es: admin y su password: admin



Menú principal

Documentación del código

main.sh

Es el punto de entrada de la aplicación. En dicho script se solicitan las credenciales de usuario y se llama a la función “Login” que se encuentra en el archivo externo “seguridad.sh”.

A continuación, se realiza la explicación de la lógica y funcionamiento de parte de sus funciones.

# Carga de script con función auxiliar "Login"

source ./seguridad.sh

# Ejecuta un bucle solicitando el nombre de usuario mientras este sea vacío.

# **-**z es un operador de prueba que devuelve verdadero si la longitud de la

# cadena es cero, es decir, si la variable está vacía.

while [ -z "$usuario" ]; do

done

# Ejecuta un bucle solicitando el nombre de usuario mientras este sea vacío.

# -z es un operador de prueba que devuelve verdadero si la longitud de la

# cadena es cero, es decir, si la variable está vacía.

while [ -z "$password" ]; do

# Espera la entrada por teclado del password del usuario.

# -s No muestra la entrada del usuario por pantalla

# -p Muestra un mensaje enste de leer la entrada.

read -s -p "Ingrese su password:" password

done

# Llama a la función "Login" pasandole las variables anteriormente obtenidas

if Login "$usuario" "$password"; then

# Si el inicio de sesión es correcto se entra en un bucle en el cual se despliega el menú de usuario.

# Luego de seleccionada cada opción y ejecutado el código que tenga asociado, se vuelve a mostrar el menú.

while [ true ]; do

MenuPrincipal

# Se lee la opción elegida por el usuario y se le pasa # al controlador de opciones del menú.

read -p "Por favor, selecciona una opción: " opcion

ControladorOpcionesMenu $opcion

echo

done

seguridad.sh

En este script se encuentran las funcionalidades de Login, ListarUsuarios, AltaUsuario, ExisteUsuario.

A continuación, se realiza la explicación de la lógica y funcionamiento de parte de sus funciones.

# Constante donde se guarda la ruta del archivo que contiene el listado de usuarios del sistema.

ARCHIVOUSUARIOS="files/usuarios.txt"

# Constante donde se guarda la ruta del archivo que contiene las diferentes variables guardadas.

ARCHIVOVARIABLES="files/variables.config"

# Función que recibe dos parámetros: $usu y $pwd.

# Luego, itera sobre un archivo ($ARCHIVOUSUARIOS) que contiene pares de

# usuarios y contraseñas separados por dos puntos (:).

Login() {

local usu=$1

local pwd=$2

while IFS=":" read -r usuDb pwdDb; do

if [[ "$usu" == "$usuDb" && "$pwd" == "$pwdDb" ]]; then

clave="Usuario"

sed -i "s/^$clave=.\*/$clave=$usu/" "$ARCHIVOVARIABLES"

return 0

fi

done <$ARCHIVOUSUARIOS

return 1

}

# La función ListarUsuarios lee el archivo de usuarios y contraseñas,

# luego imprime solo los nombres de usuario, uno por línea.

ListarUsuarios() {

awk -F ':' '{print $1}' $ARCHIVOUSUARIOS

read -p "Presiona Enter para continuar..."

MenuPrincipal

}

# AltaUsuario, se encarga de agregar un nuevo usuario al archivo de usuarios. Se pide nombre de usuario y

# se revisa que este ya no exista. En caso de que ya exista se vuelve a pedir los datos nuevamente.

# Se revisa también que se haya ingresado un password. En caso de que se cumplan las condiciones, se

# redirecciona la salida de los datos ingresados al archivo que contiene el listado de usuarios

# los cuales son anexados al final.

AltaUsuario() {

local password

local usuario

while [ -z "$usuario" ]; do

read -p "Ingrese nombre del nuevo usuario : " usuario

if ExisteUsuario "$usuario"; then

echo -e "\033[31mEl nombre '$usuario' ya se encuentra registrado\033[0m"

read -p "Presiona Enter para continuar..."

AltaUsuario

else

while [ -z "$password" ]; do

read -s -p "Ingrese su password:" password

if ! [ "$password" ]; then

echo -e "\033[31mEl password del usuario no puede quedar en blanco\033[0m"

fi

done

echo "$usuario:$password" >>"$ARCHIVOUSUARIOS"

echo ""

echo "Usuario agregado correctamente"

echo ""

read -p "Presiona Enter para continuar..."

MenuPrincipal

fi

done

}

# ExisteUsuario usa el comando grep -q busca la cadena pasada por parámetro al comienzo de cada línea (^) # seguida de : en el archivo archivo de usuarios.

# El argumento -q hace que grep funcione en modo silencioso, lo que significa que no imprimirá ninguna

# salida en la consola.

# Se establece su código de salida como 0 si encuentra una coincidencia y como 1 si no la encuentra.

ExisteUsuario() {

local usuario=$1

if grep -q "^$usuario:" "$ARCHIVOUSUARIOS"; then

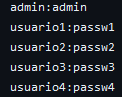
return 0

else

return 1

fi

}



Formato de archivo de usuarios

configuración.sh

Este script se encarga de guardar y leer las diferentes variables que se usaran para realizar consultas al diccionario.

# Constante donde se guarda la ruta del archivo que se usará para guardar las variables

ARCHIVOVARIABLES="files/variables.config"

# Función que se encarga de leer la configuración

# Se le debe de pasar la clave del valor a recuperar.

LeerConfig() {

# Verificar si se proporcionó un nombre de clave como argumento

if [ $# -ne 1 ]; then

echo "Uso: LeerConfig <clave>"

return 1

fi

local clave="$1"

# Verificar si el archivo de configuración existe

if [ ! -f "$ARCHIVOVARIABLES" ]; then

echo "El archivo de configuración $ARCHIVOVARIABLES no existe."

return 1

fi

# Buscar la clave en el archivo de configuración y devolver su valor

local valor=$(grep "^$clave=" "$ARCHIVOVARIABLES" | cut -d '=' -f 2)

if [ -z "$valor" ]; then

echo "No se encontró el valor para la clave '$clave' en el archivo de configuración."

return 1

fi

echo "$valor"

}

# Función encargada de guardar la letra en el archivo de configuración

# Se le pasa como parámetro si es Inicio, Fin, Contenida o Vocal.

GuardarLetra() {

#variables locales: clave, que almacena el nombre de la variable a modificar o agregar

#en el archivo de variables, y valor, que almacenará el valor ingresado por el usuario.

local clave=$1

local valor

# Utiliza un bucle while para asegurarse de que el usuario ingrese un valor para la letra de inicio.

# Si el usuario no ingresa nada, muestra un mensaje de error en rojo.

while [ -z "$valor" ]; do

read -p "Ingrese el valor de la letra ${clave}:" valor

if ! [ "$valor" ]; then

echo ""

echo -e "\033[31mEl valor de la letra ${clave} no puede quedar en blanco\033[0m"

fi

# Si el parámetro clave es vocal reviso el dato

# que me pasen sea una vocal usando una regex.

if [[ "${clave}" == "Vocal" ]]; then

valor\_minuscula=$(echo "$valor" | tr "[:upper:]" "[:lower:]")

regex\_vocal="[aeiou]"

if [[ ! "$valor\_minuscula" =~ $regex\_vocal ]]; then

echo ""

echo -e "\033[31mValores permitidos (a-e-i-o-u)\033[0m"

valor=""

fi

else

# Luego, verifica si el valor ingresado contiene caracteres que no sean letras del alfabeto.

# Si es así, muestra un mensaje de error en rojo y limpia el valor para que el usuario tenga que

# ingresar uno nuevo.

if [[ "${valor}" =~ [^a-zA-Z] ]]; then

echo ""

echo -e "\033[31mValores permitidos (Aa-Zz)\033[0m"

valor=""

fi

fi

done

# Comprobar si la variable ya existe en el archivo

if grep -q "^$clave=" "$ARCHIVOVARIABLES"; then

# La variable existe, modificar su valor

sed -i "s/^$clave=.\*/$clave=${valor:0:1}/" "$ARCHIVOVARIABLES"

echo ""

echo "La letra ${clave} ha sido modificada con el valor '$valor'."

else

# La variable no existe, agregarla al final del archivo

echo "$clave=${valor:0:1}" >>"$ARCHIVOVARIABLES"

echo ""

echo "La letra ${clave} ha sido agregada con el valor '$valor'."

fi

echo ""

read -p "Presiona Enter para continuar..."

MenuPrincipal

}

menu.sh

Este script contiene la lógica que se usaran para desplegar el menú de usuario y su controlador correspondiente.

# Función que despliega en pantalla las diferentes opciones que se le permiten realizar al usuario

MenuPrincipal() {

….

}

# Función que se encargara de capturar la opción seleccionada por el usuario.

ControladorOpcionesMenu() {

…

}

consultas.sh

Este script contiene las funciones encargadas de realizar las consultas al diccionario.

# Constante donde se guarda la ruta del archivo que contiene el diccionario suministrado.

ARCHIVODICCIONARIO="files/diccionario.txt"

# Función que se encarga de obtener las palabras del diccionario que únicamente contengan la

# vocal que previamente se configuró

ConsultarVocal() {

# Creacion de archivo que se guarda con los resultados de la busqueda.

# Ej: resultados\_vocal\_YYYY-mm-dd\_HH-MM-ss

fechaHora=$(date +"%Y-%m-%d\_%H-%M-%S")

archivoSalida="resultados/resultados\_vocal\_${fechaHora}.txt"

# Crea archivo vacío con el nombre almacenado en la variable

touch "$archivoSalida"

# Recupera la vocal guardada en el archivo de configuración con la función que se encuentra

# en el script “configuración.sh”

vocal="$(LeerConfig "Vocal")"

regex\_vocal="^aeiou"

# Usar while read para leer el archivo línea por línea

while IFS= read -r palabra; do

# La expresión regular ^([^aeiou]\*(${vocal}|${vocal^^})[^aeiou]\*)+ se utiliza para verificar si la

# palabra contiene la vocal dada ($vocal en minúsculas o mayúsculas) rodeada únicamente por

# consonantes o al inicio/final de la palabra, permitiendo que la vocal aparezca una o más veces en la

# palabra.

# ^: indica el inicio de la cadena.

# (...)+: representa una o más ocurrencias del grupo entre paréntesis.

# [^aeiou]\*: representa cero o más caracteres que no sean vocales (consonantes).

# (${vocal}|${vocal^^}): representa la vocal dada ($vocal en minúsculas o mayúsculas). Por ejemplo, si la vocal es 'a', esto se traduce a (a|A), permitiendo que 'a' o 'A' aparezcan.

# [^aeiouA]\*: representa cero o más caracteres que no sean vocales (consonantes), después de la vocal.

if [[ "$palabra" =~ ^([$regex\_vocal]\*(${vocal}|${vocal^^})[$regex\_vocal]\*)+$ ]]; then

echo "$palabra"

echo "$palabra" >>"$archivoSalida"

fi

done <"$ARCHIVODICCIONARIO"

echo "Busqueda terminada"

read -p "Presiona Enter para continuar..."

}

# Función que se encarga de obtener las palabras del diccionario que contengan la LetraInicial,

# LetraFinal y LetraContenida que previamente se configuró

ConsultarDiccionario() {

# Recupera las variables guardadas en el archivo de configuración

letraInicio="$(LeerConfig "Inicio")"

letraFin="$(LeerConfig "Fin")"

letraContenida="$(LeerConfig "Contenida")"

# Construir la expresión regular para buscar palabras

# Empieza con "letraInicio"

# Termina con "LetraFin"

# En medio tiene "LetraContenida"

regex="^${letraInicio}.\*${letraContenida}.\*${letraFin}$"

# wc (Word Count) cuenta la cantidad de palabras, el parámetro -l hace que cuenta la cantidad de líneas

cantidadPalabrasEnDiccionario=$(wc -l <"$ARCHIVODICCIONARIO")

cantidadPalabrasEnDiccionario=$((cantidadPalabrasEnDiccionario + 1))

# usamos grep para buscar las palabras que cumplen con la expresión regular

palabrasEncontradas=$(grep -E "$regex" "$ARCHIVODICCIONARIO")

# usamos if -z para verificar si se encontraron palabras

if [ -z "$palabrasEncontradas" ]; then

cantidadPalabrasEncontradas=0

else

# usamos grep -c para contar las líneas no vacías

cantidadPalabrasEncontradas=$(echo "$palabrasEncontradas" | grep -c .)

fi

porcentajeAciertos=$(echo "scale=2; $cantidadPalabrasEncontradas / $cantidadPalabrasEnDiccionario \* 100" | bc)

echo "Cantidad de palabras encontradas: $cantidadPalabrasEncontradas"

echo "Total de palabras en diccionario: $cantidadPalabrasEnDiccionario"

echo "Porcentaje de palabras encontradas: $porcentajeAciertos"

echo "Palabras encontradas:"

echo "$palabrasEncontradas"

# Creacion de archivo que se guarda con los resultados de la búsqueda.

# Ej: resultados\_YYYY-mm-dd\_HH-MM-ss

fechaHora=$(date +"%Y-%m-%d\_%H-%M-%S")

archivoSalida="resultados/resultados\_${fechaHora}.txt"

touch "$archivoSalida"

# Redirige las palabras encontradads hacia un archivo especificado por la variable.

echo "$palabrasEncontradas" >>"$archivoSalida"

# Fecha de ejecutado el reporte

echo "" >>"$archivoSalida"

fecha="Fecha ejecución reporte: $(date +'%d/%m/%Y %H:%M:%S')"

echo "$fecha"

# Agrega la fecha al archivo de salida.

echo "$fecha" >>"$archivoSalida"

# Cantidad de palabras encontradas

echo "Cantidad de palabras encontradas: ${cantidadPalabrasEncontradas}" >>"$archivoSalida"

# Cantidad de palabras totales

echo "Cantidad de palabras totales: ${cantidadPalabrasEnDiccionario}" >>"$archivoSalida"

# Porcentajes que cumplen lo pedido

echo "Porcentaje aciertos: ${porcentajeAciertos}%" >>"$archivoSalida"

# Usuario logueado

usuario="$(LeerConfig "Usuario")"

usuarioLogueado="Usuario logueado: ${usuario}"

echo "$usuarioLogueado"

echo "$usuarioLogueado" >>"$archivoSalida"

read -p "Presiona Enter para continuar..."

}

algoritmos.sh

Este script contiene la lógica de las funciones solicitadas para obtener si una palabra a palíndromo o no, y para la solicitud de ingreso de datos.

# Función que devuelve el promedio de los datos ingresados, el menor y mayor dato ingresado

Algoritmo1() {

echo ""

read -p "Ingrese la cantidad de datos que desea ingresar: " cantidadDatos

# Verificar si la cantidad ingresada es un número positivo

if ! [[ "$cantidadDatos" =~ ^[1-9][0-9]\*$ ]]; then

echo "Debe ingresar un número entero positivo."

read -p "Presiona Enter para continuar..."

Algoritmo1

else

# Inicializar variables

menor=

mayor=

suma=0

promedio=0

# Leer los datos ingresados por el usuario y calcular el menor y el mayor

for ((i = 1; i <= "$cantidadDatos"; i++)); do

read -p "Ingrese el dato $i: " dato

# Verificar si el dato ingresado es un número entero

if ! [[ $dato =~ ^[0-9]+$ ]]; then

echo "Debe ingresar un número entero."

read -p "Presiona Enter para continuar..."

Algoritmo1

fi

# Actualizar el menor y el mayor dato

if [ -z "$menor" ] || [ "$dato" -lt "$menor" ]; then

menor=$dato

fi

if [ -z "$mayor" ] || [ "$dato" -gt "$mayor" ]; then

mayor=$dato

fi

suma=$((suma + dato))

done

# Mostrar los resultados

echo ""

promedio=$(echo "scale=2; $suma / $cantidadDatos" | bc)

echo "Promedio de los datos ingresados: ${promedio}"

echo "Menor dato ingresado: ${menor}"

echo "Mayor dato ingresado: ${mayor}"

echo ""

read -p "Presiona Enter para continuar..."

MenuPrincipal

fi

}

# Función que devuelve si la palabra que se le pasa como parámetro es un palíndromo.

Palindromo() {

local palabra=""

local palabrareversa=""

read -p "Ingrese la palabra: " palabra

local largo=${#palabra}

# Obtener el reverso de la palabra

for ((i = $largo - 1; i >= 0; i--)); do

palabrareversa="$palabrareversa${palabra:$i:1}"

done

# Comparar la palabra original con su reverso

if [ "$palabra" == "$palabrareversa" ]; then

echo "La palabra \"$palabra\" es un palíndromo."

else

echo "La palabra \"$palabra\" no es un palíndromo."

fi

echo ""

read -p "Presiona Enter para continuar..."

MenuPrincipal

}