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

Un Shell Script es un programa que está creado con instrucciones que son ejecutadas por un Shell (CLI o intérprete de comandos) de Unix o Linux. El código no es compilado ni precompilado, se va ejecutando línea por línea efectuando lo que cada instrucción le indica. Necesita un programa que entienda los comandos y estructuras que contiene y esto se suele poner en la primera línea del programa. Por ejemplo #!/bin/bash significa que le pasaremos al BASH las lineas del fichero de Script. Dado que el BASH es el intérprete de comandos más famoso de Linux, los Script que se crean para este entorno también se pueden llamar Bash Script.

Principalmente sirve para automatizar tareas y para realizar procesos más complejos de los que un solo comando puede efectuar. Aunque los comando se pueden enlazar mediante tuberías o XARGS, a veces necesitamos tomar decisiones condicionales o recorrer elementos mediante bucles. Aquí es donde necesitamos organizarlo todo en un Shell Script.

El propio sistema de Linux tiene programadas multitud de tareas con sus Script del sistema, desde la rotación de logs, actualización del arranque, gestión de servidores, niveles de ejecución etc.

1. **Investigar la funcionalidad de los comandos echo y read**

echo: imprime el texto.

read: lee una línea desde entrada estándar.

1. **¿Como se indican los comentarios dentro de un script?** Con #
2. **¿Cómo se declaran y se hace referencia a variables dentro de un script?**



**3. Crear dentro del directorio personal del usuario logueado un directorio llamado practicashell-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 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**

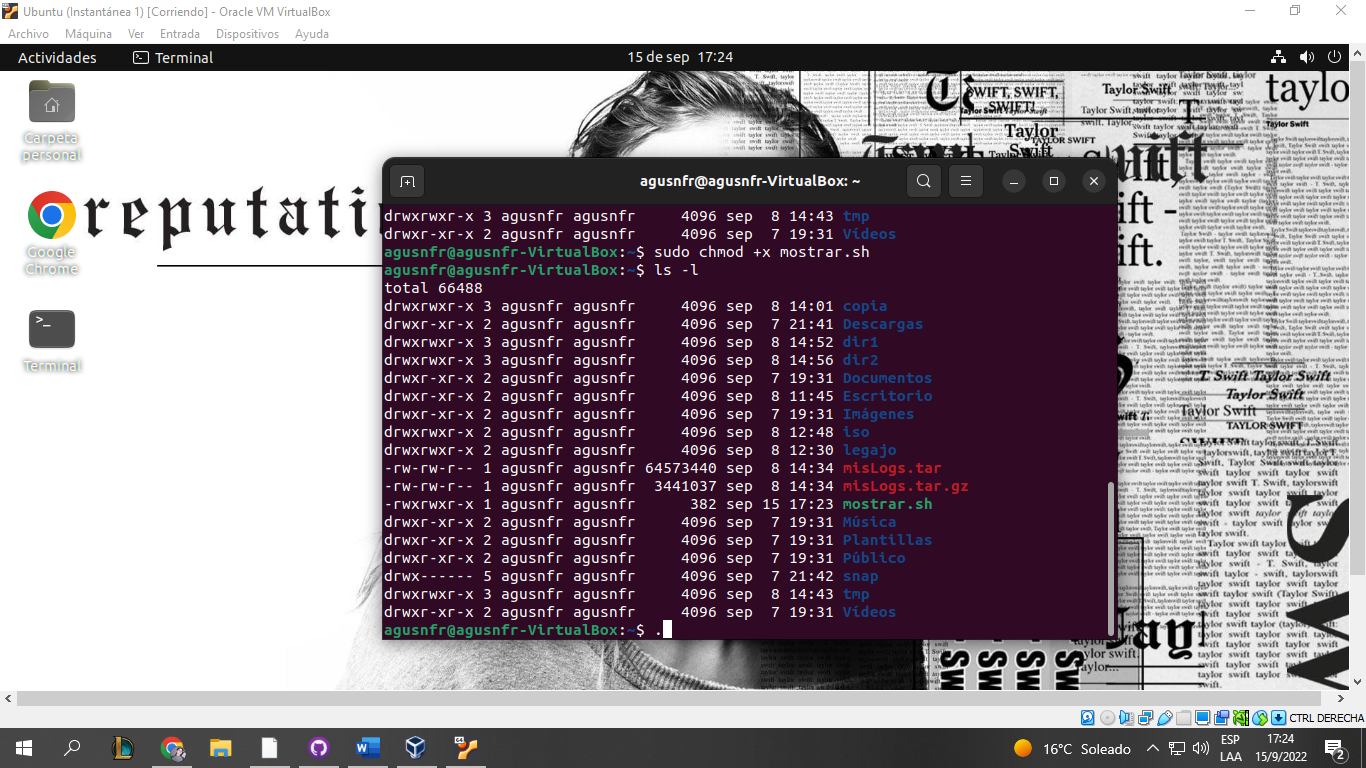
**echo "Su apellido y nombre es: “ 🡪** falta una comilla aca

**echo "$apellido $nombre"**

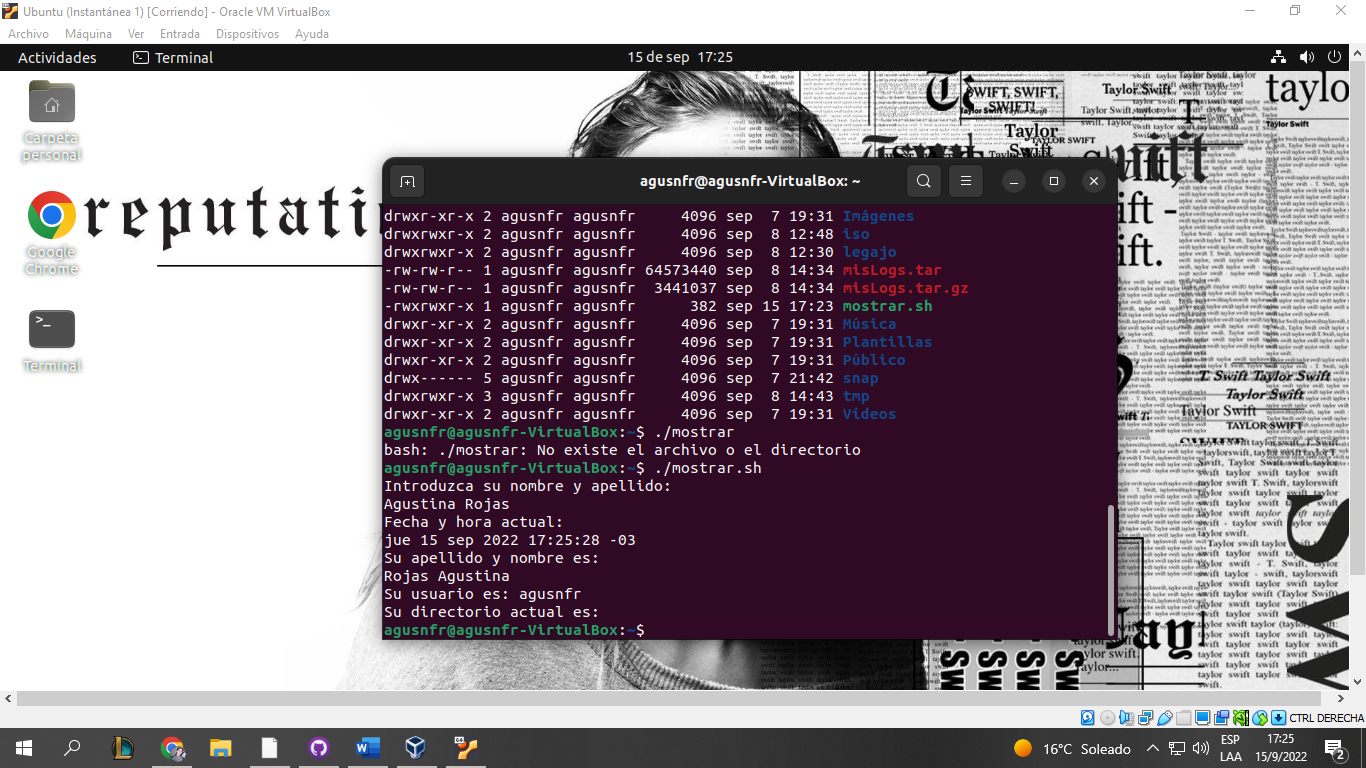
**echo "Su usuario es: `whoami`"**

**echo "Su directorio actual es:"**

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



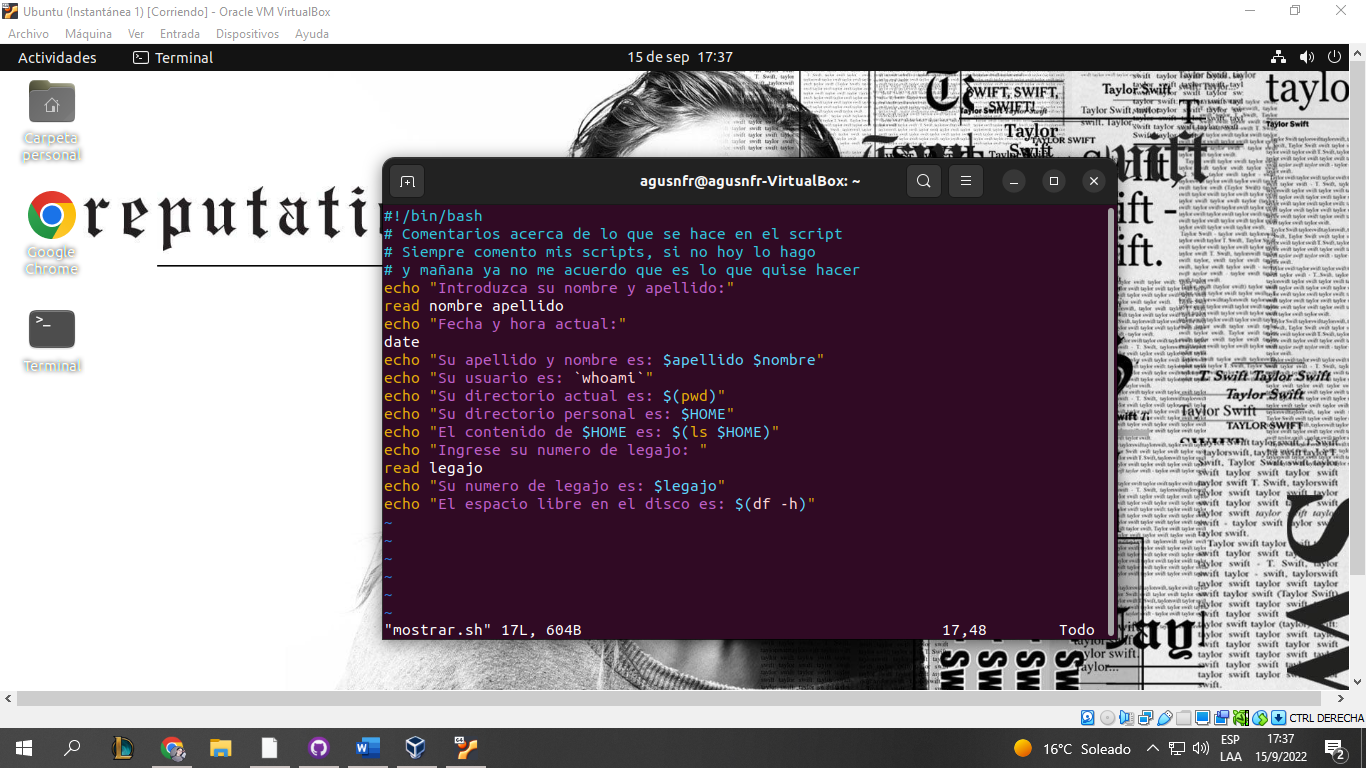
1. **Ejecutar el archivo creado de la siguiente manera: ./mostrar**
2. **¿Qué resultado visualiza?**



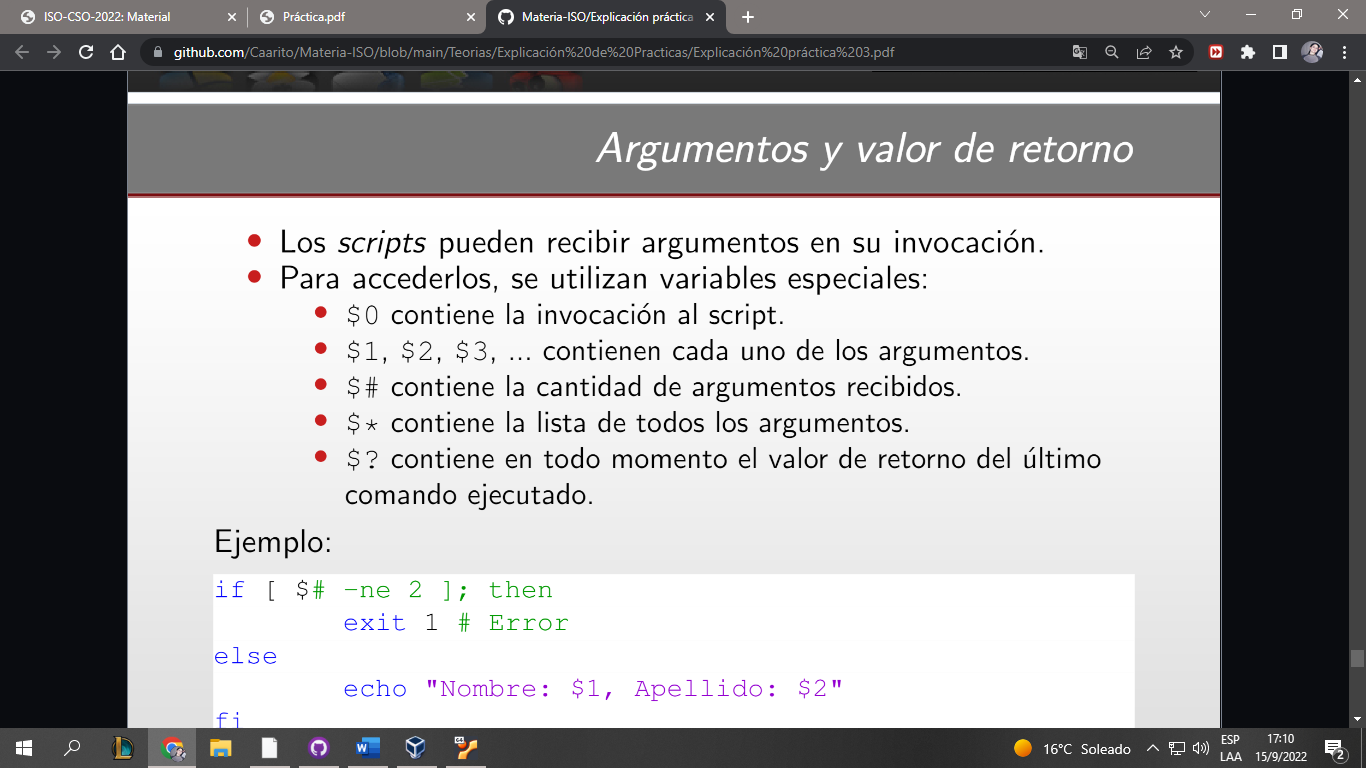
1. **Las backquotes (`) entre el comando whoami ilustran el uso de la sustitución de comandos. ¿Qué significa esto?**

Permite utilizar la salida de un comando como si fuese una cadena de texto normal, es equivalente a hacer $(whoami)

1. **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.**

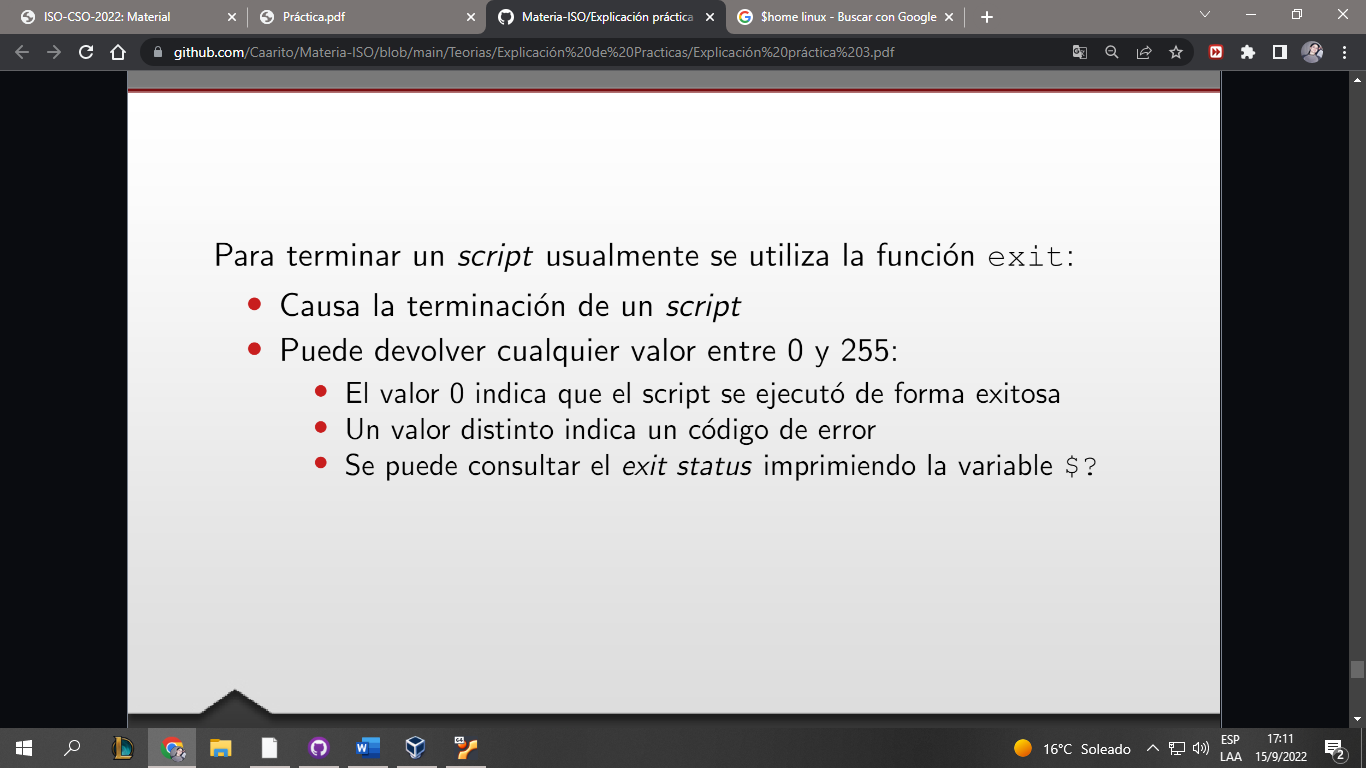


**4. Parametrización: ¿Cómo se acceden a los parámetros enviados al script al momento de su invocación? ¿Qué información contienen las variables $#, $\*, $? Y $HOME dentro de un script?**



$HOME es el directorio de trabajo por defecto del usuario.

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



**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.**

Este comando devuelve en su salida estándar el resultado de las expresiones aritméticas pasadas como argumentos. Su sintaxis es expr expresión.

Todos los elementos de la expresión deben ir separados por al menos un espacio, y ciertos operadores aritméticos llevan como prefijo una barra invertida para evitar toda confusión con los caracteres especiales del shell.

Operaciones básicas: suma, resta, multiplicación, división y módulo en números enteros.

Otras: evaluación de expresiones regulares, operaciones de cadena como subcadena, longitud de cadenas, etc.

Los operadores aritméticos son:

+: suma

-: resta

\\*: multiplicación

/: división entera

%: resto de la división entera o módulo

\( y \): paréntesis

Se utiliza generalmente una sustitución de comandos para asignar el resultado del comando expr a una variable. Se obtiene por ejemplo:

[agusnfr]$ expr 2 + 3

5

[agusnfr]$ expr 2 - 3

-1

[agusnfr]$ expr 2 + 3 \\* 4

14

[agusnfr]$ expr \( 2 + 3 \) \\* 4

20

[agusnfr]$ resultado=$(expr 9 / 2)

[agusnfr]$ echo $resultado

4

[agusnfr]$ expr $resultado % 3

1

<https://linuxhint.com/linux-expr-command/> mas info ahí

**7. El comando “test expresión” 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 que 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.**

Comando test

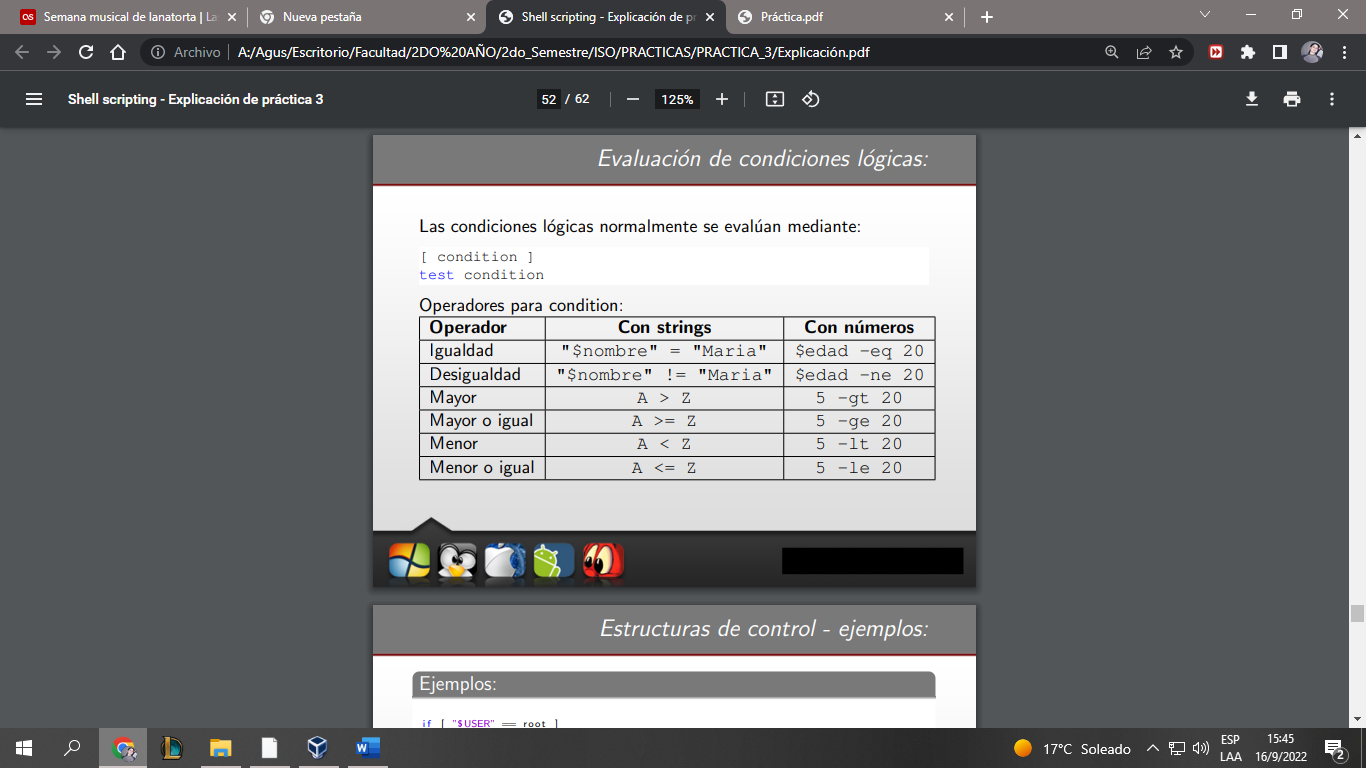
El comando test permite efectuar una serie de pruebas sobre los archivos, las cadenas de caracteres, los valores aritméticos y el entorno de usuario.

Este comando tiene un código de retorno igual a cero cuando el test es positivo, y diferente de cero en caso contrario; esto permite utilizarlo en encadenamientos de comandos con ejecución condicional (&& y ||) o en las estructuras de control que veremos más adelante.

El comando test posee dos sintaxis: test expresión y [ expresión ], donde "expresión" representa el test que se debe efectuar.

Los espacios detrás del corchete de apertura y antes del corchete de cierre son obligatorios en la sintaxis [ expresión ]. En general, todos los elementos de sintaxis del comando test deben ir separados por al menos un espacio.

El resto de la sección presenta los principales operadores que componen las expresiones de test del comando.



<https://francisconi.org/linux/comandos/test>

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

* **if**
* **case**
* **while**
* **for**
* **select**

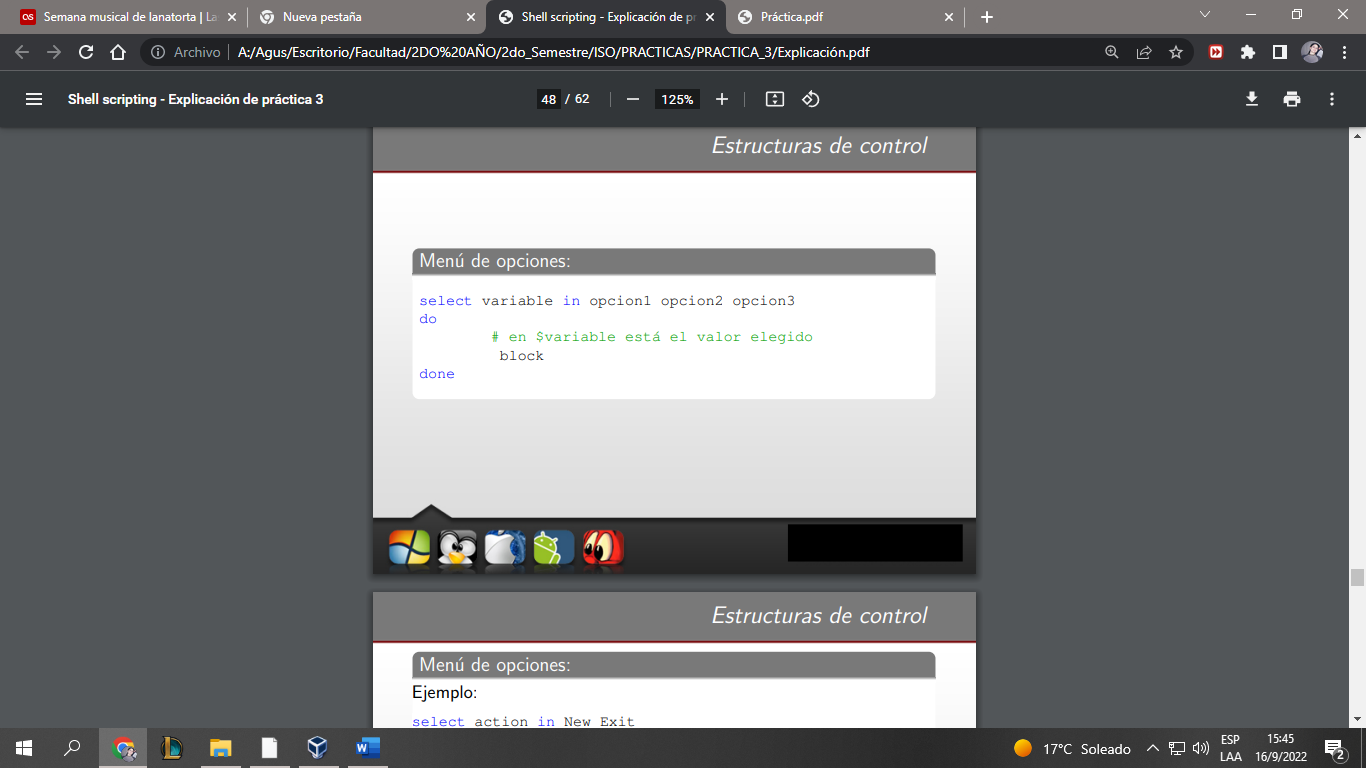
**IF**



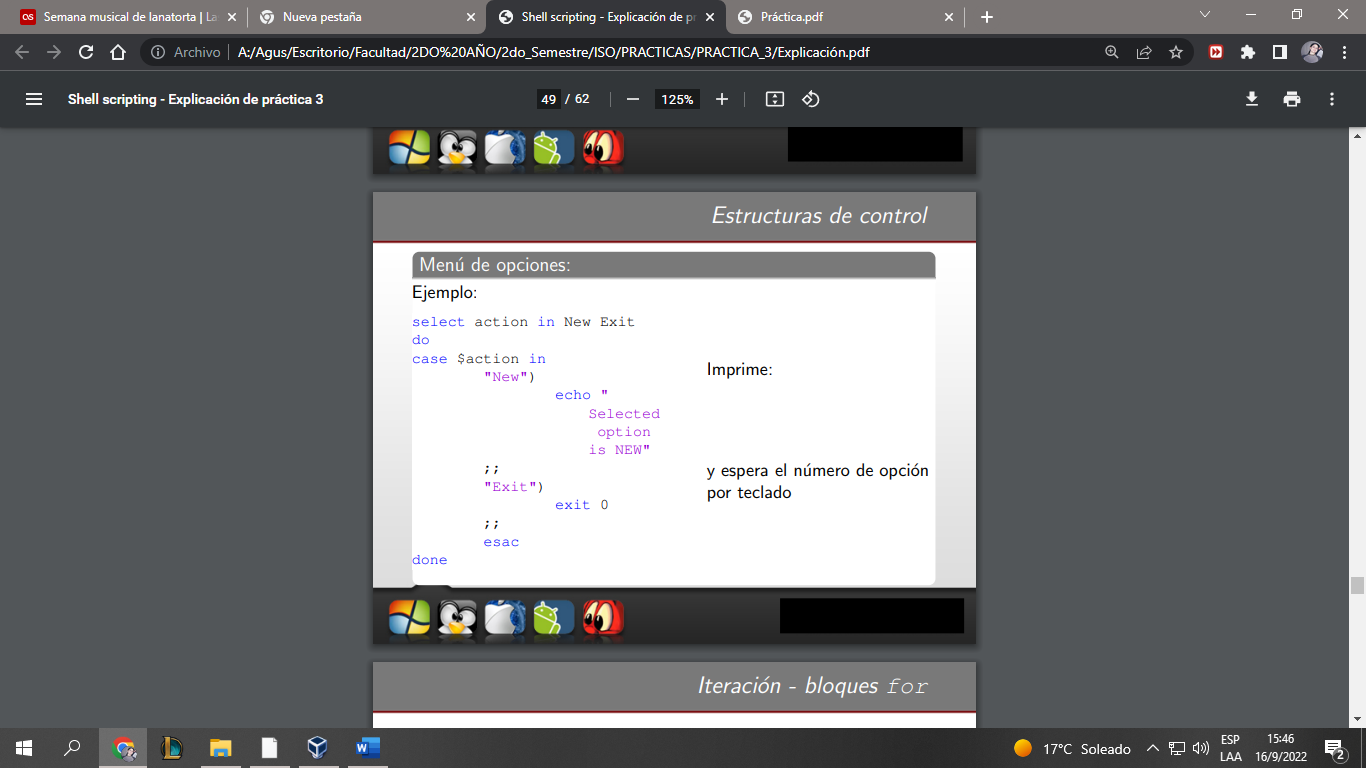
**CASE**



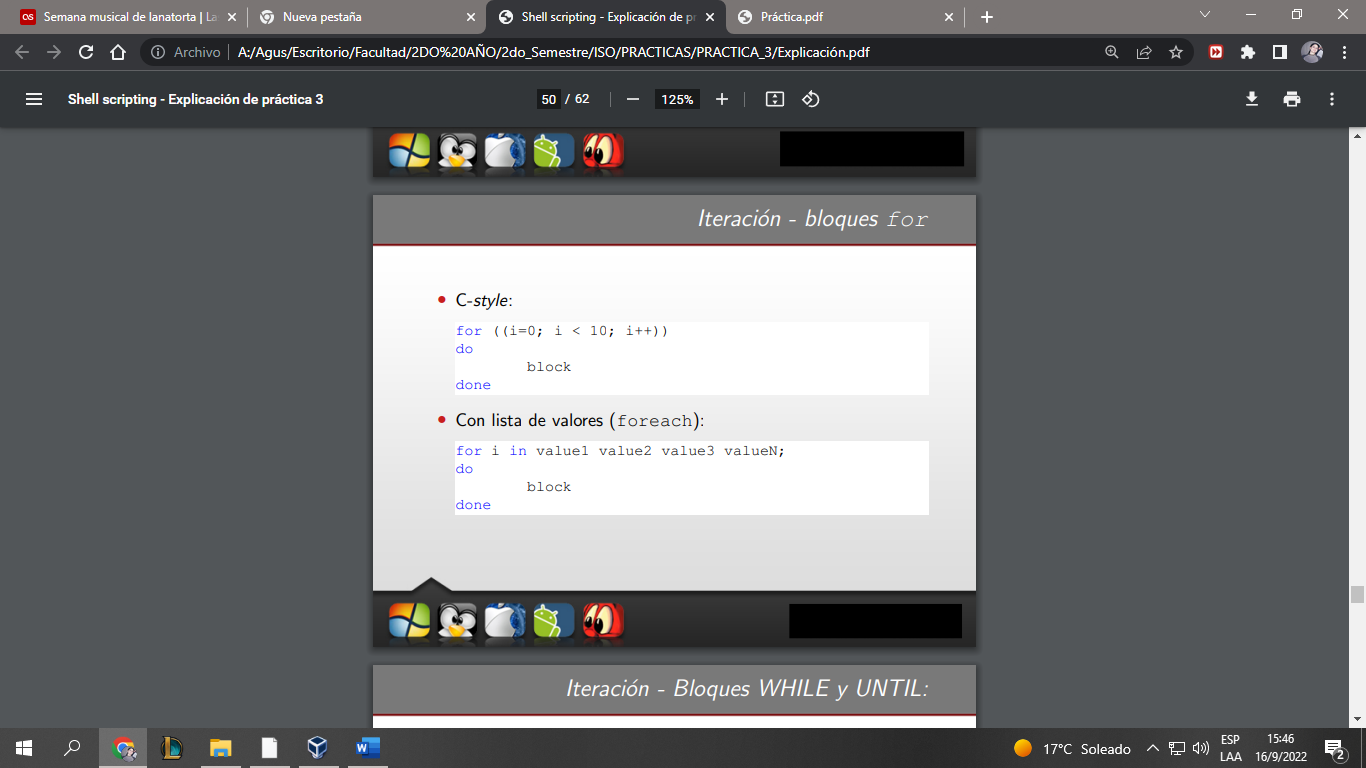
SELECT



<https://noviello.it/es/como-usar-select-en-bash-en-linux/>



**FOR**



**WHILE**



**UNTIL**



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

break [n] corta la ejecución de n niveles de loops.

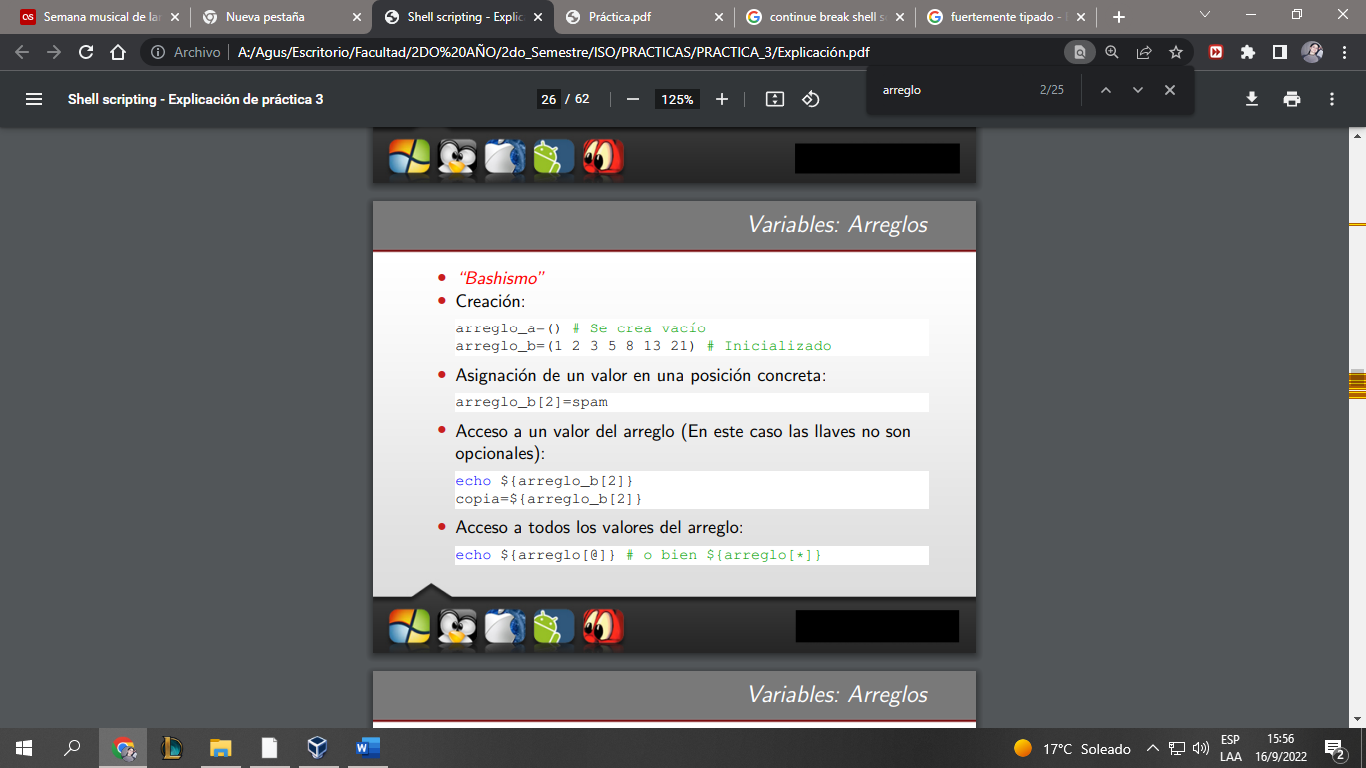
continue [n] salta a la siguiente iteración del enésimo loop que contiene esta instrucción.

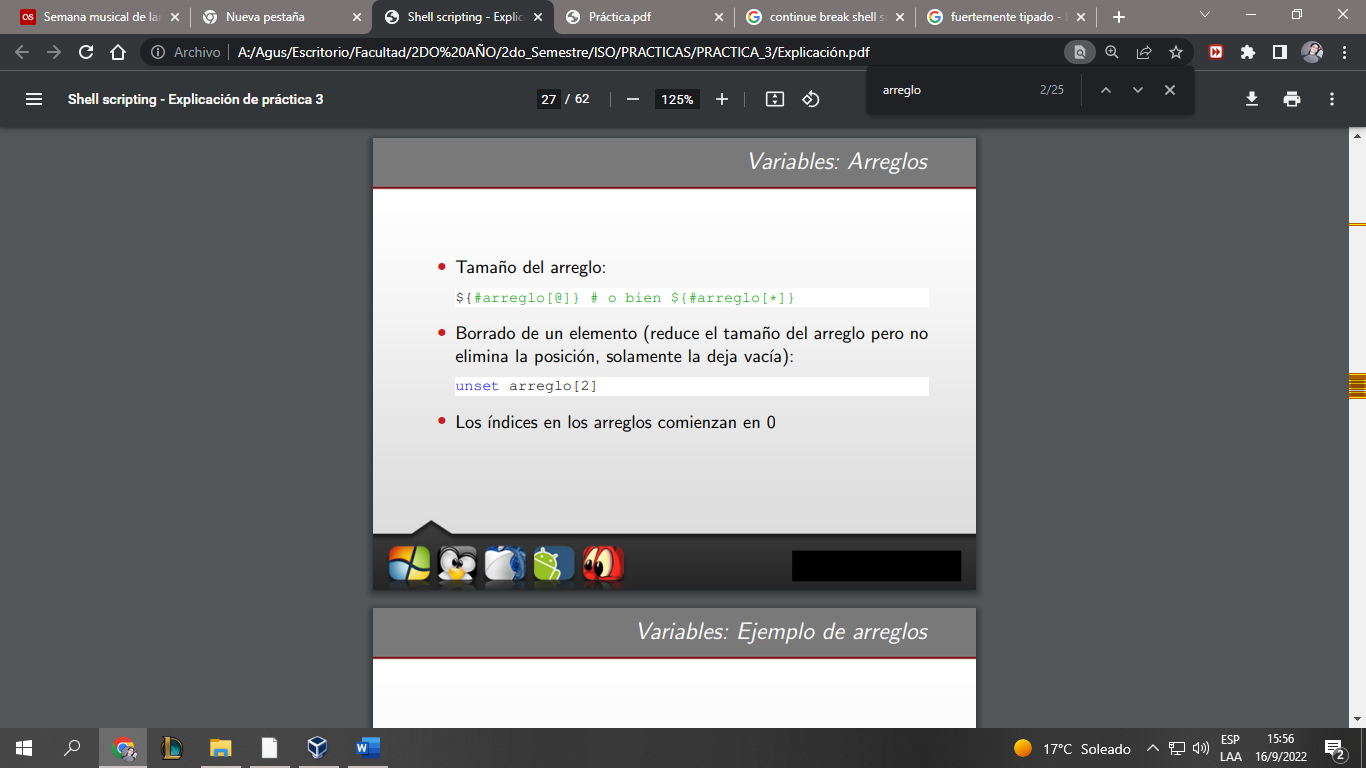
**10. ¿Qué tipo de variables existen? ¿Es shell script fuertemente tipado? ¿Se pueden definir arreglos? ¿Cómo?**

Las variables son de tipo string o arreglos.

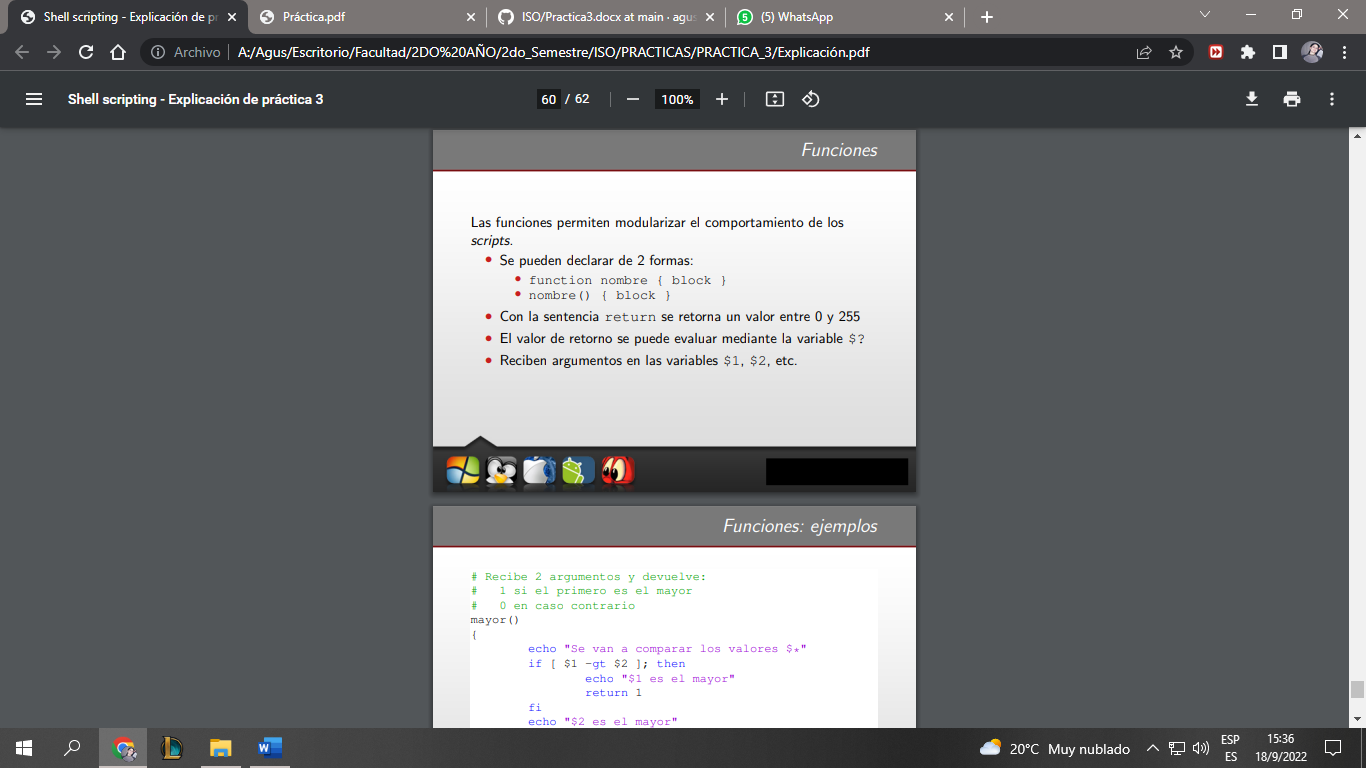
El shell es un lenguaje débilmente tipado (las variables en un idioma están fuertemente tipadas si tienen tipos explícitos; las variables sin tipos débiles son lenguajes débilmente tipados) y son diferentes de C. La definición de variables en la programación de shell no requiere un tipo, y no existe un concepto de tipo.

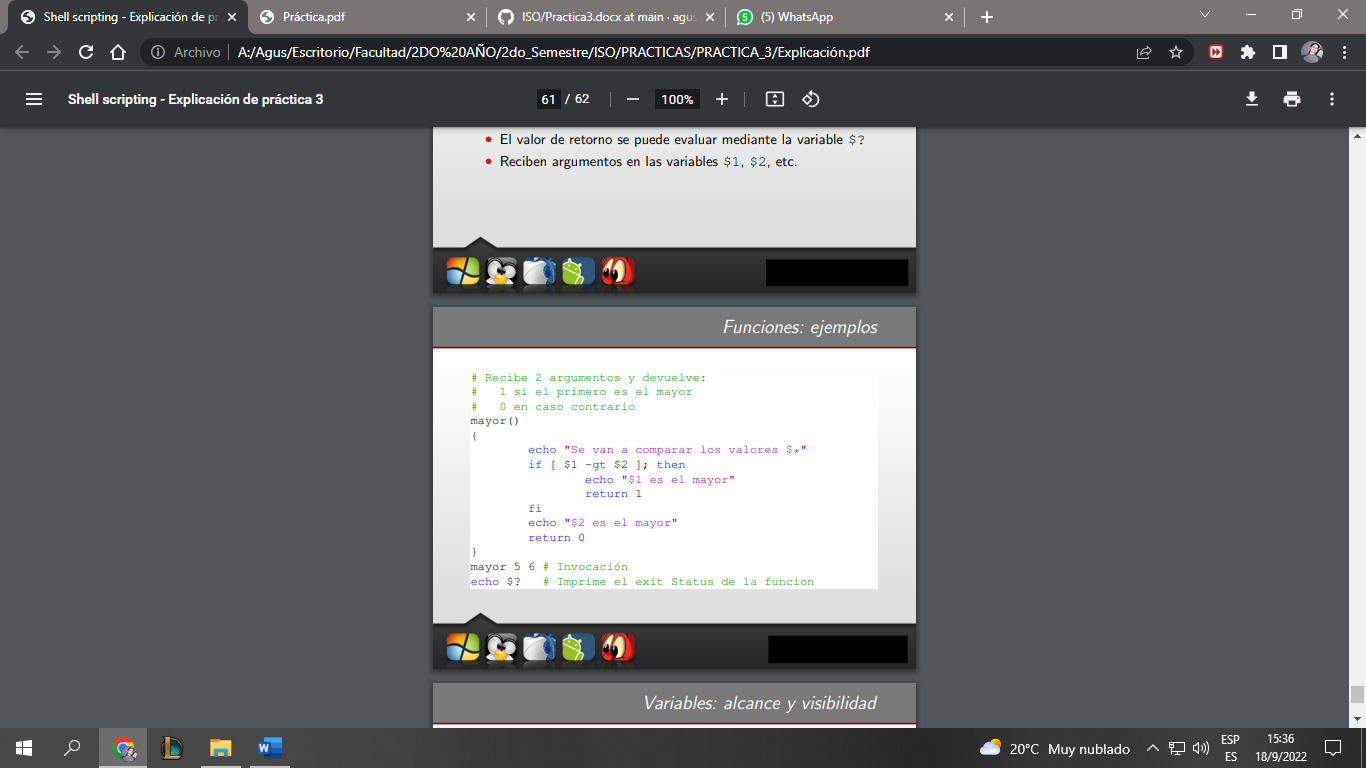
Si se puede definir arreglos.





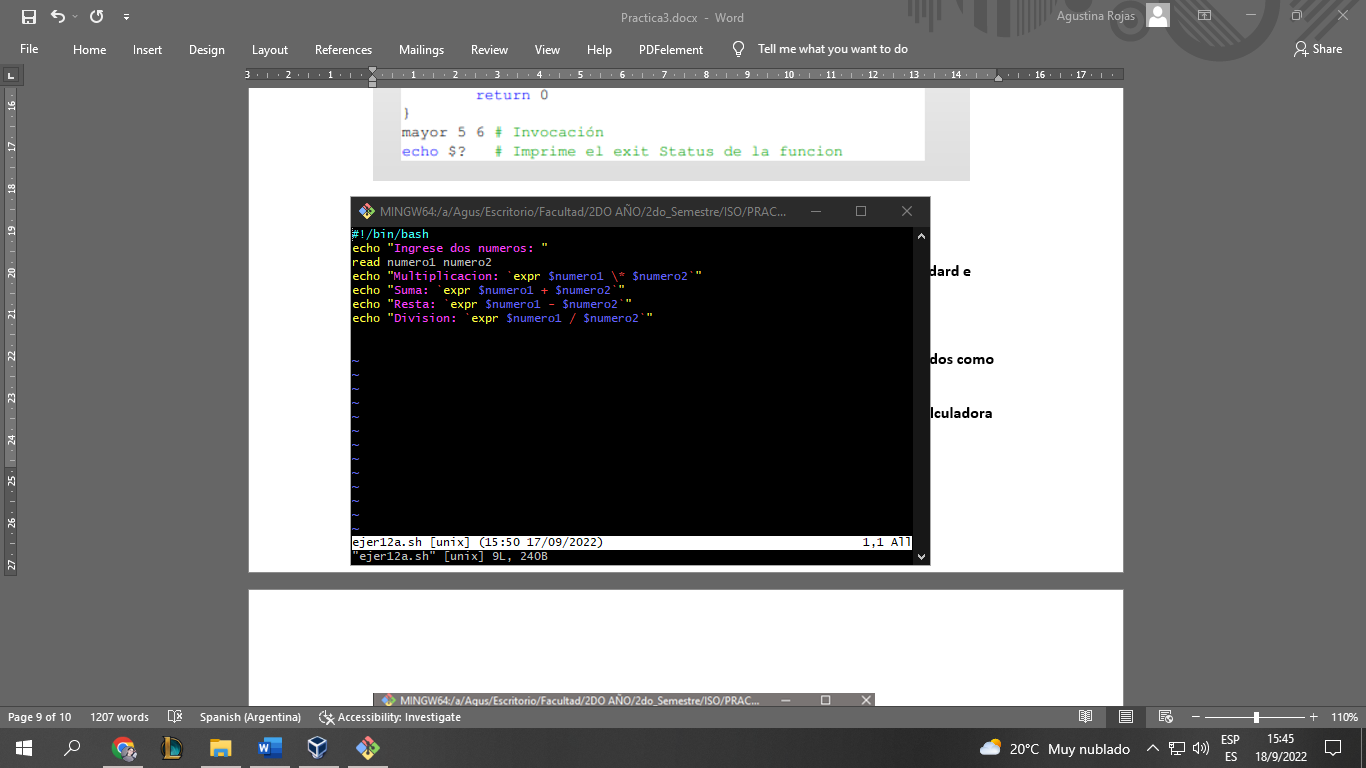
**11. ¿Pueden definirse funciones dentro de un script? ¿Cómo? ¿Cómo se maneja el pasaje de parámetros de una función a la otra?**



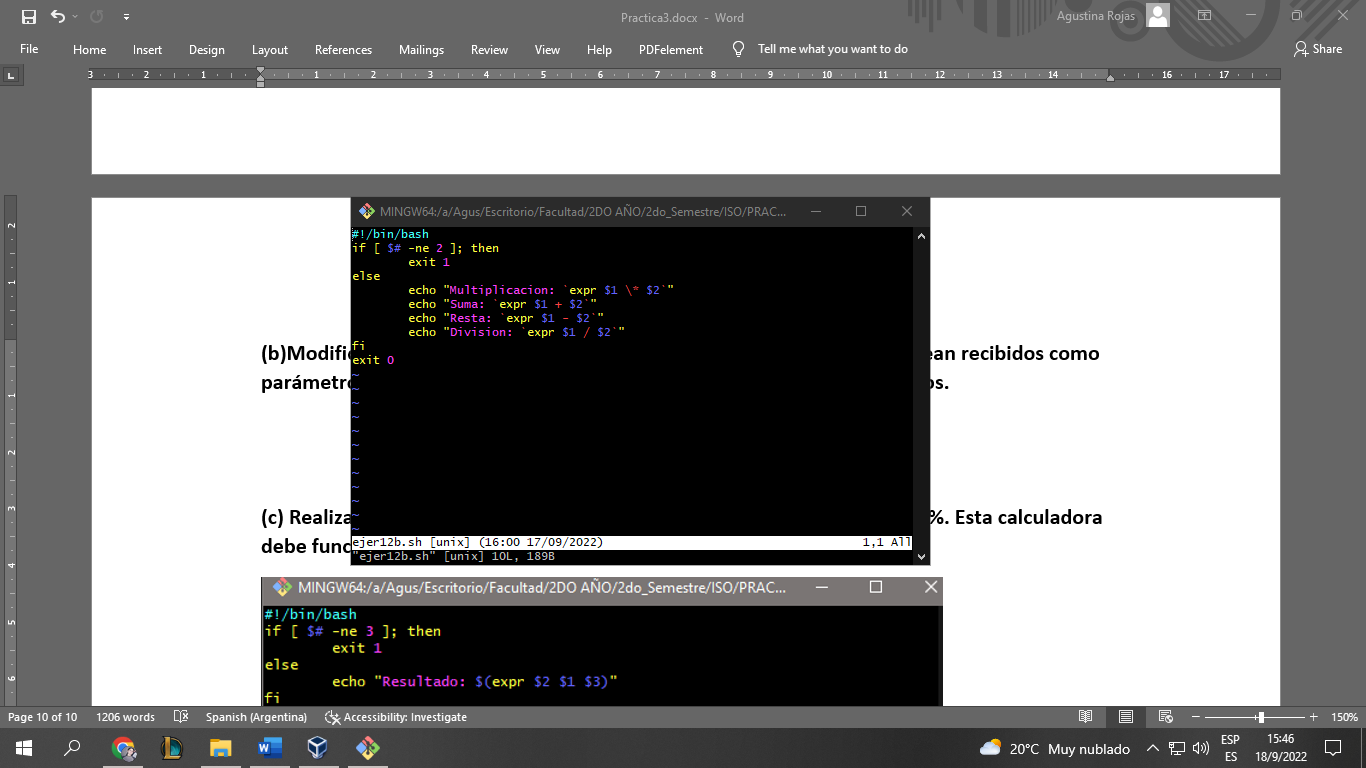


**12. Evaluación de expresiones:**

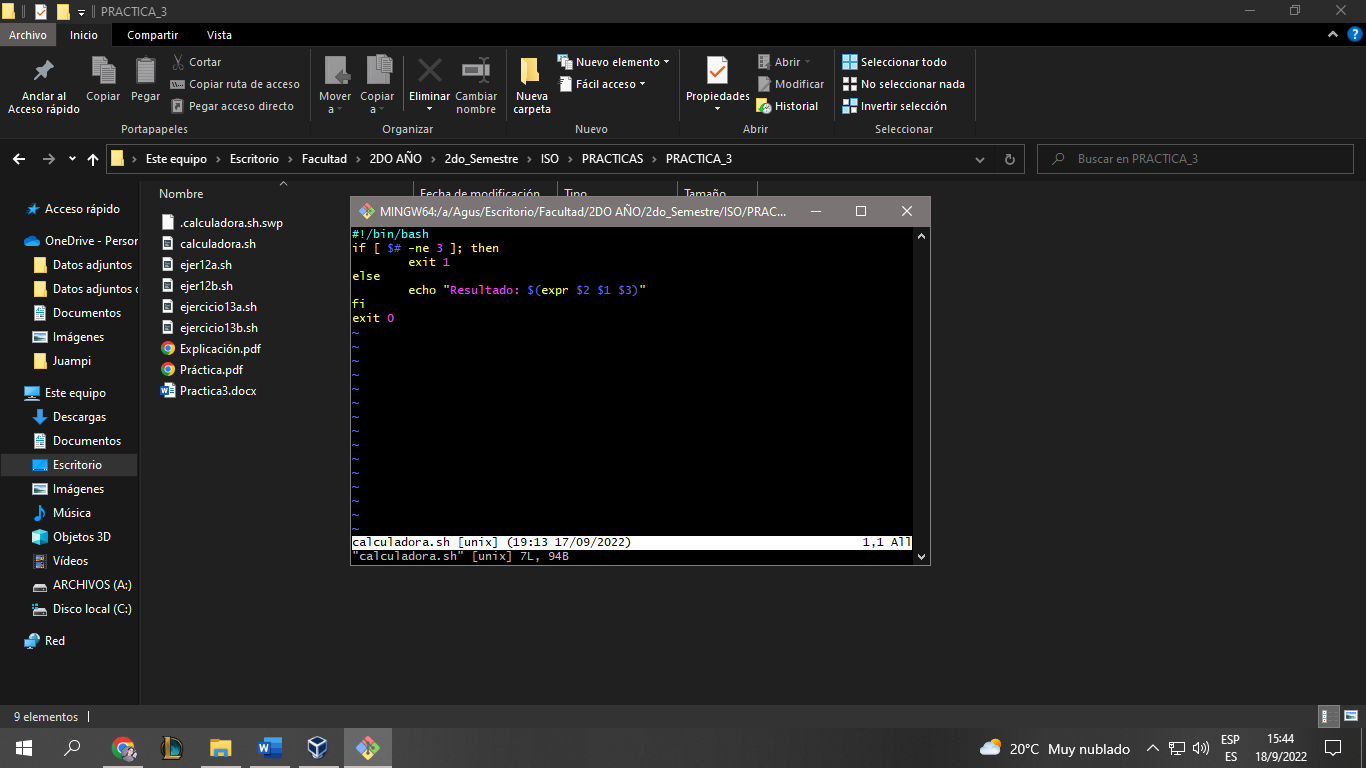
**(a) Realizar un script que le solicite al usuario 2 números, los lea de la entrada Standard e imprima la multiplicación, suma, resta y cual es el mayor de los números leídos.**



**(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.**

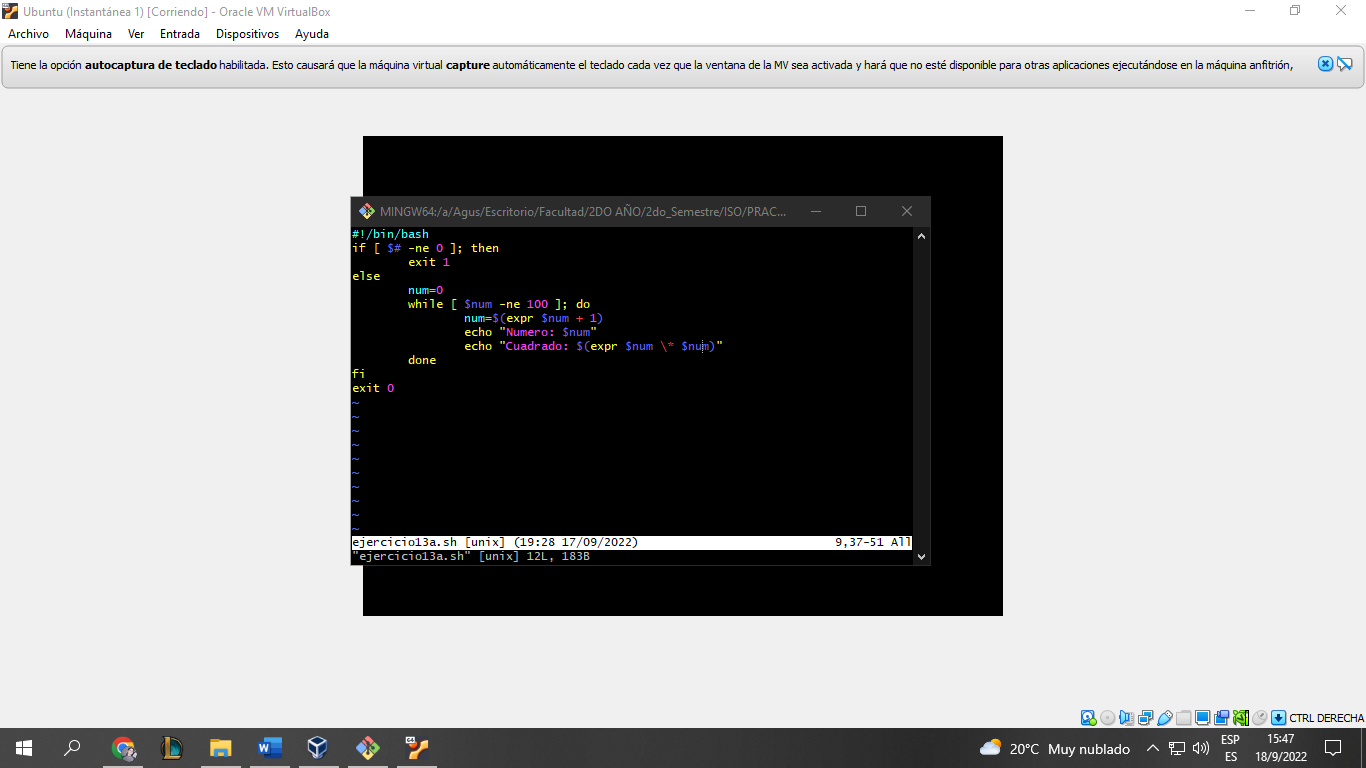


**(c) Realizar una calculadora que ejecute las 4 operaciones básicas: +, - ,\*, %. Esta calculadora debe funcionar recibiendo la operación y los números como parámetros**



**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.**

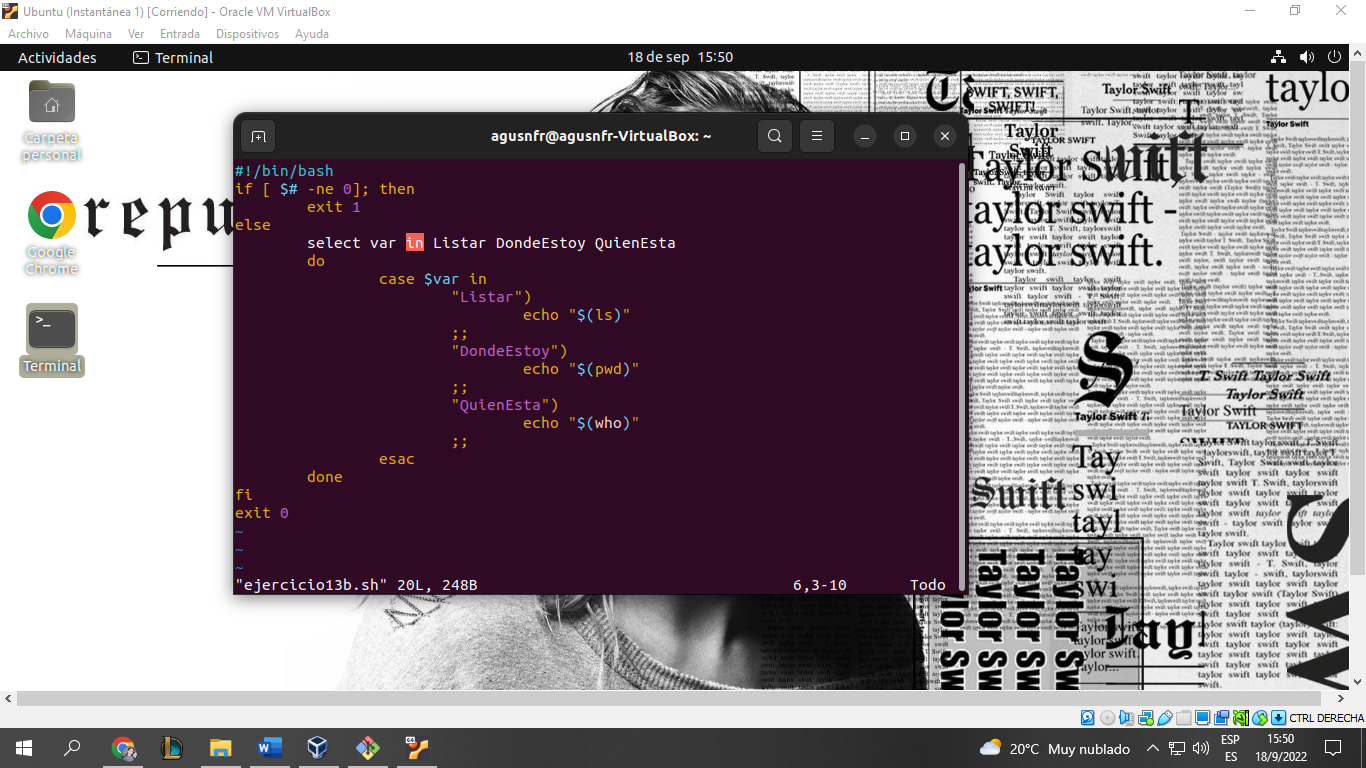


**(b) Crear un script que muestre 3 opciones al usuario: Listar, DondeEstoy y QuienEsta. Según la opción elegida se le debe mostrar:**

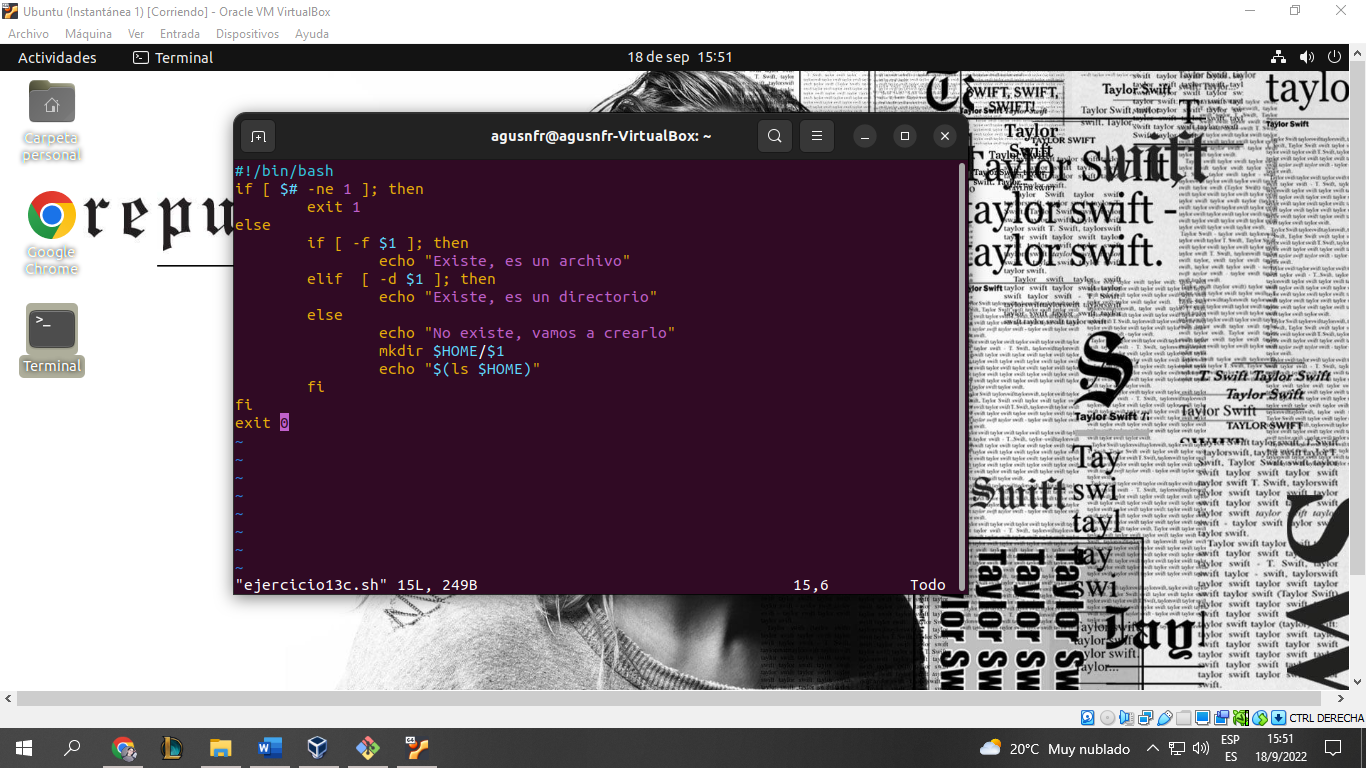
**Listar: lista el contenido del directoria actual.**

**DondeEstoy: muestra el directorio donde me encuentro ubicado.**

**QuienEsta: muestra los usuarios conectados al sistema.**



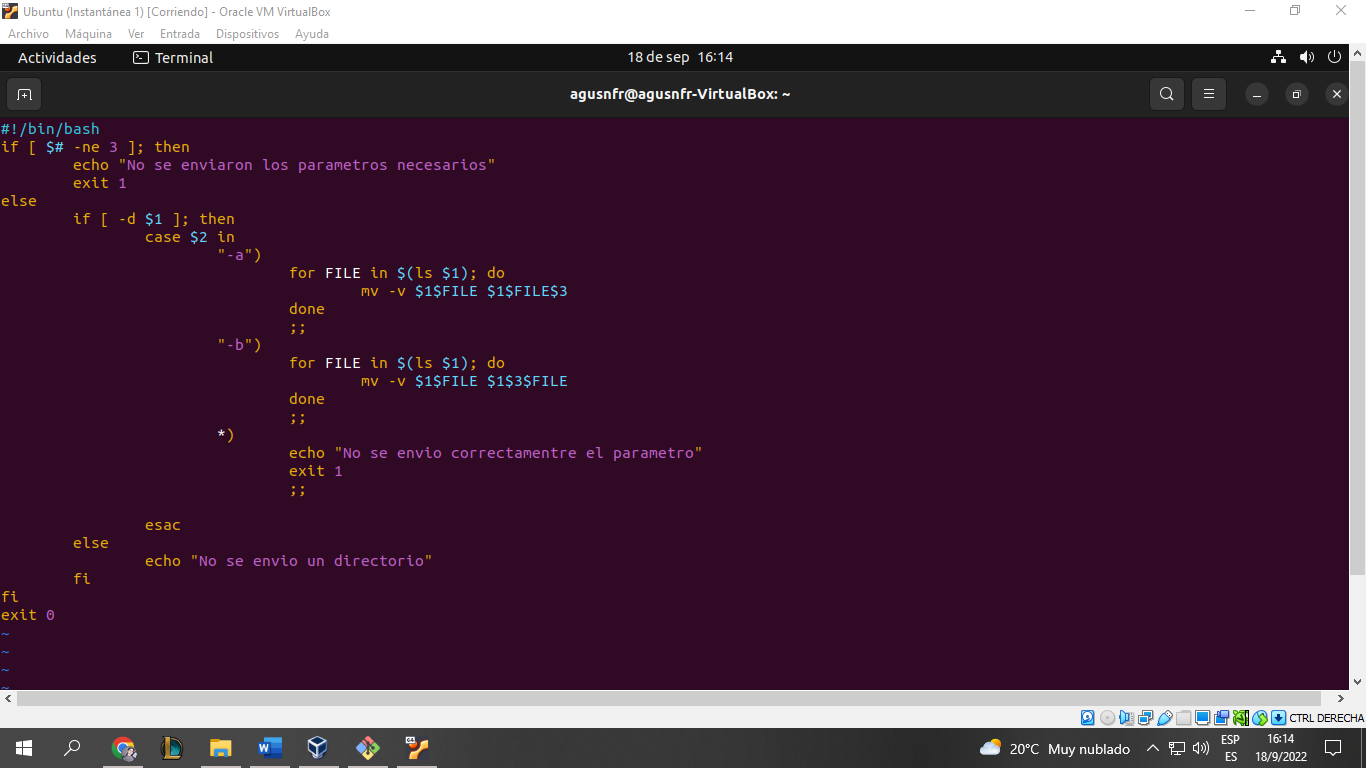
1. **Crear un script que reciba como parámetro el nombre de un archivo e informe si el mismo existe o no, y en caso afirmativo indique si es un directorio o un archivo. En caso de que no exista el archivo/directorio cree un directorio con el nombre recibido como parámetro.**



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

**“-a CADENA”: renombra el fichero concatenando CADENA al final del nombre del archivo**

**“-b CADENA”: renombra el fichero concantenado CADENA al principio del nombre del archivo**

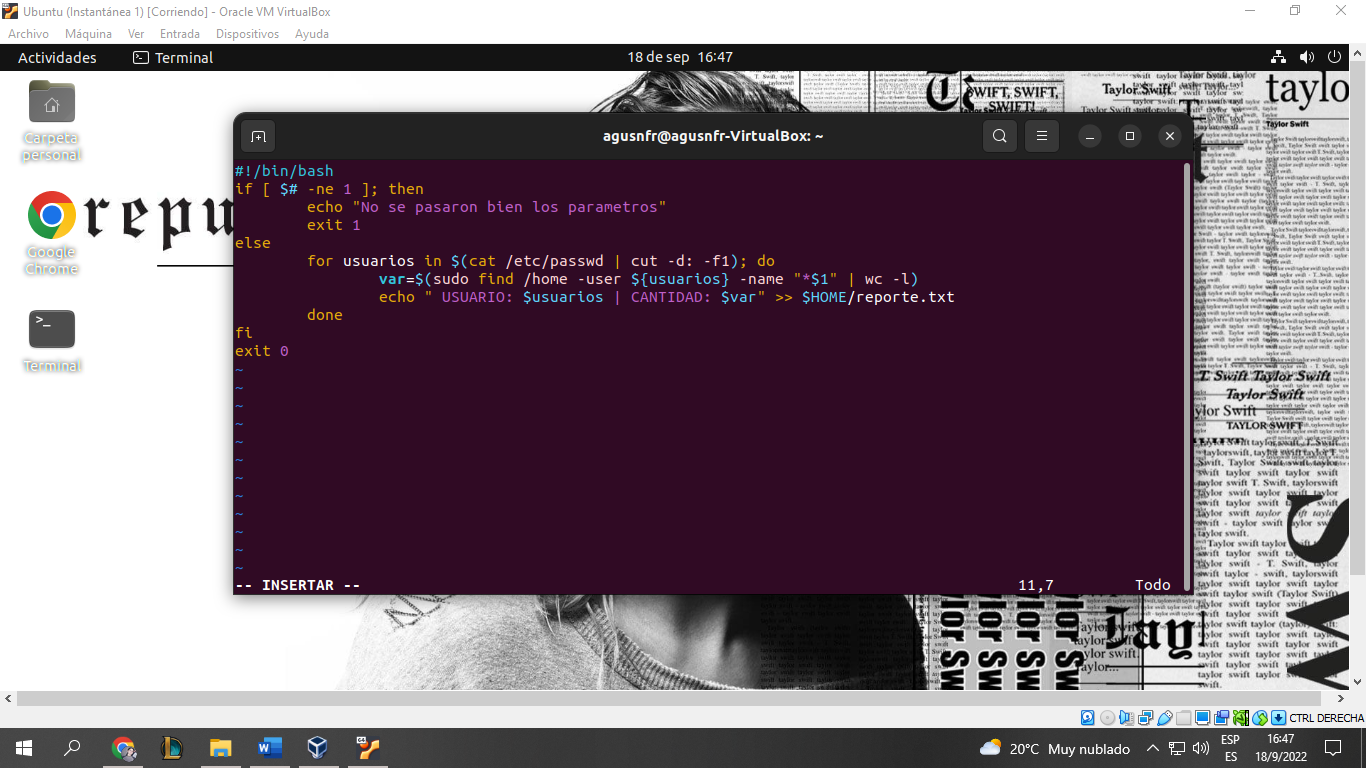


**15. Comando cut. El comando cut nos permite procesar la líneas de la entrada que reciba (archivo, entrada estándar, resultado de otro comando, etc) y cortar columnas o campos, siendo posible indicar cual es el delimitador de las mismas. Investigue los parámetros que puede recibir este comando y cite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

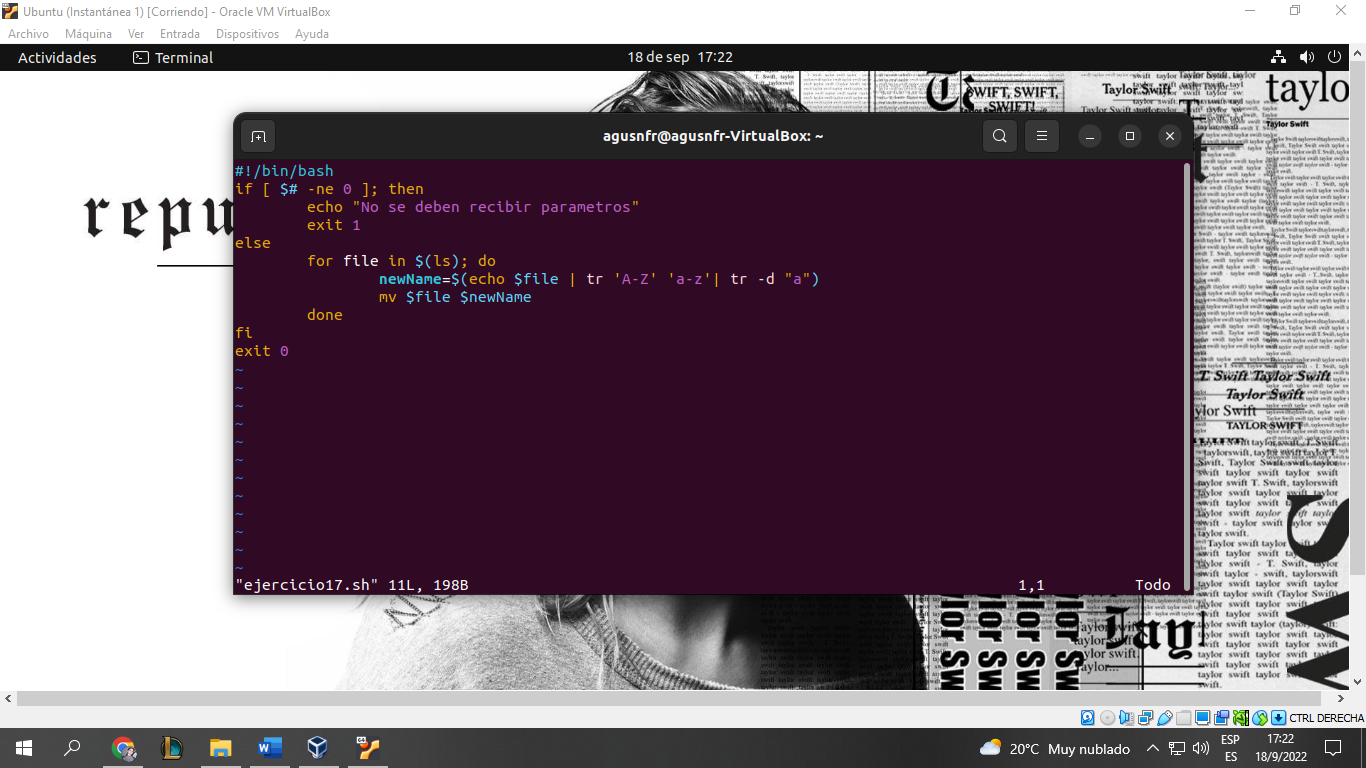
[https://geekland.eu/uso-del-comando-cut-en-linux-y-unix-con-ejemplos/#:~:text=%C2%BFQU%C3%89%20HACE%20EXACTAMENTE%20LA%20UTILIDAD,de%20un%20fichero%20de%20texto](https://geekland.eu/uso-del-comando-cut-en-linux-y-unix-con-ejemplos/" \l ":~:text=%C2%BFQU%C3%89%20HACE%20EXACTAMENTE%20LA%20UTILIDAD,de%20un%20fichero%20de%20texto).

<https://www.ochobitshacenunbyte.com/2019/06/09/ejemplos-practicos-del-comando-cut-en-linux/>

**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´**

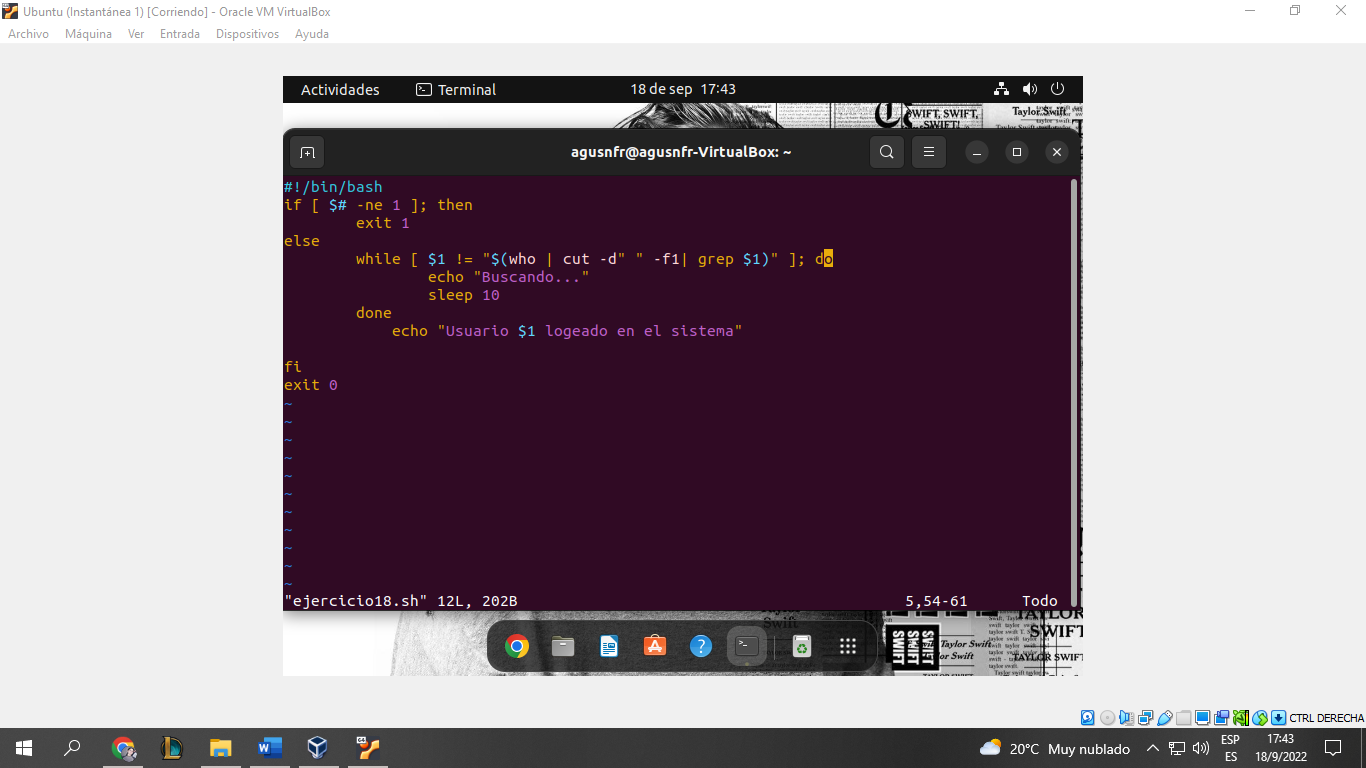


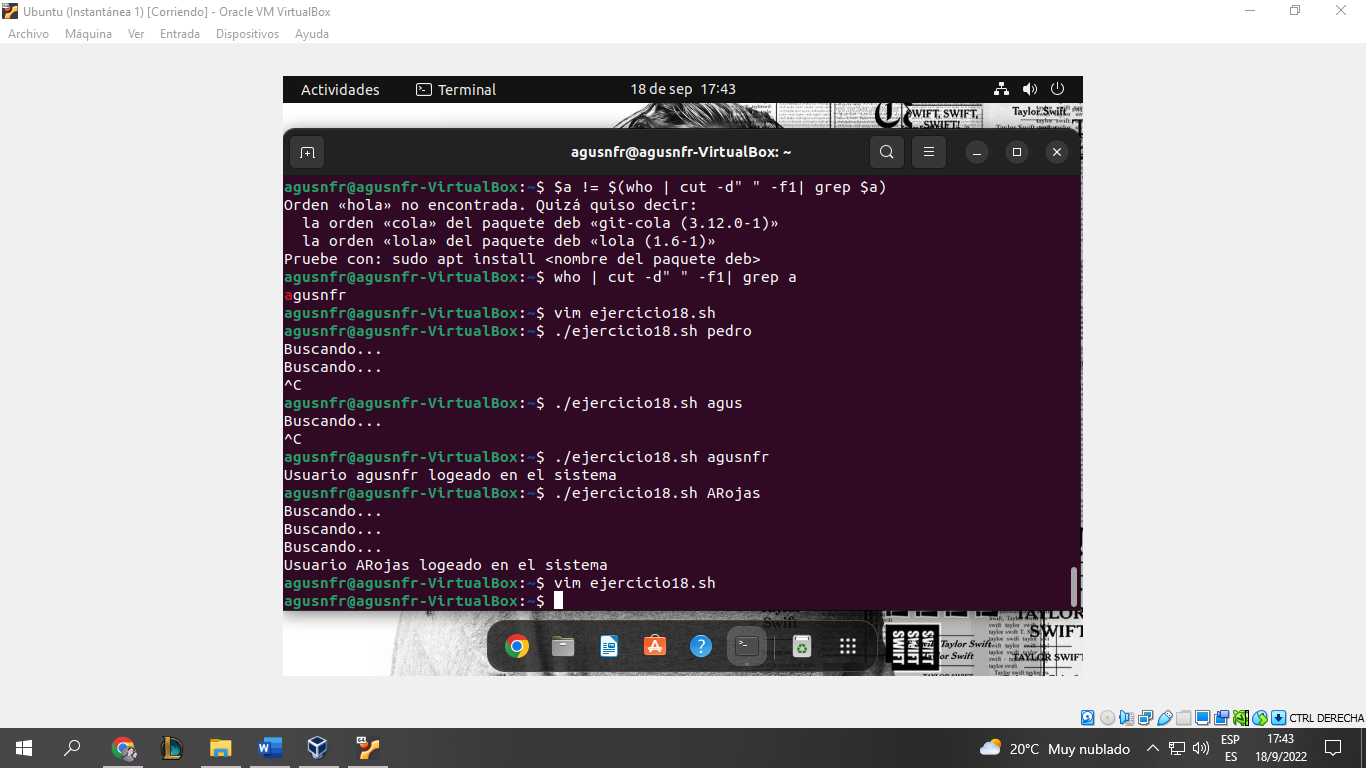
**17. Escribir un script que al ejecutarse imprima en pantalla los nombre de los archivos que se encuentran en el directorio actual, intercambiando minúsculas por mayúsculas, además de eliminar la letra a (mayúscula o minúscula). Ejemplo, directorio actual:**



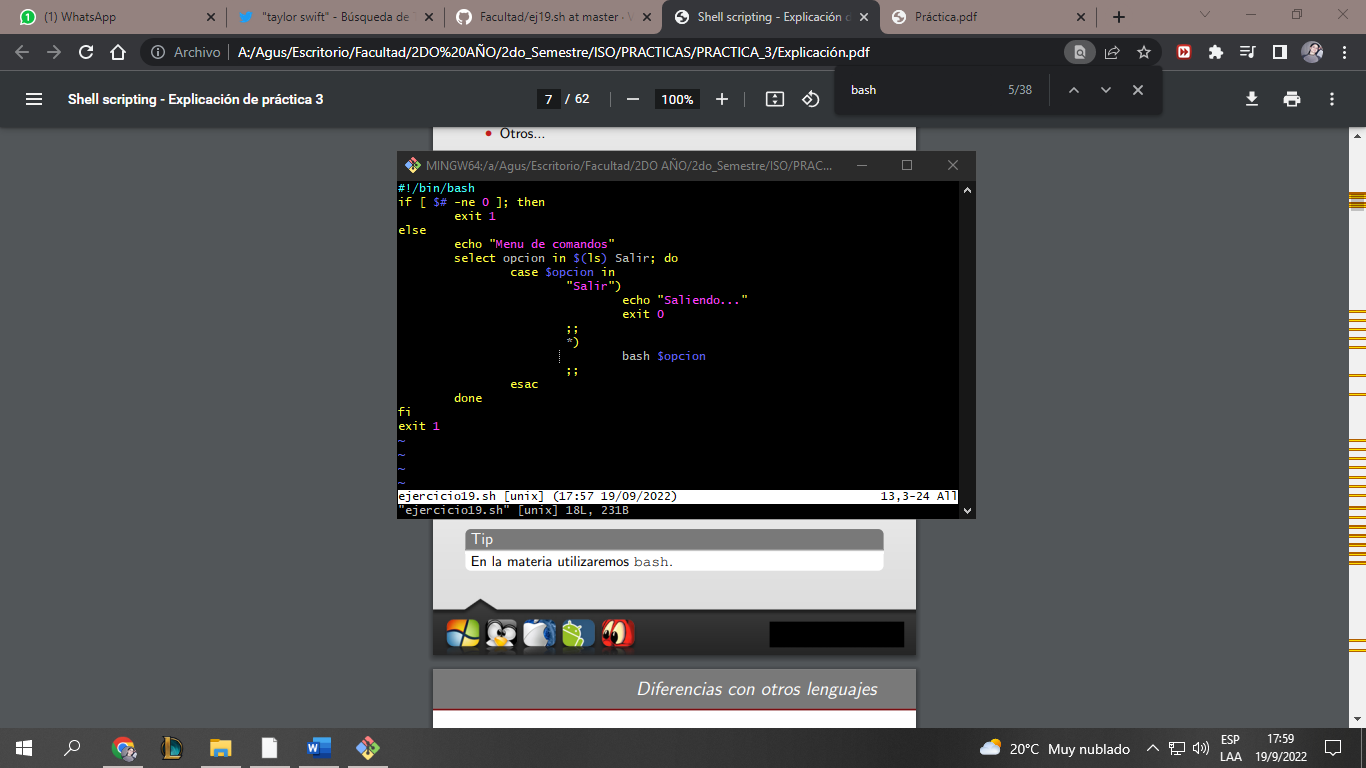
**No se como hacer que se intercambien.**

**18. Crear un script que verifique cada 10 segundos si un usuario se ha loqueado 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 menu, el cual, al ser invocado, mostrará un menú con la selección para cada uno de los scripts creados en esta práctica. Las instrucciones de como proceder deben mostrarse junto con el menú. El menú deberá iniciarse y permanecer activo hasta que se seleccione Salir.**



**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:**

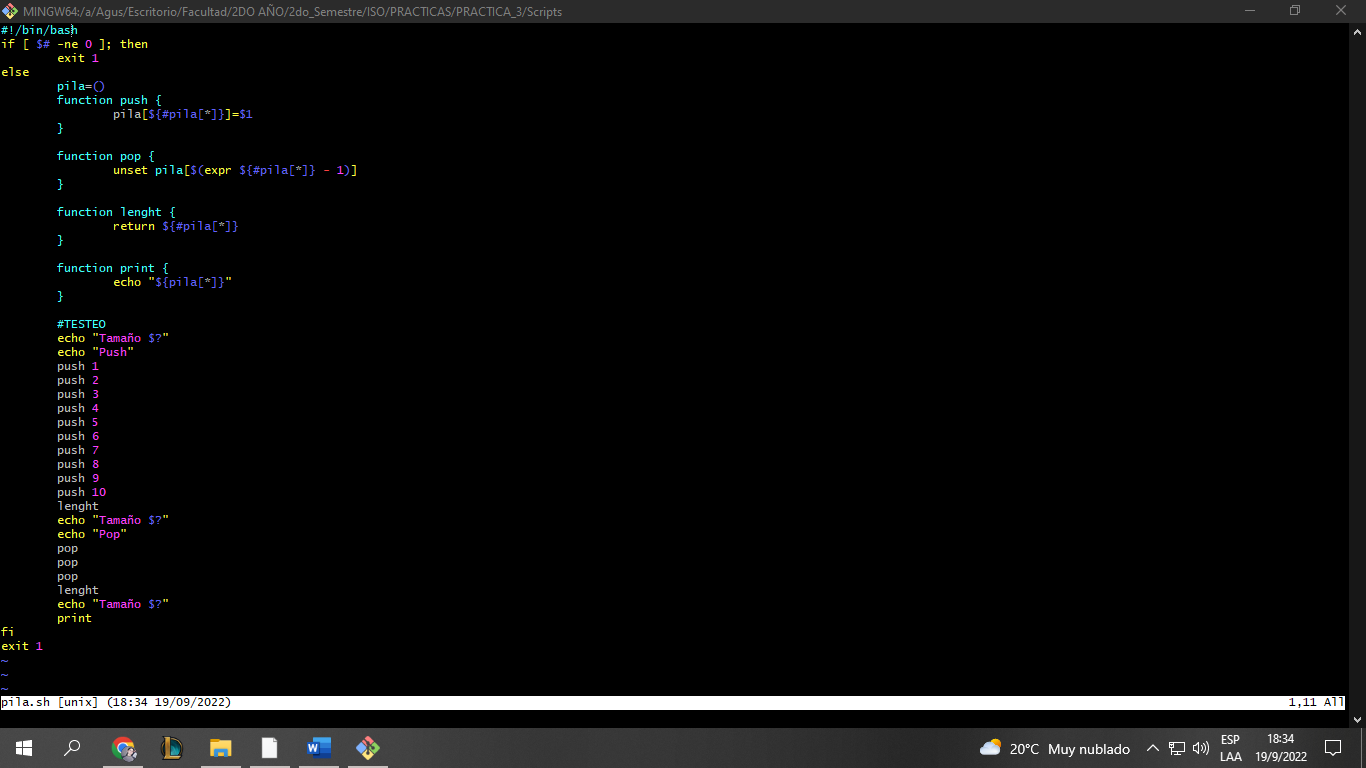
**push: Recibe un parámetro y lo agrega en la pila**

**op: Saca un elemento de la pila**

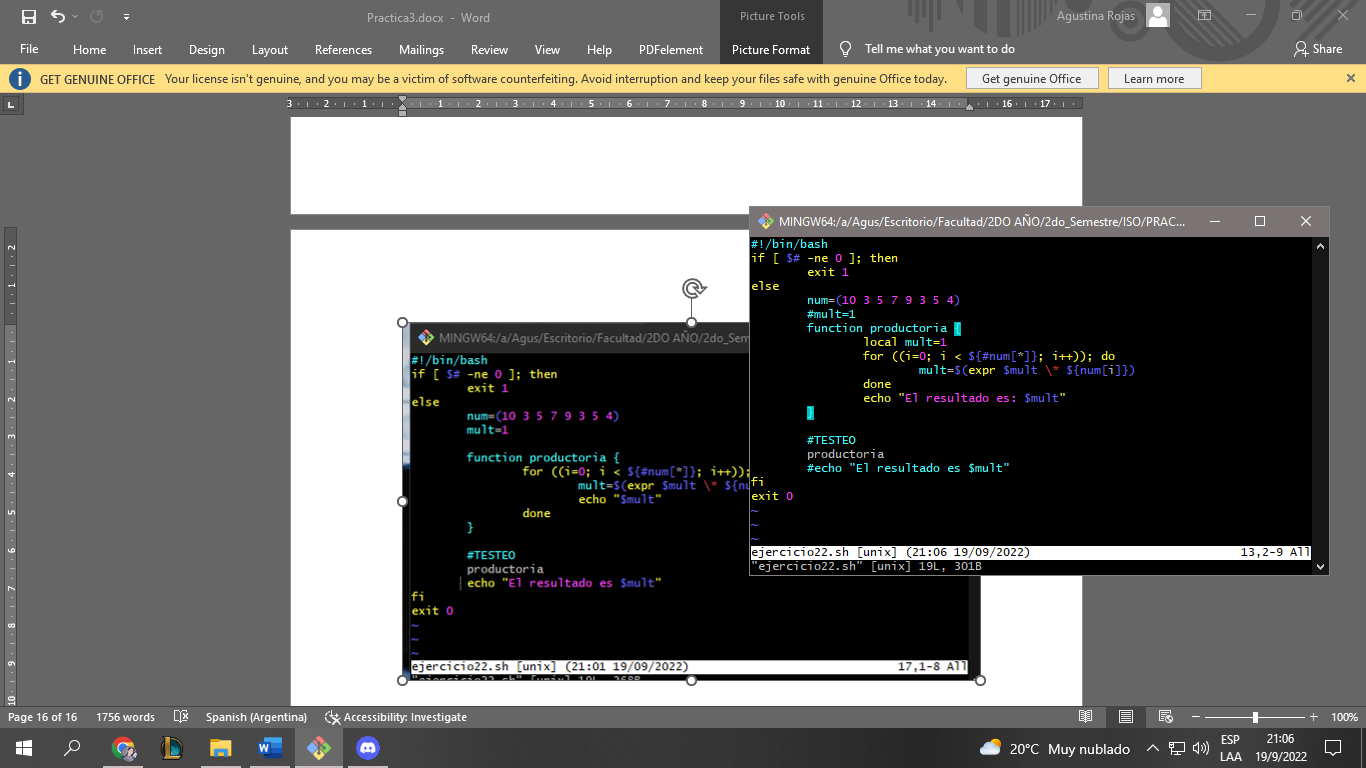
**length: Devuelve la longitud de la pila**

**print: Imprime todos elementos de la pila**

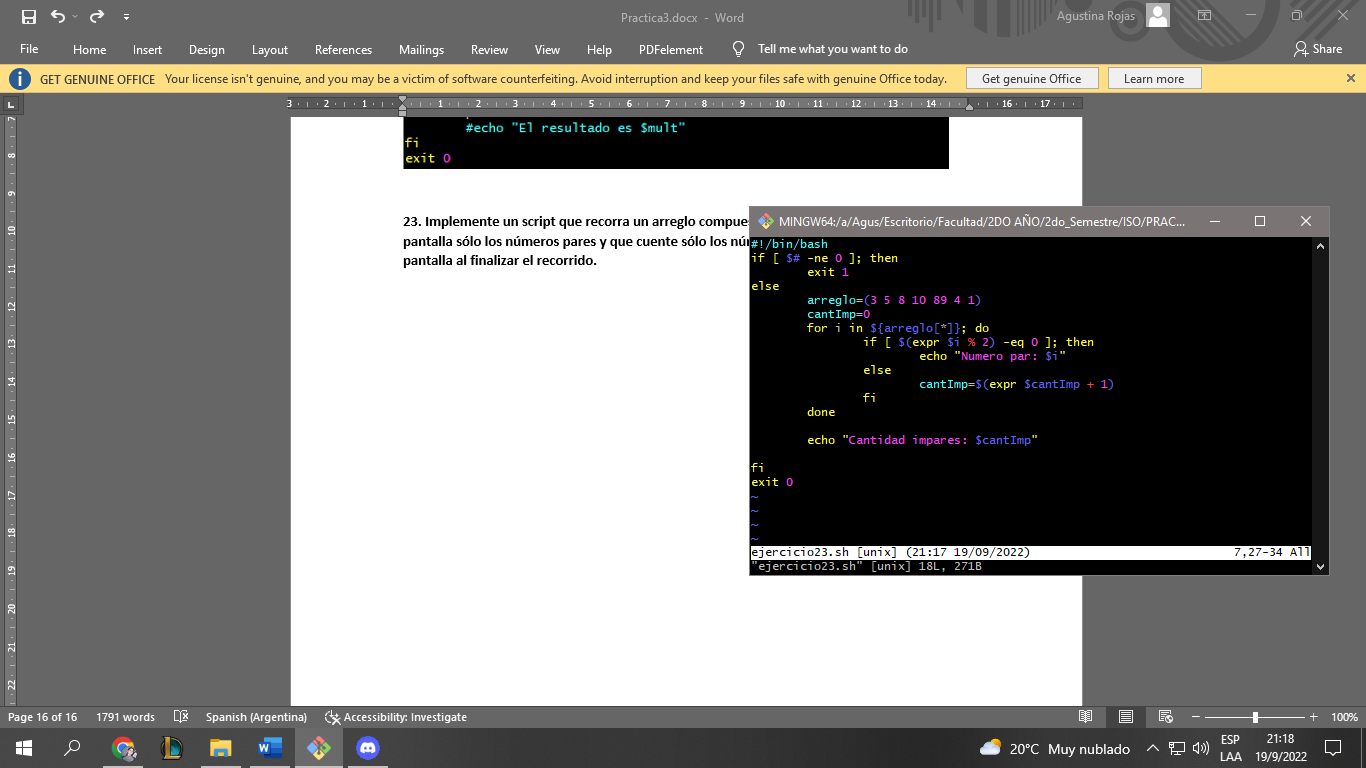
**21. Dentro del mismo script y utilizando las funciones implementadas: Agregue 10 elementos a la pila Saque 3 de ellos Imprima la longitud de la cola Luego imprima la totalidad de los elementos que en ella se encuentran**



**22. 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 del arreglo**



**23. 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.**



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

**vector1=( 1 .. N)**

**vector2=( 7 .. N)**

**Por ejemplo:**

**vector1=( 1 80 65 35 2 )**

**y**

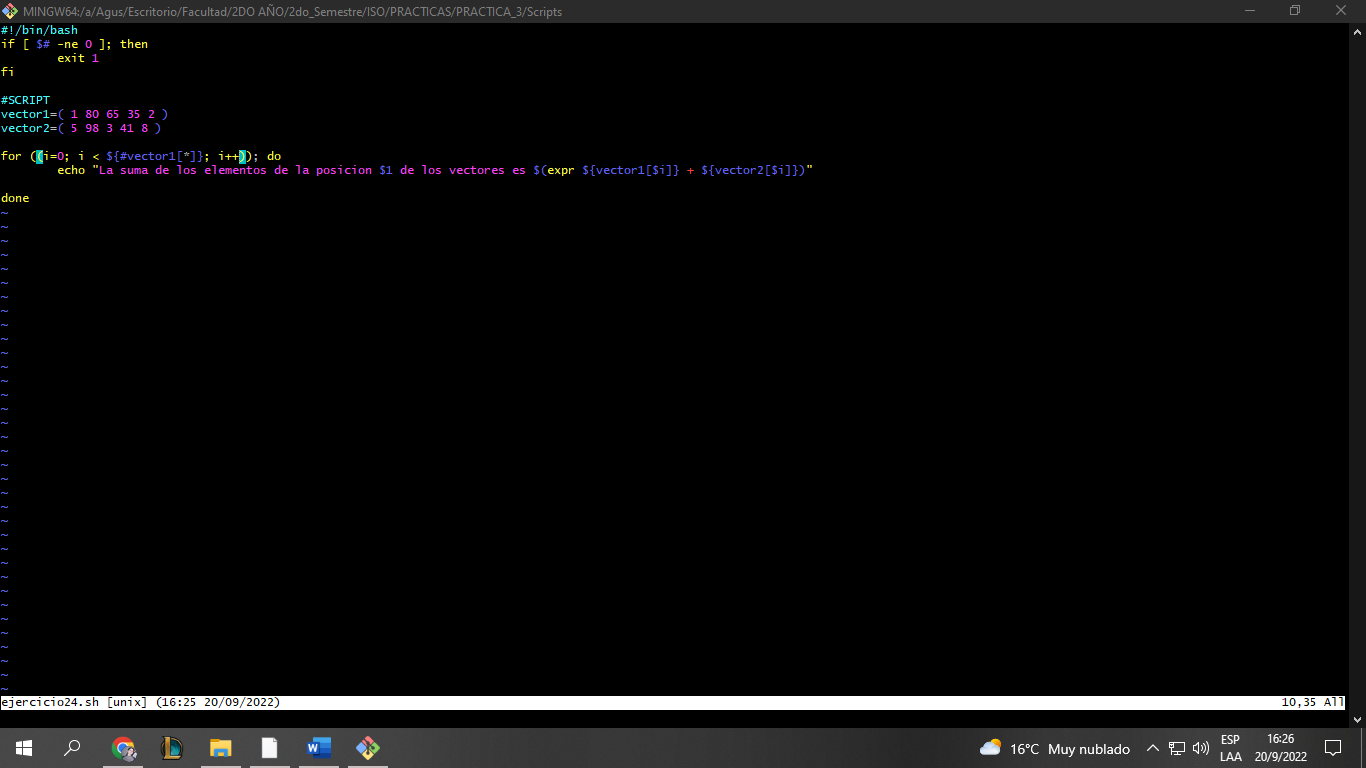
**vector2=( 5 98 3 41 8 ).**

**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**

**25. 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 parametro:**

**“-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**

En scripts

**26. 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, etc) 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.**

En scripts