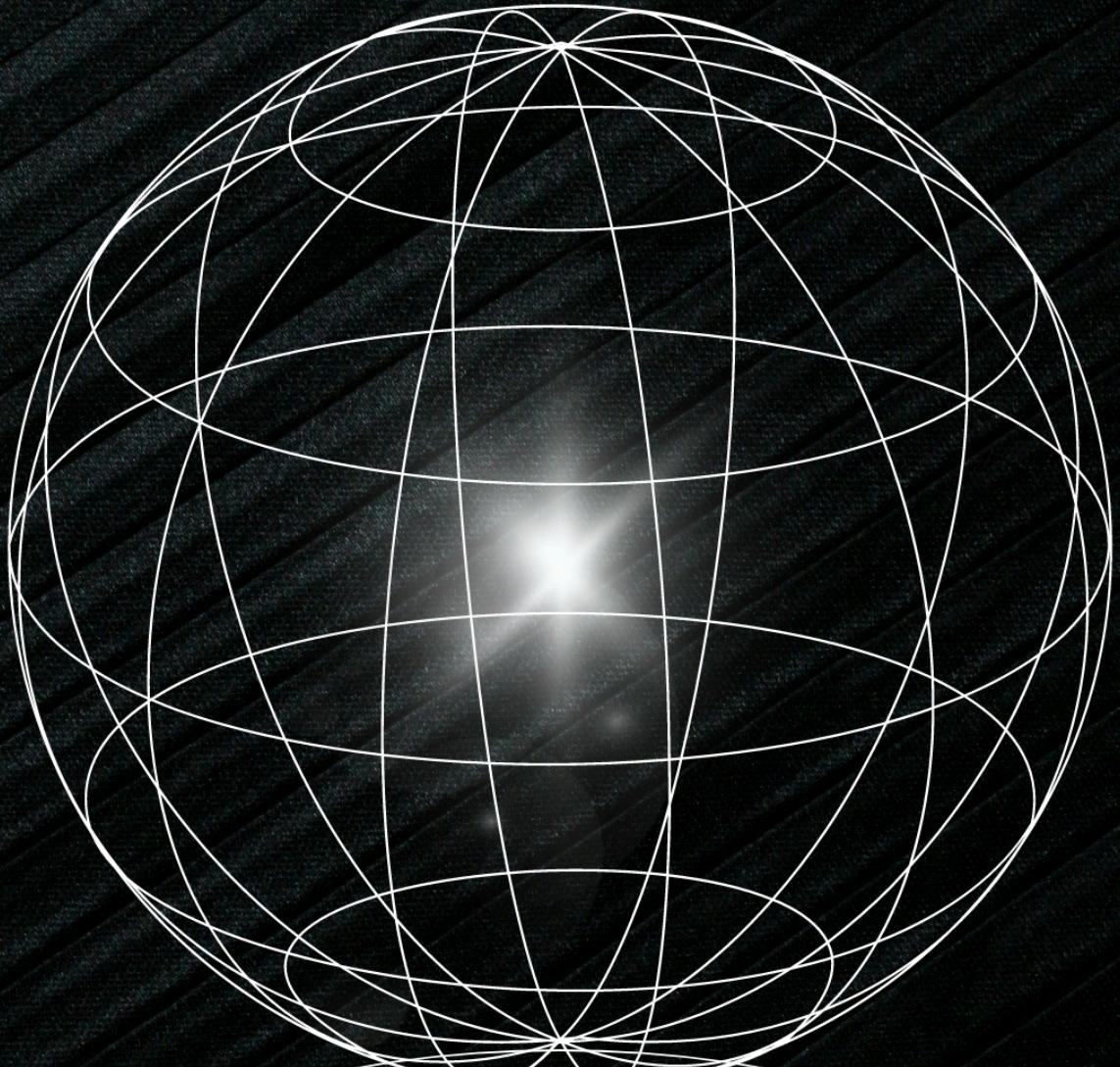




UTEC

Profesor Israel Bellizzi

eboratónio



Scripting

Salvador
Vanoli

Administración
de
Infraestructura

26/09/2024

Índice de contenido

Índice de contenido.....	1
Introducción.....	2
Datos del Shell utilizado.....	2
Breves conceptualizaciones.....	3
Bash.....	3
Bash Scripting.....	3
Funcionalidad del Bash Scripting.....	3
Comando awk.....	3
Comando column.....	3
Comando substr().....	3
Consigna.....	4
Desarrollo de la consigna.....	5
Testeo de todos los casos.....	14
Errores por malos parámetros al script.....	15
-ayuda.....	15
Errores.....	15
Resultado esperado.....	15
-s.....	15
Errores.....	15
Resultado esperado.....	16
-g.....	16
Errores.....	16
Resultado esperado.....	16
-p.....	17
Errores.....	17
Resultado esperado.....	17
-x.....	17
Errores.....	17
Resultado esperado.....	18
Log de ideas, cambios y decisiones.....	19
Función para mostrar ayuda.....	19
Función para manejar el caso -s.....	20
Funciones para manejar los casos -g y -p.....	21
Función para manejar el caso -x.....	22
Otros cambios o ideas generales.....	23
Metodología de desarrollo.....	24
Enlace al vídeo y al script.....	25
Fuentes.....	25

Introducción

En este documento se desarrollará la consigna de laboratorio de la materia Administración de Infraestructuras perteneciente a la carrera Tecnólogo de Informática en la Universidad Tecnológica (UTEC) sede San José, año 2024.

Se brindará una breve conceptualización de scripting, así como un panorama general de su utilidad en el mundo IT. A su vez, se desarrollará una tarea práctica donde se buscará elaborar un script que abarque múltiples de los temas instruidos en la materia, siendo este eficiente a la vez que legible y funcional.

Todas las opciones contempladas a lo largo del desarrollo de la consigna quedarán explícitas en el Log, para facilitar el seguimiento del estudiante.

Ya por último, habrá un enlace a un vídeo alojado en la plataforma Drive de Google donde se explicará de manera detallada y metódica la funcionalidad del script para su entendimiento.

Datos del Shell utilizado

Nombre: MINGW64

Versión: 13.2.0

Sistema Operativo: Windows 10 Home

```
gcc (Rev1, Built by MSYS2 project) 13.2.0
Copyright (C) 2023 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Breves conceptualizaciones

Bash

Es un intérprete de comandos para sistemas operativos que se basan en Unix, como lo puede ser Linux. Es una mejora a Bourne Shell (sh), ofreciendo características adicionales como el historial de comandos, el alias y funciones de scripting mejoradas.

Bash Scripting

Refiere a la creación de scripts en lenguaje Bash. Estos son archivos de texto que contienen una serie de comandos de Bash, ejecutados secuencialmente.

Funcionalidad del Bash Scripting

Se utiliza para automatizar comandos, manipular archivos y directorios, controlar el flujo de ejecución mediante estructuras condicionales y bucles así como para gestionar procesos del sistema. Esto facilita la personalización y administración de entornos de trabajo en sistemas Unix/Linux.

Comando awk

Este fue el único comando del que tuve que investigar, en especial para formatear bits a otras unidades más grandes. El comando awk es una herramienta para procesar texto en sistemas Unix y Linux, utilizada para analizar y manipular datos en archivos o cadenas de texto.

Es un lenguaje de programación especializado en el procesamiento de patrones y generación de informes. Opera en líneas y campos de texto, lo que permite a los usuarios especificar patrones y acciones para procesar cada línea de entrada.

Comando column

Este comando sirve para mostrar una cadena de texto en formato de columnas. Se tiene que combinar junto a una estructura de encabezados con el siguiente formato: “Encabezado1\tEncabezado2\t...EncabezadoN”, y al agregarlo a una cadena de texto a través de lo que puede ser, por ejemplo, un pipe, ordena cada columna de esta cadena en base a los encabezados proporcionados (column -t).

Comando substr()

Crea una subcadena a partir de otra. El formato es substr(cadena, inicio, fin), lo usé para eliminar el guión de los permisos al usar el substr que inicia en el 2do carácter.

Consigna

Realizar un script en Bash con las siguientes características:

1. La fuente de información será el comando "ls".
2. El nombre del script será: "AI_2024_list.sh"
3. El script aceptará una serie de parámetros posibles, que se listarán a continuación:
4. El parámetro "-s", servirá para especificar un archivo (ya sea usando ruta absoluta o relativa).

La sintaxis será, por ejemplo:

```
./AI_2024_list.sh -s archivo.txt
./AI_2024_list.sh -s /home/usuario/archivo.txt
```

En este caso se listará la siguiente información del archivo solicitado:

Nombre: archivo.txt

Tamaño: 148 kb

Dueño: usuario

Permisos: rwxrw-rw

5. En caso de que el parámetro sea "-g", se deberán listar los 3 archivos más grandes que contiene la carpeta. Ejemplos:

```
./AI_2023_list.sh -g
./AI_2024_list.sh -g /home/usuario/prueba
```

6. Si el parámetro es “-p”, los 3 archivos más pequeños. En ambos casos, la información a listar será del formato:

Tamaño	Nombre	Dueño	Permisos
148 kb	archivo.txt	usuario	rwxrw-rw
1092 kb	arch_la	usuario	rwxr--r-x

7. Si el parámetro es “-x”, mostrará solo los archivos ejecutables. El parámetro “-x” puede ser combinado con “-g” o “-p”.

8. En todos estos casos, si no se especifica ruta absoluta luego de los parámetros, el script ejecutara las acciones sobre la carpeta donde se aloja.

Realice todos los controles que considere necesario, para asegurarse que la cantidad y calidad de los parámetros sean los correctos que satisfaga las anteriores necesidades.

Incluya comentarios en el código, esto es obligatorio.

Desarrollo de la consigna

El script realizado estará disponible para descargar al final del documento.

```
#!/bin/bash

# Laboratorio Administración de Infraestructuras 2024
# Alumno: Salvador Vanoli

mostrar_ayuda() {
    echo
    "
-----
-----"
    echo
    echo "La sintaxis de ejecución del comando debe ser la siguiente:"
    echo
    echo "./AI_2024_list.sh [PARÁMETRO] [RUTA]"
    echo
    echo "En [PARÁMETRO] EXISTEN LAS SIGUIENTES OPCIONES:"
    echo
    echo "COMANDO -s"
    echo "Despliega la información sobre un archivo, siendo esta su nombre, tamaño, dueño y permisos"
    echo
    echo "En [RUTA] se debe incluir una ruta relativa o absoluta obligatoriamente"
    echo
    echo "COMANDO -g"
    echo "Permite ver información sobre los tres archivos más grandes del directorio especificado."
    echo
    echo "En [RUTA] se debe incluir una ruta absoluta o relativa a un directorio. Si no se incluye, [RUTA] será el directorio actual"
    echo
    echo "COMANDO -p"
```

```

echo "Permite ver información sobre los tres archivos más pequeños del
directorio especificado."
echo "En [RUTA] se debe incluir una ruta absoluta o relativa a un directorio. Si
no se incluye, [RUTA] será el directorio actual"
echo
echo "COMANDO -x"
echo "Mostrará información sólo de los archivos ejecutables del directorio
especificado."
echo "En [RUTA] se debe incluir una ruta absoluta o relativa a un directorio. Si
no se incluye, [RUTA] será el directorio actual"
echo "Puede combinarse junto a los parámetros -g o -p introducidos posterior a
-x, pero no con ambos a la vez."
echo
echo
"-----
-----
"
}

caso_s() {

    #Guardo la ruta pasada por parámetro como variable local dentro de la función
    local ruta=$1

    #Reviso que la ruta especificada exista
    if [ ! -e $ruta ]; then
        echo "ERROR: la ruta especificada no existe"
        exit 1
    #Reviso que la ruta especificada pertenezca a un archivo
    elif [ ! -f $ruta ]; then
        echo "ERROR: la ruta especificada no pertenece a un archivo"
        exit 1
    fi

    #Guardo el nombre del archivo en una variable local, uso basename para
    prescindir de la ruta y que guarde SOLO el nombre del archivo
    local nombre=$(basename $ruta)

    #Hago (ls -lh $ruta) para obtener la información del archivo pasado por
    parámetro, y luego utilizo awk para obtener solo las columnas
}

```

```
#con la información que me interesa y formatearlas. Esto lo guardo en la
variable local info
```

```
local info=$(ls -lh $ruta | awk '{print "Tamaño: "\$5, "\nDueño: "\$3,
"\nPermisos: "substr(\$1, 2)}')
```

```
#Hago echo -e para que los saltos de línea (\n) de la variable local info se
representen correctamente
```

```
echo "Nombre: $nombre"
echo -e "$info";
}
```

```
#Encapsulo la lógica para revisar que una ruta pasada por parámetro exista y sea un
directorío para poder usarla en las funciones -g, -p y -x
```

```
revisar_ruta_valida_dir() {
```

```
#Guardo la ruta pasada por parámetro como variable local dentro de la función
local ruta=$1
```

```
#Reviso que la ruta especificada exista
```

```
if [ ! -e $ruta ]; then
    echo "ERROR: la ruta especificada no existe"
    exit 1
```

```
#Reviso que la ruta especificada sea un directorio
```

```
elif [ ! -d $ruta ]; then
    echo "ERROR: la ruta especificada no pertenece a un directorio"
    exit 1
```

```
fi
```

```
}
```

```
#Encapsulo la lógica para cargar un string con archivos grandes, para así poder usarla
en las funciones -g y -x
```

```
cargar_string_archivos_grandes() {
```

```
#Guardo la ruta y la referencia al string pasado por parámetro como variables
locales de la función
```

```
local ruta=$2
local -n string_referenciado=$1
```

```
#En el string referenciado realizo el comando ls -lh con la ruta (en caso de que no
se haya pasado utiliza la actual),
```

```

#luego hace egrep de una expresión regular que toma todos los elementos que
inician con -, que van a ser siempre archivos,
#posteriormente filtra utilizando -k5, que hace que se tome la quinta columna del
ls -l (la del tamaño) y filtra en orden inverso (-r, más grandes primero)
#después de esto, toma los datos que son de interés gracias a awk (tamaño,
nombre, dueño y permisos), ya que se especifican las columnas de interés (5,9,3 y 1)
#y el tamaño lo convierte de bits a tamaño humano gracias a la función
human_size definida dentro del awk. Este revisa la cantidad de bits del tamaño y
#dependiendo de cuantos sean le agrega la unidad correspondiente (B, K, M,
G),
#ya por último elimina el primer guion (-) de la columna de permisos para que
coincida con el formato especificado en la consigna
string_referenciado=$(ls -l $ruta | egrep '^-' | sort -k5 -r | awk '
function human_size(size) {
    if (size < 1024) return size "B";
    else if (size < 1048576) return sprintf("%.1fK", size / 1024);
    else if (size < 1073741824) return sprintf("%.1fM", size / 1048576);
    else return sprintf("%.1fG", size / 1073741824);
}
{
    #Elimina el primer carácter '-' de los permisos gracias a la función
substract
    permisos = substr($1, 2)

    #Muestra el orden y formato final de los caracteres
    print human_size($5), $9, $3, permisos
}')
}

caso_g() {

    #Guardo la ruta pasada por parámetro como variable local dentro de la función
    local ruta=$1

    #Llamo a la función revisar_ruta_valida_dir con la ruta pasada por parámetro
    para que detecte si es correcta
    revisar_ruta_valida_dir $ruta

    #Creo un string llamado datos_archivos
    local datos_archivos

```

#Cargo el string con los elementos más grandes del directorio pasado por parámetro

```
cargar_string_archivos_grandes datos_archivos $ruta
```

#Especifico las columnas, agrego un salto de línea y pongo la información obtenida. Obtengo los primeros 4 valores (encabezados y 3 primeros elementos), y formo columnas.

```
echo -e "Tamaño\tNombre\tDueño\tPermisos\n$datos_archivos" | head -n 4 | column -t
```

```
}
```

#Encapsulo la lógica para cargar un string con archivos pequeños, para así poder usarla en las funciones -p y -x

```
cargar_string_archivos_pequeños() {
```

#Guardo la ruta y la referencia al string pasado por parámetro como variables locales de la función

```
local ruta=$2
local -n string_referenciado=$1
```

#En el string referenciado realizo el comando ls -lh con la ruta (en caso de que no se haya pasado utiliza la actual),

#luego hace egrep de una expresión regular que toma todos los elementos que inicien con -, que van a ser siempre archivos,

#posteriormente filtra utilizando -k5, que hace que se tome la quinta columna del ls -l (la del tamaño) y filtra en orden normal (más pequeños primero)

#después de esto, toma los datos que son de interés gracias a awk (tamaño, nombre, dueño y permisos), ya que se especifican las columnas de interés (5,9,3 y 1)

#y el tamaño lo convierte de bits a tamaño humano gracias a la función human_size definida dentro del awk. Este revisa la cantidad de bits del tamaño y #dependiendo de cuantos sean le agrega la unidad correspondiente (B, K, M, G), #ya por último elimina el primer guion (-) de la columna de permisos para que coincida con el formato especificado en la consigna

```
string_referenciado=$(ls -l $ruta | egrep '^-' | sort -k5 | awk '
```

```
function human_size(size) {
```

```
    if (size < 1024) return size "B";
    else if (size < 1048576) return sprintf("%.1fK", size / 1024);
    else if (size < 1073741824) return sprintf("%.1fM", size / 1048576);
    else return sprintf("%.1fG", size / 1073741824);
```

```

}

{

    #Elimina el primer carácter '-' de los permisos gracias a la función substract
    permisos = substr($1, 2)

    #Muestra el orden y formato final de los caracteres
    print human_size($5), $9, $3, permisos
}

caso_p() {

    #Guardo la ruta pasada por parámetro como variable local dentro de la función
    local ruta=$1

    #Llamo a la función revisar_ruta_valida_dir con la ruta pasada por parámetro
    para que detecte si es correcta
    revisar_ruta_valida_dir $ruta

    #Creo un string llamado datos_archivos
    local datos_archivos

    #Cargo el string con los elementos más pequeños del directorio pasado por
    parámetro
    cargar_string_archivos_pequeños datos_archivos $ruta

    #Especifico las columnas, agrego un salto de línea y pongo la información
    obtenida. Obtengo los primeros 4 valores (encabezados y 3 primeros elementos), y
    formo columnas.
    echo -e "Tamaño\tNombre\tDueño\tPermisos\n$datos_archivos" | head -n 4 |
    column -t
}

caso_x() {

    #Guardo la ruta y el parámetro pasados a la función como variable local dentro
    de la misma
    local parametro=$1
    local ruta=$2
}

```

```

local datos_archivos

case $parametro in
    "-g")
        revisar_ruta_valida_dir $ruta
        cargar_string_archivos_grandes datos_archivos $ruta
    ;;
    "-p")
        revisar_ruta_valida_dir $ruta
        cargar_string_archivos_pequeños datos_archivos $ruta
    ;;
    *)
        #Si el parámetro enviado no es -g ni -p, pero aún así incluyeron
        un segundo parámetro (en teoría una ruta), el parámetro es inválido
        if [ ! -z $2 ]; then
            echo "El parámetro colocado posterior a -x no es válido"
            exit 1
        elif [ ! -z $parametro ]; then
            #Si el parámetro pasado por comando no es ni -g ni -p, ni
            tampoco se pasó una ruta, se supone que parámetro es la ruta,
            #por lo que se revisa su validez
            revisar_ruta_valida_dir $parametro
        fi

        #Guardo en datos_archivos los archivos del directorio (inician
        con -) con los datos de interés seleccionados gracias a awk y borro el guión de los
        permisos (-).
        datos_archivos=$(ls -lh $parametro | egrep '^-' | awk '{print $5,
        $9, $3, substr($1, 2)})'

    ;;
esac

#Muestro los elementos, siempre y cuando sean ejecutables. El awk revisa que
la cuarta columna (permisos) contenga una x (ejecutable),
#y si la tiene imprime la línea entera, también la imprime si es la primera línea
(NR == 1). Por último, muestro los resultados en columnas.
if [[ "$parametro" == "-g" || "$parametro" == "-p" ]]; then
    #Si el parámetro es -g o -p, entonces muestro solo los primeros 4 resultados
(head -n 4)

```

```

        echo -e "Tamaño\tNombre\tDueño\tPermisos\n$datos_archivos" | awk
'{if ($4 ~ /x/ || NR == 1) print $0}' | column -t | head -n 4
else
# En caso contrario, muestro todos los resultados
echo -e "Tamaño\tNombre\tDueño\tPermisos\n$datos_archivos" | awk '{if ($4
~/x/ || NR == 1) print $0}' | column -t
fi
}

case $1 in
    "-ayuda")
        if (( $# != 1 )); then
            echo "ERROR: el comando '-ayuda' no admite parámetros"
            exit 1
        fi
        mostrar_ayuda
        ;;
    "-s")
        if (( $# != 2 )); then
            echo "ERROR: la cantidad de parámetros del comando '-s' no
coincide, para más ayuda ingrese '-ayuda'"
            exit 1
        fi
        caso_s $2
        ;;
    "-g")
        if (( $# < 1 || $# > 2 )); then
            echo "ERROR: la cantidad de parámetros del comando '-g' no
coincide, para más ayuda ingrese '-ayuda'"
            exit 1
        fi
        caso_g $2
        ;;
    "-p")
        if (( $# < 1 || $# > 2 )); then
            echo "ERROR: la cantidad de parámetros del comando '-p' no coincide,
para más ayuda ingrese '-ayuda'"
            exit 1
        fi
        caso_p $2
    esac
esac

```

```
;;
"-x")
    if (( $# < 1 || $# > 3 )); then
        echo "ERROR: la cantidad de parámetros del comando '-x' no coincide,
para más ayuda ingrese '-ayuda'"
        exit 1
    fi
    caso_x $2 $3
;;
*)
    echo "Parámetros no reconocidos, ingrese '-ayuda' para más
información"
    exit 1
;;
esac

exit
```

Testeo de todos los casos

Árbol de directorios y archivos utilizados para testear

C:\Users\salva\OneDrive\Escritorio\LaboratorioInfraestructura2024				
	Nombre	Fecha de modificación	Tipo	Tamaño
do	test	11/9/2024 20:37	Carpeta de archivos	
tos	AI_2024_list.sh	11/9/2024 20:32	Shell Script	10 KB
s	archivo.txt	11/9/2024 15:50	Documento de te...	0 KB
dio Co	Kanthus.png	5/12/2023 13:09	Archivo PNG	94 KB
nuevi	nuevabd24-3.sql	24/3/2024 13:39	Archivo de origen ...	19 KB
tios	nuevabd26-3.sql	26/3/2024 16:39	Archivo de origen ...	19 KB
	nuevabd27-3.sql	27/3/2024 20:46	Archivo de origen ...	20 KB
	nuevabd28-3.sql	22/4/2024 19:56	Archivo de origen ...	10 KB
	test.sh	11/9/2024 18:46	Shell Script	1 KB
	test2.sh	11/9/2024 18:48	Shell Script	1 KB
	test3.sh	11/9/2024 18:47	Shell Script	1 KB
	X2Download.app - MUSICA DE ASCENSO...	15/9/2023 19:38	Archivo MP3	10.747 KB

C:\Users\salva\OneDrive\Escritorio\LaboratorioInfraestructura2024\test				
	Nombre	Fecha de modificación	Tipo	Tamaño
ido	abc	11/9/2024 20:37	Carpeta de archivos	
s	archivotest.txt	11/9/2024 16:10	Documento de te...	0 KB
ntos	Kanthus.png	5/12/2023 13:09	Archivo PNG	94 KB
s	test.sh	11/9/2024 18:46	Shell Script	1 KB
	test2.sh	11/9/2024 18:48	Shell Script	1 KB
	test3.sh	11/9/2024 18:47	Shell Script	1 KB

C:\Users\salva\OneDrive\Escritorio\LaboratorioInfraestructura2024\test\abc				
	Nombre	Fecha de modificación	Tipo	Tamaño
ido	Kanthus.png	5/12/2023 13:09	Archivo PNG	94 KB
s	nuevabd26-3.sql	26/3/2024 16:39	Archivo de origen ...	19 KB
ntos	nuevabd27-3.sql	27/3/2024 20:46	Archivo de origen ...	20 KB
s	nuevabd28-3.sql	22/4/2024 19:56	Archivo de origen ...	10 KB
	test.sh	11/9/2024 18:46	Shell Script	1 KB
	test2.sh	11/9/2024 18:48	Shell Script	1 KB
	test3.sh	11/9/2024 18:47	Shell Script	1 KB

Errores por malos parámetros al script

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh
Parámetros no reconocidos, ingrese '-ayuda' para más información
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -h
Parámetros no reconocidos, ingrese '-ayuda' para más información
```

-ayuda

Errores

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -ayuda -a
ERROR: el comando '-ayuda' no admite parámetros
```

Resultado esperado

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -ayuda
-----
La sintaxis de ejecución del comando debe ser la siguiente:
./AI_2024_list.sh [PARÁMETRO] [RUTA]
En [PARÁMETRO] EXISTEN LAS SIGUIENTES OPCIONES:
COMANDO -s
Despliega la información sobre un archivo, siendo ésta su nombre, tamaño, dueño y permisos
En [RUTA] se debe incluir una ruta relativa o absoluta obligatoriamente

COMANDO -g
Permite ver información sobre los tres archivos más grandes del directorio especificado.
En [RUTA] se debe incluir una ruta absoluta o relativa a un directorio. Si no se incluye, [RUTA] será el directorio actual

COMANDO -p
Permite ver información sobre los tres archivos más pequeños del directorio especificado.
En [RUTA] se debe incluir una ruta absoluta o relativa a un directorio. Si no se incluye, [RUTA] será el directorio actual

COMANDO -x
Mostrará información sólo de los archivos ejecutables del directorio especificado.
En [RUTA] se debe incluir una ruta absoluta o relativa a un directorio. Si no se incluye, [RUTA] será el directorio actual
Puede combinarse junto a los parámetros -g o -p introducidos posterior a -x, pero no con ambos a la vez.
```

-s

Errores

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -s
ERROR: la cantidad de parámetros del comando '-s' no coincide, para más ayuda ingrese '-ayuda'
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -s -s -s
ERROR: la cantidad de parámetros del comando '-s' no coincide, para más ayuda ingrese '-ayuda'
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -s test
ERROR: La ruta especificada no pertenece a un archivo
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -s lalala
ERROR: La ruta especificada no existe
```

Resultado esperado

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -s Kanthus.png
Nombre: Kanthus.png
Tamaño: 94K
Dueño: salva
Permisos: rw-r--r--
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -s C:/Users/salva/OneDrive/Escritorio/LaboratorioInfraestructura2024/Kanthus.png
Nombre: Kanthus.png
Tamaño: 94K
Dueño: salva
Permisos: rw-r--r--
```

-g

Errores

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -g -g -g
ERROR: La cantidad de parámetros del comando '-g' no coincide, para más ayuda ingrese '-ayuda'
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -g test.sh
ERROR: La ruta especificada no pertenece a un directorio
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -g lalala
ERROR: La ruta especificada no existe
```

Resultado esperado

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -g
Tamaño  Nombre          Dueño  Permisos
10.5M   X2Download.app  salva  rw-r--r--
93.6K   Kanthus.png    salva  rw-r--r--
19.8K   nuevabd27-3.sql salva  rw-r--r--
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -g test
Tamaño  Nombre      Dueño  Permisos
93.6K   Kanthus.png salva  rw-r--r--
25B     test2.sh    salva  rwxr-xr-x
13B     test.sh    salva  rwxr-xr-x
```

-p

Errores

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -p -p -p
ERROR: La cantidad de parámetros del comando '-p' no coincide, para más ayuda ingrese '-ayuda'
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -p test.sh
ERROR: La ruta especificada no pertenece a un directorio
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -p Lalala
ERROR: La ruta especificada no existe
```

Resultado esperado

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -p
Tamaño  Nombre      Dueño  Permisos
0B      archivo.txt salva  rw-r--r--
12B     test3.sh    salva  rwxr-xr-x
13B     test.sh    salva  rwxr-xr-x
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -p test
Tamaño  Nombre      Dueño  Permisos
0B      archivotest.txt salva  rw-r--r--
12B     test3.sh    salva  rwxr-xr-x
13B     test.sh    salva  rwxr-xr-x
```

-X

Errores

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -x Kanthus.png
ERROR: La ruta especificada no pertenece a un directorio
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -x -g Kanthus.png
ERROR: La ruta especificada no pertenece a un directorio
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -x -p Kanthus.png
ERROR: La ruta especificada no pertenece a un directorio
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -x lalala
ERROR: La ruta especificada no existe
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -x -g lalala
ERROR: La ruta especificada no existe
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -x -p lalala
ERROR: La ruta especificada no existe
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -x -x -x -x
ERROR: La cantidad de parámetros del comando '-x' no coincide, para más ayuda ingrese '-ayuda'
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -x -h test
El parámetro colocado posterior a -x no es válido
```

Resultado esperado

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -x
Tamaño  Nombre      Dueño  Permisos
12K     AI_2024_list.sh  salva  rwxr-xr-x
13      test.sh       salva  rwxr-xr-x
25      test2.sh      salva  rwxr-xr-x
12      test3.sh      salva  rwxr-xr-x
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -x test
Tamaño  Nombre      Dueño  Permisos
13      test.sh       salva  rwxr-xr-x
25      test2.sh      salva  rwxr-xr-x
12      test3.sh      salva  rwxr-xr-x
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -x -g
Tamaño  Nombre      Dueño  Permisos
11.4K   AI_2024_list.sh  salva  rwxr-xr-x
25B     test2.sh       salva  rwxr-xr-x
13B     test.sh        salva  rwxr-xr-x
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -x -g test
Tamaño  Nombre      Dueño  Permisos
25B     test2.sh       salva  rwxr-xr-x
13B     test.sh        salva  rwxr-xr-x
12B     test3.sh       salva  rwxr-xr-x
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -x -p
Tamaño  Nombre          Dueño  Permisos
12B     test3.sh        salva  rwxr-xr-x
13B     test.sh         salva  rwxr-xr-x
25B     test2.sh        salva  rwxr-xr-x
```

```
salva@SALVA MINGW64 ~/OneDrive/Escritorio/LaboratorioInfraestructura2024
$ ./AI_2024_list.sh -x -p test
Tamaño  Nombre          Dueño  Permisos
12B     test3.sh        salva  rwxr-xr-x
13B     test.sh         salva  rwxr-xr-x
25B     test2.sh        salva  rwxr-xr-x
```

Log de ideas, cambios y decisiones

Función para mostrar ayuda

```
mostrar_ayuda() {
    echo "-----"
    echo
    echo "La sintaxis de ejecución del comando debe ser la siguiente:"
    echo "./AI_2024_list.sh [PARÁMETRO] [RUTA]"
    echo
    echo "En [PARÁMETRO] EXISTEN LAS SIGUIENTES OPCIONES:"
    echo
    echo "COMANDO -s"
    echo "Despliega la información sobre un archivo, siendo esta su nombre, tamaño, dueño y permisos"
    echo "En [RUTA] se debe incluir una ruta relativa o absoluta obligatoriamente"
    echo
    echo "COMANDO -g"
    echo "Permite ver información sobre los tres archivos más grandes del directorio especificado."
    echo "En [RUTA] se debe incluir una ruta absoluta o relativa a un directorio. Si no se incluye, [RUTA] será el directorio actual"
    echo
    echo "COMANDO -p"
    echo "Permite ver información sobre los tres archivos más pequeños del directorio especificado."
    echo "En [RUTA] se debe incluir una ruta absoluta o relativa a un directorio. Si no se incluye, [RUTA] será el directorio actual"
    echo
    echo "COMANDO -x"
    echo "Mostrará información sólo de los archivos ejecutables del directorio especificado."
    echo "En [RUTA] se debe incluir una ruta absoluta o relativa a un directorio. Si no se incluye, [RUTA] será el directorio actual"
    echo "Puede combinarse junto a los parámetros -g o -p introducidos posterior a -x, pero no con ambos a la vez."
    echo
    echo "-----"
}
```

Una idea que tuve desde antes de arrancar con el laboratorio fue la de incluir una función de ayuda, que muestre la sintaxis y las diferentes entradas que puede tener el script. Fue una buena idea implementarla como primer elemento ya que resultó en una herramienta útil cada vez que quería recordar qué debería hacer y cómo debería comportarse cada parámetro.

Función para manejar el caso -s

```

caso_s() {
    #Guardo la ruta pasada por parámetro como variable local dentro de la función
    local ruta=$1

    #Reviso que la ruta especificada exista
    if [ ! -e $ruta ]; then
        echo "ERROR: la ruta especificada no existe"
        exit 1
    #Reviso que la ruta especificada pertenezca a un archivo
    elif [ ! -f $ruta ]; then
        echo "ERROR: la ruta especificada no pertenece a un archivo"
        exit 1
    fi

    #Guardo el nombre del archivo en una variable local, uso basename para prescindir de la ruta y que guarde SOLO el nombre del archivo
    local nombre=$(basename $ruta)

    #Hago (ls -lh $ruta) para obtener la información del archivo pasado por parámetro, y luego utilizo awk para obtener solo las columnas
    #con la información que me interesa y formatearlas. Esto lo guardo en la variable local info
    local info=$(ls -lh $ruta | awk '{print "Tamaño: "$5, "\nDueño: "$3, "\nPermisos: "substr($1, 2)})'

    #Hago echo -e para que los saltos de linea (\n) de la variable local info se representen correctamente
    echo "Nombre: $nombre"
    echo -e "$info";
}

```

En esta función, en la parte de la declaración de info, originalmente iba a hacer print del nombre del archivo también, lo que hubiera sido un error en la resolución de la consigna ya que, en caso de que el archivo esté en una cadena de directorios, imprimirá esta cadena junto al nombre.

Decidí usar basename para extraer el nombre y manejarlo en un echo aparte. Otra manera hubiera sido pasando como parámetro al awk dicho nombre, para formatear todo en un mismo lugar y luego utilizar un solo echo, pero complejizaba mucho la sintaxis del código y la funcionalidad iba a ser exactamente la misma.

Otra decisión que tomé mientras implementaba la función fue la de guardar la ruta pasada por parámetro en una variable local, ya que simplifica mucho la lectura para quien no conoce el código.

Funciones para manejar los casos -g y -p

```
#Encapsulo la lógica para cargar un string con archivos grandes, para así poder usarla en las funciones -g y -x
cargar_string_archivos_grandes() {
    #Guardo la ruta y la referencia al string pasado por parámetro como variables locales de la función
    local ruta $2
    local -n string_referenciado=$1

    #En el string referenciado realizo el comando ls -lh con la ruta (en caso de que no se haya pasado utiliza la actual)
    #luego hace egrep de una expresión regular que toma todos los elementos que inicien con -, que van a ser siempre archivos,
    #posteriormente filtra -k5, que hace que se tome la quinta columna del ls -l (la del tamaño) y filtra en orden normal (más pequeños primero)
    #después de esto, toma los datos que son de interés gracias a awk (tamaño, nombre, dueño y permisos), ya que se especifican las columnas de interés (5,9,3 y 1)
    #y el tamaño lo convierte de bits a tamaño humano gracias a la función human_size definida dentro del awk. Este revisa la cantidad de bits del tamaño y
    #dependiendo de cuantos sean le agrega la unidad correspondiente (B, K, M, G),
    #ya por ultimo elimina el primer guion (-) de la columna de permisos para que coincida con el formato especificado en la consigna
    string_referenciado=$(ls -l $ruta | egrep '^-' | sort -k5 -r | awk '
        function human_size(size) {
            if (size < 1024) return size "B";
            else if (size < 1048576) return sprintf("%Kfk", size / 1024);
            else if (size < 1073741824) return sprintf("%IM", size / 1048576);
            else return sprintf("%.1fG", size / 1073741824);
        }
        {
            #Elimina el primer carácter '-' de los permisos gracias a la función substract
            permisos = substr($1, 2)

            #Muestra el orden y formato final de los caracteres
            print human_size($5), $9, $3, permisos
        }
    ')
}

caso_g() {
    #Guardo la ruta pasada por parámetro como variable local dentro de la función
    local ruta $1

    #Llamo a la función revisar_ruta_valida_dir con la ruta pasada por parametro para que detecte si es correcta
    revisar_ruta_valida_dir $ruta

    #Creo un string llamado datos_archivos
    local datos_archivos

    #Cargo el string con los elementos más grandes del directorio pasado por parametro
    cargar_string_archivos_grandes datos_archivos $ruta

    #Especifico las columnas, agrego un salto de linea y pongo la información obtenida. Obtengo los primeros 4 valores (encabezados y 3 primeros elementos), y formo columnas.
    echo -e "$ruta\n$datos_archivos" | head -n 4 | column -t
}

```

```
#Encapsulo la lógica para cargar un string con archivos pequeños, para así poder usarla en las funciones -p y -x
cargar_string_archivos_pequeños() {
    #Guardo la ruta y la referencia al string pasado por parámetro como variables locales de la función
    local ruta $2
    local -n string_referenciado=$1

    #En el string referenciado realizo el comando ls -lh con la ruta (en caso de que no se haya pasado utilizar la actual),
    #luego hace egrep de una expresión regular que toma todos los elementos que inicien con -, que van a ser siempre archivos,
    #posteriormente filtra -k5, que hace que se tome la quinta columna del ls -l (la del tamaño) y filtra en orden normal (más pequeños primero)
    #después de esto, toma los datos que son de interés gracias a awk (tamaño, nombre, dueño y permisos), ya que se especifican las columnas de interés (5,9,3 y 1)
    #y el tamaño lo convierte de bits a tamaño humano gracias a la función human_size definida dentro del awk. Este revisa la cantidad de bits del tamaño y
    #dependiendo de cuantos sean le agrega la unidad correspondiente (B, K, M, G),
    #ya por ultimo elimina el primer guion (-) de la columna de permisos para que coincida con el formato especificado en la consigna
    string_referenciado=$(ls -l $ruta | egrep '^-' | sort -k5 -r | awk '
        function human_size(size) {
            if (size < 1024) return size "B";
            else if (size < 1048576) return sprintf("%Kfk", size / 1024);
            else if (size < 1073741824) return sprintf("%IM", size / 1048576);
            else return sprintf("%.1fG", size / 1073741824);
        }
        {
            #Elimina el primer carácter '-' de los permisos gracias a la función substract
            permisos = substr($1, 2)

            #Muestra el orden y formato final de los caracteres
            print human_size($5), $9, $3, permisos
        }
    ')
}

caso_p() {
    #Guardo la ruta pasada por parámetro como variable local dentro de la función
    local ruta $1

    #Llamo a la función revisar_ruta_valida_dir con la ruta pasada por parámetro para que detecte si es correcta
    revisar_ruta_valida_dir $ruta

    #Creo un string llamado datos_archivos
    local datos_archivos

    #Cargo el string con los elementos más pequeños del directorio pasado por parámetro
    cargar_string_archivos_pequeños datos_archivos $ruta

    #Especifico las columnas, agrego un salto de linea y pongo la información obtenida. Obtengo los primeros 4 valores (encabezados y 3 primeros elementos), y formo columnas.
    echo -e "$ruta\n$datos_archivos" | head -n 4 | column -t
}

```

En estas funciones originalmente no había creado una función aparte para cargar el string, sino que lo hacía mismo dentro de la función que manejaba el caso. Después, cuando fui a implementar la función del caso -x, vi necesario manejar la lógica del cargado en funciones aparte, que encapsulan solo esa parte de la lógica, para no tener que repetir código.

Una cosa que cambié posteriormente en estas funciones para cargar strings es que no carguen solo los tres primeros elementos, sino todos. Así puedo reutilizar la función al realizar el manejador del caso -x que debe mostrar TODOS los archivos ejecutables, y como se puede combinar con -g y -p no me interesaría que al cargar dichos strings me corte la cantidad de elementos ejecutables solo a tres.

Otro cambio en las funciones de carga de strings fue una subfunción dentro del awk que formatea para que muestre correctamente la unidad de tamaño. Con ls -lh no pude ya que al hacer sort por tamaño, como este no era un número natural, sino uno decimal y con letras (8.9G, por ejemplo), quedaban desordenados. Primero tenía que hacer el sort en bits con el ls -l y luego mostrar las unidades. Por esto me decidí a usar la función embebida en el awk (human_size(size)) que busqué en internet.

Un cambio a las funciones de los casos (-p y -g) fue que la cadena obtenida gracias a la función de carga de strings se imprimiera en formato de columnas gracias al comando column -t. Originalmente las columnas estaban desordenadas y no cumplían con el formato brindado por el profesor.

Otra función para no repetir código sería revisar_ruta_valida_dir, que, dada una ruta, revisa que exista y que pertenezca a un directorio. Esta será aprovechada también en la función -x y su lógica es la siguiente:

```
#Encapsulo la lógica para revisar que una ruta pasada por parámetro exista y sea un directorio para poder usarla en las funciones -g, -p y -x
revisar_ruta_valida_dir() {
    #Guardo la ruta pasada por parámetro como variable local dentro de la función
    local ruta=$1

    #Reviso que la ruta especificada exista
    if [ ! -e $ruta ]; then
        echo "ERROR: la ruta especificada no existe"
        exit 1
    #Reviso que la ruta especificada sea un directorio
    elif [ ! -d $ruta ]; then
        echo "ERROR: la ruta especificada no pertenece a un directorio"
        exit 1
    fi
}
```

Función para manejar el caso -x

```
case_x() {
    #Guardo la ruta y el parámetro pasados a la función como variable local dentro de la misma
    local parametro=$1
    local ruta=$2

    local datos_archivos

    case $parametro in
        "-g")
            revisar_ruta_valida_dir $ruta
            cargar_string_archivos_grandes datos_archivos $ruta
        ;;
        "-p")
            revisar_ruta_valida_dir $ruta
            cargar_string_archivos_pequeños datos_archivos $ruta
        ;;
        *)
            #Si el parámetro enviado no es -g ni -p, pero aún así incluyeron un segundo parámetro (en teoría una ruta), el parámetro es inválido
            if [ ! -z $2 ]; then
                echo "El parámetro colocado posterior a -x no es válido"
                exit 1
            elif [ ! -z $parametro ]; then
                #Si el parámetro pasado por comando no es ni -g ni -p, ni tampoco se pasó una ruta, se supone que parámetro es la ruta,
                #por lo que se revisa su validez
                revisar_ruta_valida_dir $parametro
            fi
            #Guardo en datos_archivos los archivos del directorio (inician con -) con los datos de interés seleccionados gracias a awk y borro el guion de los permisos (-).
            datos_archivos=$(ls -lh $parametro | egrep '^-[a-z]' | awk '{print $5, $9, $3, substr($1, 2)})'

            esac
        ;;
        #Muestro los elementos, siempre y cuando sean ejecutables. El awk revisa que la cuarta columna (permisos) contenga una x (ejecutable),
        #y si la tiene imprime la línea entera, también la imprime si es la primera línea (NR == 1). Por último, muestro los resultados en columnas.
        #Si el parámetro es -g || $parametro == "-p" ]; then
        #    #Si el parámetro es -g o -p, entonces muestro solo los primeros 4 resultados (head -n 4)
        #    echo -e "Tamaño\tNombre\tDueño\tPermisos\n$datos_archivos" | awk '{if ($4 ~ /x/ || NR == 1) print $0}' | column -t | head -n 4
    else
        # En caso contrario, muestro todos los resultados
        echo -e "Tamaño\tNombre\tDueño\tPermisos\n$datos_archivos" | awk '{if ($4 ~ /x/ || NR == 1) print $0}' | column -t
    fi
}
```

Esta función al incluir las dos anteriores fue la que más complicó la lógica y me hizo modificar funciones que ya estaban funcionales para que puedan funcionar con esta. Como comenté antes, sería este el caso de las funciones de carga de strings.

Otro factor a tener en cuenta sobre esta función fue que tuve que pensar en cómo tratar los distintos casos con los parámetros:

- Si me mandan un parámetro adicional (-g o -p) y una ruta.
- Si me mandan solo una ruta, sin parámetro adicional.
- Si me mandan un parámetro adicional, sin una ruta.

Estos últimos dos son los más complicados, ya que tuve que revisar que el parámetro no sea una ruta o que la ruta no sea un parámetro, ya que el orden de los parámetros enviados siempre va a ser \$1 y \$2, pero no tengo manera de saber si el \$1 que me mandaron es ruta o es parámetro y viceversa.

Otro cambio que tuve que realizar, al igual que en las funciones -g y -p, fue la de formatear el resultado en columnas para poder cumplir con la consigna correctamente.

Otros cambios o ideas generales

En un inicio usaba ls -l para traer la información de archivos o directorios, tuve que cambiarlo a ls -lh para que así muestre el dato del tamaño con unidades en los casos de -s y -x cuando no se usan otros parámetros. Para -p y -g y sus combinaciones con -x, usé una función embebida en el comando awk que transforma una unidad en bits a otra más legible, con unidades mayores. Este comando junto a column y substr, del que hablaré a continuación, fueron los únicos que tuve que buscar debido a que no se encontraban incluidos en los contenidos teóricos proporcionados.

En todas las funciones en un inicio mostraba mal el formato de los permisos, ya que incluía el guión inicial que indica si es un directorio o no (-), gracias al comando substr logré hacer que no se muestre, logrando un formato adecuado.

Metodología de desarrollo

Procederé a realizar un punteo del proceso de desarrollo llevado a cabo. Para más detalle sobre decisiones, ideas y cambios, revisar el log.

- Leí la letra y la interpreté de manera que pudiera comenzar a implementar las funcionalidades detalladas con un plan en mente.
- Desde un inicio planteé hacerme un resumen de las funcionalidades, sus parámetros y casos de error que terminé planteando en el comando ‘-ayuda’.
- Cree el archivo .sh, me hice un par de archivos para testear en una carpeta (para verla consultar “Testeo de todos los casos”, en la parte la cadena de directorios utilizada) y cree el case que ejecutaría todos los casos, junto a los if’s que solo permitían la ejecución de la función indicada cuando la cantidad de parámetros introducidos era correcta.
- Comencé a desarrollar cada funcionalidad. Como comenté previamente, comencé con el caso de ‘ayuda’ para tener una guía a mano en todo momento de qué implementar.
- Posteriormente desarrollé la función -s, la cual no me significó mayor problema. Los únicos cambios de plan que tuve, como quedó planteado en el log, fueron los de usar ls -lh para ver los tamaños correctamente (con unidades), y de separar el nombre del archivo y colocarlo fuera del awk para poder usar basename(), y que se muestre sin la ruta. La última consideración fue usar echo -e para que se muestren los saltos de línea incluidos en el ‘awk’.
- Una vez terminada la función anterior, comencé con la función -g, que sabía que sería muy similar a la -p, y resultó ciertamente así. Para ambas en un inicio había creado una sola función, que después tuve que dividir para poder reutilizar en la funcionalidad -x. Resultó algo más complicada, sobre todo el tema de cómo mostrar correctamente los tamaños de archivos pero pudiendo filtrar a la vez, cosa que resolví gracias a una función embebida en el awk, que conseguí investigando en internet. Aparte de esto utilizan otra función que revisa que la ruta pasada por parámetro exista, y si lo hace que pertenezca a un directorio. La única diferencia entre -g y -p es que una filtra normal (sort) y otra filtra invertido (sort -r).
- Por último, la funcionalidad -x. Esta fue ciertamente la más complicada de implementar, sobre todo porque al poder utilizarse junto a las otras dos funciones, tuve que cambiar parte de la lógica ya existente para poder reutilizar código correctamente. Para más información, revisar el log.
- Una vez completadas todas las funcionalidades, las revisé y agregué detalles faltantes, como puede ser que se muestren los tamaños, que se muestre en columnas o que los permisos no tengan el guión inicial.

Enlace al video y al script

Ver vídeo explicativo

https://drive.google.com/file/d/15eOAcgEmIeA1Z_y-DagqWfLLub2MyiKS/view?usp=sharing

Descargar script

<https://drive.google.com/file/d/19j1j4XmOd6OBhzougUl51mlHCdzzx6m5/view?usp=sharing>

Fuentes

Hostinger. (n.d.). ¿Qué es Bash? Hostinger. Recuperado de

https://www.hostinger.es/tutoriales/bash-script-linux#%C2%BFQue_es_Bash

Klemens, M. (2020, octubre 14). *Bash scripting tutorial: Linux shell script and command line for beginners*. freeCodeCamp. Recuperado de

<https://www.freecodecamp.org/news/bash-scripting-tutorial-linux-shell-script-and-command-line-for-beginners/>

GeeksforGeeks. (n.d.). *Bash scripting: Introduction to bash and bash scripting*.

Recuperado de

<https://www.geeksforgeeks.org/bash-scripting-introduction-to-bash-and-bash-scripting/>

Bellizzi, I. (2024). *Materiales proporcionados por la materia Administración de Infraestructuras*. Universidad UTEC, Administración de Infraestructura.