Practica 3 (Shell Scriping) - Resolucion

Trabajo Práctico 3.pdf

---

1. ¿Qué es el Shell Scripting? ¿A qué tipos de tareas están orientados los script? ¿Los scripts deben compilarse? ¿Por qué?

    -Son secuencias de comandos destinados a ejecutarse en un Shell. Estan orientados en automatizar tareas o realizar varias tareas diviendolas en subtaeras mas sencillas. No debn compilarse porque son interpretados en tiempo de ejecucion por el interprete de comandos

2. Investigar la funcionalidad de los comandos echo y read.

-echo → imprime una cadena de texto o variables en la salida estandar

-read → lee de teclado las entradas delu suario hasta que se encuentra un salto de linea

a. ¿Cómo se indican los comentarios dentro de un script?

    -Comentario simple # ; Comentario de varias lineas <<COMMENT

b. ¿Cómo se declaran y se hace referencia a variables dentro de un script?

```
NOMBRE="pepe" # SIN ESPACIOS AL REDEDOR DEL =
echo $NOMBRE # → pepe
echo ${NOMBRE}todo_esto_no_es_parte # CONCATENA

arreglo_a=() # ARRAY VACIO
```

```
arreglo_b=(1 2 3 4) # INICIALIZADO
arreglo_b[2]=spam
# ARRANCA EN 1, IMPRIME SPAM
echo ${arreglo2[2]}
# DEVUELVE TODOS LOS ELEMENTOS DEL ARREGLO CON '@' o '*'
echo ${arreglo[@]}
# TAMAÑO DEL ARREGLO
echo ${#arreglo[@]}
# BORRA CONTENIDO DE LA POS 2, QUEDA VACIA
unset arreglo[2]

# GUARDA EN MIS ARCHIVOS EL CONTENIDO DEL HOME
# DEL USUARIO WHOAMI
mis_archivos = "$(ls/home/$(whoami))"
```

3. Crear dentro del directorio personal del usuario logueado un directorio llamado *practica-shell-script* y dentro de él un archivo llamado mostrar.sh cuyo contenido sea el siguiente:

#!/bin/bash

Comentarios acerca de lo que hace el script # Siempre comento mis scripts, si no lo hago hoy, # 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:"

a. Asignar al archivo creado los permisos necesarios de manera que pueda ejecutarlo

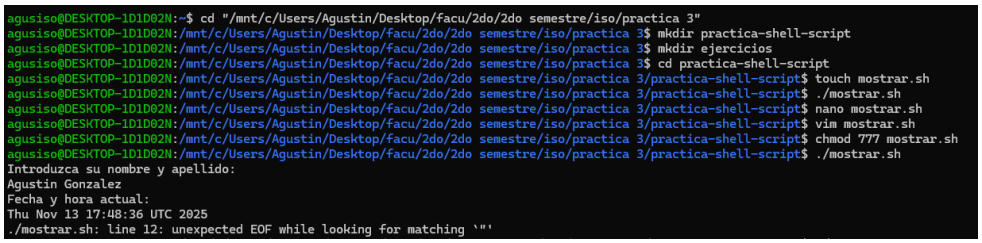
b. Ejecutar el archivo creado de la siguiente manera: ./mostrar.sh

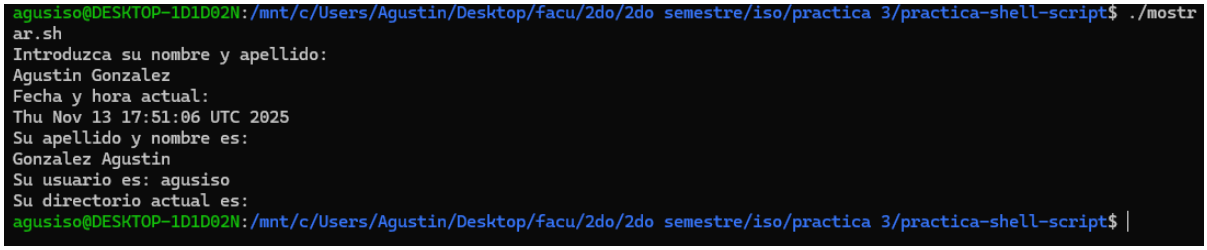
c. ¿Qué resultado visualiza?

d. Las backquotes (`) entre el comando whoami ilustran el uso de la sustitución de comandos. ¿Qué significa esto?

-Se utilizan para ejecutar un comando dentro de otro. En este caso ejecuta el comando que muestra el nombre del usuario y sustituye la salida del comando interno en el comando externo

e. Realizar modificaciones al script anteriormente creado de manera de poder mostrar distintos resultados (cuál es su directorio personal, el contenido de un directorio en particular, el espacio libre en disco, etc.). Pida que se introduzcan por teclado (entrada estándar) otros datos.





4. Parametrización: ¿Cómo se acceden a los parámetros enviados al script al momento de su invocación? ¿Qué información contienen las variables $#, $*, $? y $HOME dentro de un script?

-Se acceden a traves de variables, las cuales estan predefinidas y contienen la informacion de los argumentos proporcionados al script. Los parametros se almacenan en variables $1, $2, ... $N.

-Cuando ejecutas el script con X cantidad de parametros le pasas los parametros justo despues del llamado

→ ./script.sh param1 param2

-$# → Contiene numero de argumentos que se pasaron al script (en el caso de arriba seria 2)

-$* → Representa todos los argumentos en una cadena de texto sin espacios en blanco entre ellos

-$? → Contiene el codigo de salida del ultimo comando ejecutado. En caso de exito → 0, en caso de fallo → <> 0

-$HOME → Ruta al directorio raiz del usuario actual

5. ¿Cual es la funcionalidad de comando exit? ¿Qué valores recibe como parámetro y cuál es su significado?

-Es una funcion que se utiliza para terminar un script. Recibe como parametro un valor entre 0 y 255 y si significado es el siguiente:

-Si el script termino de manera exitosa → Exit = 0

-Cualquier otro tipo de error o estado particular→ Exit <> 0

6. El comando *expr* permite la evaluación de expresiones. Su sintaxis es: expr arg1 op arg2, donde arg1 y arg2 representan argumentos y op la operación de la expresión. Investigar que tipo de operaciones se pueden utilizar.

-Evalua la expresion y mostrar el resultado en la salida estandar

-Tipos:

-Aritmeticas:

Operación	Ejemplo	Resultado
Suma	`expr 5 + 3`	8
Resta	`expr 10 - 4`	6
Multiplicación	`expr 4 \* 2`	8
División	`expr 9 / 3`	3
Módulo (resto)	`expr 10 % 3`	1

-Comparacion:

Operador	Ejemplo	Resultado
=	expr 5 = 5	1
!=	expr 5 != 3	1
>	expr 6 \> 3	1
<	expr 2 \< 5	1
>=	expr 5 \>= 5	1
<=	expr 4 \<= 2	0

-Logicas:

Operador	Significado	Ejemplo	Resultado
`	`	OR lógico	expr 0 | 1
&	AND lógico	expr 1 \& 0	0

-Con cadenas:

Operación	Descripción	Ejemplo	Resultado
length	Longitud de una cadena	expr length "hola"	4
index	Posición del primer carácter de una subcadena	expr index "ubuntu" "b"	2
substr	Extrae una subcadena	expr substr "sistemas" 3 4	stem

7. El comando *testexpresion* permite evaluar expresiones y generar un valor de retorno, true o false. Este comando puede ser reemplazado por el uso de corchetes de la siguiente manera [expresión]. Investigar qué tipo de expresiones pueden ser usadas con el comando test. Tenga en cuenta operaciones para: evaluación de archivos, evaluación de cadenas de caracteres y evaluaciones numéricas.

-Evaluacion de archivos:

Expresión	Significado	Ejemplo
-e archivo	Existe	[-e /home/user/archivo.txt]
-f archivo	Es un archivo regular	[-f archivo.txt]

Expresión	Significado	Ejemplo
`-d directorio`	Es un directorio	`[ -d /home/user/docs ]`
`-r archivo`	Tiene permiso de lectura	`[ -r archivo.txt ]`
`-w archivo`	Tiene permiso de escritura	`[ -w archivo.txt ]`
`-x archivo`	Tiene permiso de ejecución	`[ -x script.sh ]`
`-s archivo`	El archivo no está vacío	`[ -s archivo.txt ]`

-Evaluacion de cadenas de carecteres:

Expresión	Significado	Ejemplo
`-z cadena`	La cadena está vacía	`[ -z "$var" ]`
`-n cadena`	La cadena **no** está vacía	`[ -n "$var" ]`
`cadena1 = cadena2`	Son iguales	`[ "$a" = "$b" ]`
`cadena1 != cadena2`	Son distintas	`[ "$a" != "$b" ]`

-Evaluacion numerica:

Operador	Significado	Ejemplo
`-eq`	Igual	`[ $a -eq $b ]`
`-ne`	Distinto	`[ $a -ne $b ]`
`-gt`	Mayor que	`[ $a -gt $b ]`
`-ge`	Mayor o igual	`[ $a -ge $b ]`
`-lt`	Menor que	`[ $a -lt $b ]`
`-le`	Menor o igual	`[ $a -le $b ]`

8. Estructuras de control. Investigue la sintaxis de las siguientes estructuras de control incluidas en shell scripting:

➢ if:

```
if [ condition ]
then
  bloque
elif [ condition ]
then
  bloque
else
  bloque
fi
```

➢ case:

Selección:

```
case $variable in
"valor 1")
  bloque
  ;;
"valor 2")
  bloque
  ;;
*)
  bloque
  ;;
esac
```

➢ while:

while

```
while [ condicion ] # Mientras se cumpla la condición
do
  bloque
done
```

➢ for:

- C-*style*:

```
for ((i=0; i < 10; i++))
do
  bloque
done
```

- Con lista de valores (foreach):

```
for i in valor1 valor2 valor3 valorN
do
  bloque
done
```

➢ select:

```
select accion in Nuevo Salir
do
  case $accion in
    "Nuevo")
      echo "Seleccionado: Nuevo"
      ;;
    "Salir")
      exit 0
      ;;
  esac
done
```

9. ¿Qué acciones realizan las sentencias *break* y *continue* dentro de un bucle? ¿Qué parámetros reciben?

-Break: termina el bucle actual y pasa el control del program al comando que sigue al bucle terminado. Se usa para salir de un bucle for, while, until o select en determinado momento y pasar la ejecucion a la siguiente iteracion. break n, siendo n un argumento opcional mayor o igual a 1

-Continue: omite los comandos restantes dentro del cuerpo del bucle que lo encierra para la iteracion actual. Pasa el cnotrol del programa a la siguiente iteracion. continue n, siendo n mayor o igual a 1 e indica que se reanuda el bucle n-esimo

10.¿Qué tipo de variables existen? ¿Es shell script fuertemente tipado? ¿Se pueden definir arreglos? ¿Cómo?

- `bash` soporta *strings* y *arrays*
- Los nombres son *case sensitive*
- Para crear una variable:

```
NOMBRE="pepe" # SIN espacios alrededor del =
```

- Para accederla se usa `$`:

```
echo $NOMBRE
```

- Para evitar ambigüedades se pueden usar llaves:

```
# Esto no accede a $NOMBRE
echo $NOMBREesto_no_es_parte_de_la_variable
# Esto sí
echo ${NOMBRE}esto_no_es_parte_de_la_variable
```

- Creación:

```
arreglo_a=() # Se crea vacío
arreglo_b=(1 2 3 5 8 13 21) # Inicializado
```

- Asignación de un valor en una posición concreta:

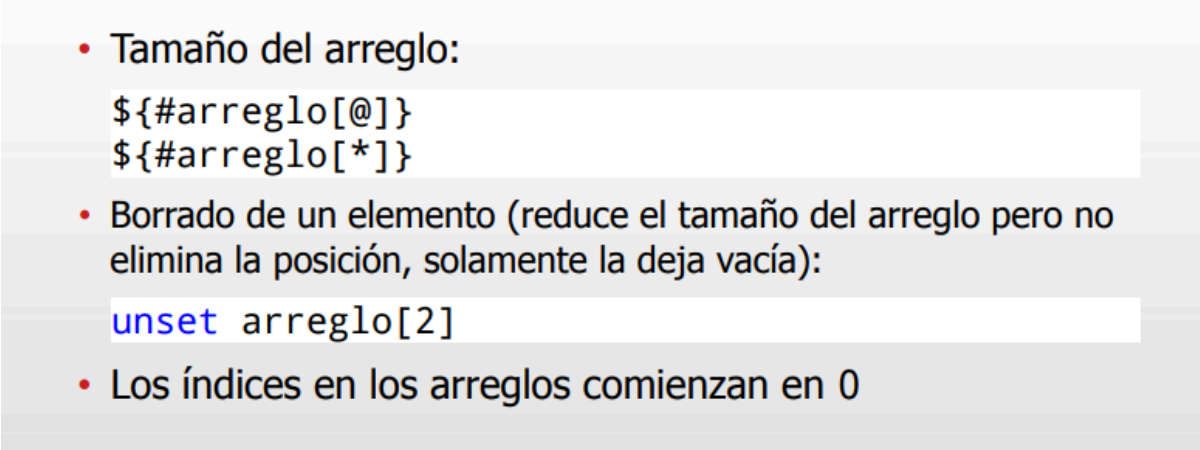
```
arreglo_b[2]=spam
```

- Acceso a un valor del arreglo (en este caso las llaves no son opcionales):

```
echo ${arreglo_b[2]}
copia=${arreglo_b[2]}
```

- Acceso a todos los valores del arreglo:

```
echo ${arreglo[@]} # o bien ${arreglo[*]}
```



(en el punto 2 tengo mas o menos lo mismo explicado)

11.¿Pueden definirse funciones dentro de un script? ¿Cómo? ¿Cómo se maneja el pasaje de parámetros de una función a la otra?

-Las funciones permiten modularizar el comportamiento de los scripts. Se pueden declarar de 2 formas:

-function nombre { bloque }

-nombre() { bloque }

-Con la sentencia return se retorna un valor entre 0 y 255

-El valor de retorno se puede evaluar mediante la variable $?

-Reciben argumentos en las variables $1, $2, etc.

-Pasaje de parametros:

```bash
# Recibe 2 argumentos y devuelve:
# 1 si el primero es el mayor
# 0 en caso contrario

mayor() {
  echo "Se van a comparar los valores: $*"
  if [ $1 -gt $2 ]; then
    echo "$1 es el mayor"
    return 1
  fi
  echo "$2 es el mayor"
  return 0
}

mayor 5 6     # Invocación
echo $?       # Imprime el exit status de la función
```

12.Evaluación de expresiones.

a. Realizar un script que le solicite al usuario 2 números, los lea de la entrada Standard e imprima la multiplicación, suma, resta y cual es el mayor de los números leídos.

b. Modificar el script creado en el inciso anterior para que los números sean recibidos como parámetros. El script debe controlar que los dos parámetros sean enviados.

c. Realizar una calculadora que ejecute las 4 operaciones básicas: +, - ,*, %.

Esta calculadora debe funcionar recibiendo la operación y los números como parámetros

HECHO EN SHELL EN CARPETA EJERCICIOS

13.Uso de las estructuras de control:

a. Realizar un script que visualice por pantalla los números del 1 al 100 así como sus cuadrados.

b. Crear un script que muestre 3 opciones al usuario: Listar, DondeEstoy y QuienEsta. Según la opción elegida se le debe mostrar:

➢ Listar: lista el contenido del directorio actual.

➢ DondeEstoy: muestra la ruta deldirectorio donde me encuentro ubicado. ➢ QuienEsta: muestra los usuarios conectados al sistema.

c. Crear un script que reciba como parámetro el nombre de un archivo e informe si el mismo existe o no, y en caso afirmativo indique si es un directorio o un archivo. En caso de que no exista el archivo/directorio cree un directorio con el nombre recibido como parámetro.

HECHO EN SHELL EN CARPETA EJERCICIOS

14.Renombrando Archivos: haga un script que renombre solo archivos de un directorio pasado como parámetro, agregandole una CADENA, contemplando las opciones:

➢ "-a CADENA": renombra el fichero concatenando CADENA al final del nombre del archivo

➢ "-b CADENA": renombra el fichero concatenando CADENA al comienzo del nombre del archivo

Ejemplos:

Si tengo los siguientes archivos: /tmp/a /tmp/b , al ejecutar: **./renombra /tmp/ -aEJ** obtendré como resultado: /tmp/aEJ /tmp/bEJ. Y si ejecuto: **./renombra /tmp/ -b EJ** el resultado será: /tmp/EJa /tmp/EJb

HECHO EN SHELL EN CARPETA EJERCICIOS

15.El comando *cut* nos permite procesar las líneas de la entrada que reciba (archivo, entrada estándar, resultado de otro comando, etc) y cortar columnas o campos, siendo posible indicar cuál es el delimitador de las mismas. Investigue los parámetros que puede recibir este comando y cite ejemplos de uso.

-d [delimitador]→ Especifica el delimitador por el cual va a separar la fila (tab default)

-f [num_campo] [archivo] → Especifica los campos a extraer

-c [rango] [archivo] → Extrae un rango de caracteres especificos de una linea de texto

-b [rango] [archivo] → Extrae un rango de caracteres pero en bytes de una linea de texto

16.Realizar un script que reciba como parámetro una extensión y haga un reporte con 2 columnas, el nombre de usuario y la cantidad de archivos que posee con esa extensión. Se debe guardar el resultado en un archivo llamado *reporte.txt*

17.Escribir un script que al ejecutarse imprima en pantalla los nombre de los archivos que se encuentran en el directorio actual, intercambiando minúsculas por mayúsculas, además de eliminar la letra a (mayúscula o minúscula).

Por ejemplo, si en el directorio actual están los siguientes archivos:

➢ IsO

➢ pepE

➢ Maria

y ejecutó: ./ejercicio17 , se obtendrá como resultado:

➢ iSo

➢ PEPe

➢ mRI

Ayuda: Investigar el comando *tr*

18.Crear un script que verifique cada 10 segundos si un usuario se ha logueado en el sistema (el nombre del usuario será pasado por parámetro). Cuando el usuario finalmente se loguee, el programa deberá mostrar el mensaje ”Usuario XXX logueado en el sistema” y salir.

19.Escribir un Programa de “Menu de Comandos Amigable con el Usuario” llamado menú, el cual, al ser invocado, mostrará un menú con la selección para cada uno de los scripts creados en esta práctica. Las instrucciones de cómo proceder deben mostrarse junto con el menú. El menú deberá iniciarse y permanecer activo hasta que se seleccione Salir. Por ejemplo:

MENU DE COMANDOS

03. Ejercicio 3

12. Evaluar Expresiones

13. Probar estructuras de control

...

Ingrese la opción a ejecutar: 03

20.Realice un script que simule el comportamiento de una estructura de PILA e implemente las siguientes funciones aplicables sobre una estructura global definida en el script:

➔ push: Recibe un parámetro y lo agrega en la pila

➔ pop: Saca un elemento de la pila

➔ length: Devuelve la longitud de la pila

➔ print: Imprime todos elementos de la pila

Dentro del mismo script y utilizando las funciones implementadas:

1. Agregue 10 elementos a la pila

2. Saque 3 de ellos

3. Imprima la longitud de la pila

4. Luego imprima la totalidad de los elementos que en ella se encuentran.

21.Dada la siguiente declaración al comienzo de un script:

num=(10 3 5 7 9 3 5 4)

(la cantidad de elementos del arreglo puede variar).

Implemente la función **productoria** dentro de este script, cuya tarea sea multiplicar todos los números que el arreglo contiene.

22.Implemente un script que recorra un arreglo compuesto por números e imprima en pantalla sólo los números pares y que cuente sólo los números

impares y los informe en pantalla al finalizar el recorrido.

23.Dada la definición de 2 vectores del mismo tamaño y cuyas longitudes no se conocen.

vector1=(1 .. N)

vector2=(1.. N)

Por ejemplo:

vector1=(1 80 65 35 2) y vector2=(5 98 3 41 8).

Complete este script de manera tal de implementar la suma elemento a elemento entre ambos vectores y que la misma sea impresa en pantalla de la siguiente manera:

➢ La suma de los elementos de la posición 0 de los vectores es 6

➢ La suma de los elementos de la posición 1 de los vectores es 178 ...

➢ La suma de los elementos de la posición 4 de los vectores es 10

24.Realice un script que agregue en un arreglo todos los nombres de los usuarios del sistema pertenecientes al grupo “users”. Adicionalmente el script puede recibir como parámetro:

➢ “-b n”: Retorna el elemento de la posición n del arreglo si el mismo existe. Caso contrario, un mensaje de error.

➢ “-l”: Devuelve la longitud del arreglo

➢ “-i”: Imprime todos los elementos del arreglo en pantalla

25.Escriba un script que reciba una cantidad desconocida de parámetros al momento de su invocación (debe validar que al menos se reciba uno). Cada parámetro representa la ruta absoluta de un archivo o directorio en el sistema. El script deberá iterar por todos los parámetros recibidos, y solo para aquellos parámetros que se encuentren en posiciones impares (el primero, el tercero, el verificar si el archivo o directorio existen en el sistema, imprimiendo en pantalla

que tipo de objeto es (archivo o directorio). Además, deberá informar la cantidad de archivos o directorios inexistentes en el sistema.

26.Realice un script que implemente a través de la utilización de funciones las operaciones básicas sobre arreglos:

- ➢ inicializar: Crea un arreglo llamado array vacío
- ➢ agregar_elem <parametro1>: Agrega al final del arreglo el parámetro recibido
- ➢ eliminar_elem <parametro1>: Elimina del arreglo el elemento que se encuentra en la posición recibida como parámetro. Debe validar que se reciba una posición válida
- ➢ longitud: Imprime la longitud del arreglo en pantalla
- ➢ imprimir: Imprime todos los elementos del arreglo en pantalla
- ➢ inicializar_Con_Valores <parametro1><parametro2>: Crea un arreglo con longitud <parametro1>y en todas las posiciones asigna el valor <parametro2>

27.Realice un script que reciba como parámetro el nombre de un directorio. Deberá validar que el mismo exista y de no existir causar la terminación del script con código de error 4. Si el directorio existe deberá contar por separado la cantidad de archivos que en él se encuentran para los cuales el usuario que ejecuta el script tiene permiso de lectura y escritura, e informar dichos valores en pantalla. En caso de encontrar subdirectorios, no deberán procesarse, y tampoco deberán ser tenidos en cuenta para la suma a informar.

28.Implemente un script que agregue a un arreglo todos los archivos del directorio /home cuya terminación sea .doc. Adicionalmente, implemente las siguientes funciones que le permitan acceder a la estructura creada:

- ➢ verArchivo <nombre_de_archivo>: Imprime el archivo en pantalla si el mismo se encuentra en el arreglo. Caso contrario imprime el mensaje de error “Archivo no encontrado” y devuelve como valor de retorno 5 ➢

cantidadArchivos: Imprime la cantidad de archivos del /home con terminación .doc

➢ borrarArchivo <nombre_de_archivo>: Consulta al usuario si quiere eliminar el archivo lógicamente. Si el usuario responde Si, elimina el elemento solo del arreglo. Si el usuario responde No, elimina el archivo del arreglo y también del FileSystem. Debe validar que el archivo exista en el arreglo. En caso de no existir, imprime el mensaje de error “Archivo no encontrado” y devuelve como valor de retorno 10

29.Realice un script que mueva todos los programas del directorio actual (archivos ejecutables) hacia el subdirectorio “bin” del directorio HOME del usuario actualmente logueado. El script debe imprimir en pantalla los nombres de los que mueve, e indicar cuántos ha movido, o que no ha movido ninguno. Si el directorio “bin” no existe,deberá ser creado.

30.Implemente la estructura de datos Set (Conjunto de valores) en Bash. Un conjunto se define como una colección de valores únicos, es decir que solo almacena una vez cada valor, aún cuando se intente agregar el mismo valor más de una vez. La implementación debe soportar las siguientes operaciones mediante funciones:

- **initialize** - inicializa el set vacío.
- **initialize_with** - inicializa el set con un conjunto de valores que recibe como argumento (debe validar que se reciba al menos uno).
- **add** - Agrega un valor al conjunto, el cual recibe como argumento. No debe agregar elementos repetidos. El resultado de la operación será un éxito solo si el valor puede ser agregado al conjunto.
- **remove** - Elimina uno o más valores del conjunto, los cuales recibe como argumentos. Si la operación elimina al menos un valor, se considera un éxito. ● **contains** - Chequea si el conjunto contiene un valor recibido como argumento. El resultado será éxito si el valor está en el conjunto.
- **print** - Imprime los elementos del conjunto, de a uno por línea.

● **print_sorted** - Imprime los elementos del conjunto, de a uno por línea y ordenados alfabéticamente. *Tip: Investigar cómo combinar el comando sort con la función print.*

En un script separado, incorporar y utilizar las funciones implementadas para desarrollar un juego de bingo. El bingo deberá generar números aleatorios dentro de un rango entre 0 y un valor máximo que puede especificarse mediante un argumento del script, de manera opcional. El valor máximo no puede ser 0 ni superior a 32767, y en caso de no especificarse se tomará como valor por defecto 99. En cada ronda se generará un nuevo número que ya no haya sido utilizado y se lo cantará, imprimiendo en la salida estándar. Luego de esto, se esperará entrada del usuario para saber si se debe cantar "BINGO" para finalizar la partida o se debe cantar un nuevo numero. Al finalizar, el script debera imprimir en orden los números se cantaron hasta que se produjo el bingo.

Tip: Investigar la variable de entorno $RANDOM de Bash para obtener valores aleatorios.

31.Realice un script que reciba como argumento una lista de posibles nombres de usuarios del sistema y, para cada uno de los que efectivamente existan en el sistema y posean un directorio personal configurado que sea válido, realice las modificaciones necesarias en su directorio personal para que tenga un subdirectorio llamado "directorio_iso" con la siguiente estructura:

```
directorio_iso
├── 2025
    │ ├── 01
    │ │ └── archivo.txt
    │ ├── 02
    │ │ └── archivo.txt
    │ ├── 03
    │ │ └── archivo.txt
    │ ├── 04
```

```
│ │ └── archivo.txt
│ ├── 05
│ │ └── archivo.txt
│ ├── 06
│ │ └── archivo.txt
│ ├── 07
│ │ └── archivo.txt
│ ├── 08
│ │ └── archivo.txt
│ ├── 09
│ │ └── archivo.txt
│ ├── 10
│ │ └── archivo.txt
│ ├── 11
│ │ └── archivo.txt
│ └── 12
│ └── archivo.txt
└── 2026
 ├── 01
 │ └── archivo.txt
 ├── 02
 │ └── archivo.txt
 ├── 03
 │ └── archivo.txt
 ├── 04
 │ └── archivo.txt
 ├── 05
```

```
│ └── archivo.txt
├── 06
│ └── archivo.txt
├── 07
│ └── archivo.txt
├── 08
│ └── archivo.txt
├── 09
│ └── archivo.txt
├── 10
│ └── archivo.txt
├── 11
│ └── archivo.txt
└── 12
  └── archivo.txt
```