TODO PAS

(PRACTICAS)

PRACTICA 1:

·**Shell** → **Programa que recoge comandos del ordenador y se los proporciona al SO** para que los ejecute. En GNU/Linux, usamos **bash** como shell.

·**Terminal** → **Programa que emula la terminal** de un computador, **iniciando una sesión de Shell**.

·**Scripting** → Creación de programas informáticos **para la automatización de tareas en un computador**.

·**Pasos para crear un programa:**

1. **Crear un archivo** con extensión **.sh**
2. **Añadir esta cabecera** → #!/ bin / bash
3. **Escribir el código** que necesite.
4. **Darle permisos de ejecución** o **ejecutar con** → ***bash*** *"nombre_programa.sh"*.

COMANDOS

·**Abrir editor de texto** → ***gedit &***
(& hace que la terminal se quede libre del proceso)

·**Escribir en la terminal** → ***echo** "texto de ejemplo"*

·**Dar permisos** → ***chmod** nivel_permisos nombre_archivo*
(Usaremos u+x para darle permisos de ejecución a nuestro user)

·**Ejecutar un programa:**

1. ***programa.sh*** (directorio del fichero debe de estar en $PATH)
2. ***./programa.sh*** o ***home/venom06/programa.sh*** (chmod u+x)
3. ***bash programa.sh***

·**Asignar variable** → ***NOMBRE_VAR=**"dato_var"***

·**Usar dato variable** → Ej: *echo **$NOMBRE_VAR***
(Muestra valor de la variable)

·**Mostrar caracteres reservados** → ***" $\var "***
(Muestra directamente la cadena "$var", no la variable de dentro)

·Variables de entorno → **Establecidas por el SO** para uso interno y del usuario. Para **listarlas**, usar comando **env**. Algunas son:

-*$HOME*: **Directorio personal** del usuario.

-*$PATH*: **Carpetas** donde están los **comandos**.

-*$HOSTNAME*: **Nombre máquina**.

-*$MACHTYPE*: **Arquitectura** del ordenador.

-*$PS1*: **Cadena antes del prompt en consola**. Sirve para modificarla a nuestro gusto.

- \t: hora.
- \d: fecha.
- \w: directorio actual.
- \h: nombre de la máquina.
- \W: última parte del directorio actual.
- \u: nombre de usuario.

-*$UID*: **ID** inmodificable **del usuario**.

-*$SHLVL*: **Nivel de anidamiento**. (Consola inicial es 0, de ahí para arriba).

-*$RANDOM*: **Devuelve num aleatorio**.

-*$SECONDS*: **Devuelve num segundos que bash lleva en marcha**.

·Crear variables de entorno → ***export*** *NOMBRE_VAR*
(Sirve para que sus procesos hijas puedan usarlas. Estos se crean con bash)

·Variables intrinsecas → Son **variables predefinidas por el intérprete de comandos que almacenan información** sobre el **entorno de ejecución**, el **estado de los comandos** y los **argumentos pasados al script**. Solo **lectura**. Listado:

-*$#*: **Num de argumentos** (argc)

-*$n*: **Argumento numero** *n* de los pasados. Si hay mas de 10, poner de la forma ***${n}.***

-*$**: **Todos los argumentos** como una sola **cadena**.

-*$@*: **Todos los argumentos** como una **array**.

-*$!*: **PID** del **último proceso** lanzado con **&**.

·Leer datos por teclado → ***read*** *var_guardar_leído*
Lista de opciones para este comando:

-*p "mensaje"*: **Muestra el mensaje** "**mensaje**" al pedir la información al usuario.

-*s*: **No muestra entrada del usuario**.

-*nN*: **Acepta** solo **N caracteres** en la escritura

-*tT*: **Deja un tiempo T para responder** la solicitud.

```
victor@victor-ayrna:~$ read -s -t5 -n1 -p "si (S) o no (N)?" respuesta
si (S) o no (N)?S
victor@victor-ayrna:~$ echo $respuesta
S
```

·Sustitución de comandos → *NOMBRE_VAR =`**ls**`* o *NOMBRE_VAR =**$(ls)***
Si hacemos echo de $*NOMBRE_VAR*, será como **hacer el comando ls** en la carpeta que lo ejecutemos.

·Operaciones aritméticas → ***let** X=5*3+1* o *x=**$[5*3+1]*** o *x=**$((5*3+1))***

·Condicional IF → ***if/elif** [expresión] **then** instrucción* / ***else** instrucción **fi***
IMPORTANTE: espacios antes y después [expresión].

·Comparación cadenas:

-*[s1 == s2]*: **true si s1 es igual a s2**, sino false.

-*[s1 != s2]*: **true si s1 es distintito a s2**, sino false.

-*[s1]*: **true si s1 no está vacía**, sino false.

-*[-n s1]*: **true si s1 tiene longitud > 0**, sino false.

-*[-z s2]*: **true si s2 tiene longitud = 0**, sino false.

-*[[s1 == s2*]]*: **true si s1 empieza por s2**, sino false.

·Comparación números:

-*[n1 -lt n2]*: **true** si **n1 es menor que n2**, sino false.

-*[n1 -gt n2]*: **true** si **n1 es mayor que n2**, sino false.

-*[n1 -le n2]*: **true** si **n1 es menor o igual que n2**, sino false.

-*[n1 -ge n2]*: **true** si **n1 es mayor o igual que n2**, sino false.

-*[n1 -eq n2]*: **true** si **n1 es igual a n2**, sino false.

-*[n1 -ne n2]*: **true** si **n1 es distinto de n2**, sino false.

·**Comparación ficheros:** (uso ***man [*** para saber más de ellos)

-*[-e f1]*: **true** si **existe el fichero f1**, sino false.

-*[-s f1]*: **true** si **f1 tiene un tamaño mayor que 0**, sino false.

-*[-f f1]*: **true** si **f1 es un fichero normal**, sino false.

-*[-d f1]*: **true** si **f1 es un directorio**, sino false.

-*[-l f1]*: **true** si **f1 es un enlace simbólico**, sino false.

-*[-r f1]*: **true** si **f1 tiene permisos de lectura**, sino false.

-*[-w f1]*: **true** si **f1 tiene permisos de escritura**, sino false.

-*[-x f1]*: **true** si **f1 tiene permisos de ejecución**, sino false.

·**Operadores lógicos:**

- *!* : **NO**. Si es true, devuelve false, y viceversa.

- *&&* o *-a* : **AND**.

- || o *-o* : **OR**.

·**Condicional CASE** → ***case** var **in** valor1) instrucciones;; valor2) instrucciones;; **esac***

·**Condicional FOR** → ***for** var **in** lista_valores **do** instrucciones; **done***

·**Condicional FOR C** → ***for ((**inicialización **;** comparación **;** norma_final**))***

```
#!/bin/bash

echo -n "Introduzca un número: "; read x;
let sum=0
for (( i=1; $i<$x; i=$i+1 ))
do
  let "sum=$sum + $i"
done
echo "La suma de los primeros $x números naturales es: $sum"
```

·**Comando seq** → ***(-seq** numero)*
Devuelve una secuencia del 1 al número.

·**Arrays:**

-Crear arrays: **nombreArray[pos]=Valor** o **nombreArray=(**Valor Valor1)

-Acceder a un valor: **${ nombreArray[pos] }**

-Acceder a todos los valores: **${ nombreArray[*] }**

-Longitud del array: **${nombreArray[@]}**

·Funciones → *function nombreFunc()*

```
#!/bin/bash
function chequea() {
  if [ -e "$1" ]
  then
    return 0
  else
    return 1
  fi
}

echo -n "Introduzca el nombre del archivo: "
read x
if chequea $x
then
  echo "El archivo $x existe !"
else
  echo "El archivo $x no existe !"
fi
```

Para pasarle argumentos, usarlas de esta forma.

·Redireccionamiento de salida:

-*comando > salida.txt*: Salida del comando **sobrescribe lo que había en el fichero con la salida**.

-*comando >> salida.txt*: Salida del comando **escribe lo que había en el fichero con la salida**, sin sobrescribir.

-*comando 2> error.txt*: Salida de **error** del comando **sobrescribe lo que había en el fichero con la salida**.

-*comando 2> error.txt*: Salida de **error** del comando **escribe lo que había en el fichero con la salida**, sin sobrescribir.

-*comando 2>&1*: redirecciona la salida de **error de comando** a la salida **estándar**.

-*comando &> todo.txt*: redirecciona **TODAS LAS SALIDAS** al fichero, sobrescribiendo.

-*comando < ficheroConDatos.txt*: redirecciona **entrada estándar** en un **fichero**.

·**Tuberías** → Sirve para **redireccionar entrada/salida comandos entre sí**, sin usar ficheros. **comando1 | comando2**. La **entrada de comando2 será tomada de la salida de comando1**.

·***Cat*** → Visualiza el contenido de uno o más ficheros. **cat** ejemplo.txt

·***Head*** → Sirve para **mostrar las n primeras líneas** de un fichero. **head** -1 ejemplo.txt

·***Tail*** → Sirve para **mostrar las n últimas líneas** de un fichero. **tail** -1 ejemplo.txt

·***Wc*** → Sirve para **mostrar el número de líneas, palabras o caracteres** de un fichero. **wc** -l ejemplo.txt

·***Cmp*** → Sirve **para comparar dos ficheros**, diciendo donde dejan de ser iguales. **cmp** ejemplo1.txt ejemplo2.txt

·***Sort*** → Sirve para **ordenar los datos** de un fichero. (Usar man sort para ver opciones). **sort** ejemplo.txt

·***Find*** → Sirve para **buscar ficheros en una carpeta en función de varios parámetros**. **find** -name "*.txt"

·***Basename*** → Sirve para **devolver el nombre de un archivo** con o sin su extensión.

·***Dirname*** → Sirve para **devolver la ruta de un archivo**.

·***Stat*** → **Muestra las propiedades** de un fichero.

·***Tr*** → Remplaza el primer carácter por el segundo en un texto. echo TIERRA | **tr** 'R' 'L'

· **Brace extension** → Es un operador conformado por llaves **{}** que nos permite **generar combinaciones de cadenas de texto**.
echo fichero.{pdf,jpg} → fichero.pdf fichero.jpg

- Solución: cambiar el IFS para que solo se utilice el \n:

```
1  victor@Laptop:~$ OLDIFS=$IFS
2  victor@Laptop:~$ IFS=$'\n'
3  victor@Laptop:~$ array=($(echo -e "El uno\nEl dos\nEl tres"))
4  victor@Laptop:~$ echo ${array[0]}
5  El uno
6  victor@Laptop:~$ echo ${array[1]}
7  El dos
8  victor@Laptop:~$ IFS=$OLDIFS
```

PRACTICA 2:

·**Expresión Regular (regex)** → Es una **secuencia de caracteres que forma un patrón de búsqueda**. Se utilizan para **encontrar, filtrar o manipular** cadenas de texto.

Carácter	BRE	ERE	Significado
\	✓	✓	Interpreta de forma literal el siguiente carácter
.	✓	✓	Selecciona un carácter cualquiera
*	✓	✓	Selecciona ninguna, una o varias veces lo anterior
^	✓	✓	Principio de línea
$	✓	✓	Final de línea
[...]	✓	✓	Cualquiera de los caracteres que hay entre corchetes
\n	✓	✓	Utilizar la *n*-ésima selección almacenada
{n,m}	X	✓	Selecciona lo anterior entre n y m veces
+	X	✓	Selecciona una o varias veces lo anterior
?	X	✓	Selecciona una o ninguna vez lo anterior
|	X	✓	Selecciona lo anterior o lo posterior
(...)	X	✓	Selecciona la secuencia que hay entre paréntesis[2]
\{n,m\}	✓	X	Selecciona lo anterior entre n y m veces
\(...\)	✓	X	Selecciona la secuencia que hay entre paréntesis[2]
|	✓	X	Selecciona lo anterior o lo posterior

·**Grep** → Es un **comando para la búsqueda de expresiones regulares** en un fichero de texto determinado. Utiliza la BRE, pero puedes utilizar **ERE** añadiendo el **flag -E** o usando **egrep**. Debemos poner la **expresión regular siempre entre comillas simples**. Algunos flags útiles son:

-*grep -i*: **Ignora mayúsculas y minúsculas**.

-*grep -o*: **Muestra solo el patrón**, no la línea completa.

-*grep -v*: Lo contrario al anterior.

grep -e: Permite **especificar varios patrones**. (Ej: **grep -e** 'perro' **-e** 'gato' fichero.txt)

-*grep -w*: Busca la **palabra completa exacta**. Por ejemplo, evita que al buscar "pie" te devuelva "piedad" o "ciempiés".

-*grep -A n,-B n,-C n*: Muestra las **n líneas antes del emparejamiento** (B, before), las **n líneas después del emparejamiento** (A, after) o **las n líneas antes y después del emparejamiento** (C, context)

·**Sed** → Es un **comando para editar el texto, cambiando un patrón por otro especificado**. Para usar **ERE**, debemos poner el **flag -r**. Es necesario poner -e

para que ejecute el comando a continuación. La estructura del comando es esta:

sed [opciones] '[dirección][instrucción][argumentos]' archivo

En opciones, añadiremos los flags comentados antes y algunos que podrán ser necesarios, como estos:

-*sed -n*: Activa el **modo silencioso**. **No muestran nada** a no ser que se **active el argumento p** (ver después).

-*sed -i*: **Edita el archivo** y cambia la salida.

En la parte de **dirección**, debemos poder sobre **que líneas del archivo se fijara el comando para actuar**. Si **queremos una sola línea**, ponemos el numero de esta, si **queremos un rango**, lo pones de la forma ***n1,n2*** , y si **queremos todas menos algunas**, añadimos **!** más marcar cuales no queremos. También **pueden ser regex**, no números de líneas.

En la parte de **argumentos**, podemos añadir varias configuraciones, como son:

-*d*: **Borra las líneas seleccionadas**.

-*p*: **Imprime las líneas seleccionadas**. Este es **útil para no repetir las líneas** dos veces cuando las encuentres, así que usamos **-n y p**.

-*s*: **Sustituir un patron por otro**. Esta es su sintaxis:
s/patron/reemplazo/[banderas]

Tiene banderas internas como **n**, para que solo **cambie la n-ésima apariencia**, **g**, para que **reemplace todas**, o **p**, que **muestra la línea** (útil con -n).

Ejemplo:

```
i72jivem@VTS3:~/PAS/p2$ cat ejemplo.txt
Este es otro ejemplo de expresiones regulares
La segunda parte ya la veremos
,,,,adios,hola
```

-n y p en acción, mostrando todo meno las 2 primeras líneas

```
i72jivem@VTS3:~/PAS/p2$ cat ejemplo.txt | sed -n -e '1,2!p'
....adios.hola
```

Borra las líneas que empiezen con L, gracias a la configuración **d**.

```
i72jivem@VTS3:~/PAS/p2$ cat ejemplo.txt | sed -e '/^L/d'
Este es otro ejemplo de expresiones regulares
....adios.hola
```

Intercambia los “La” o “la” del texto por “El”, gracias a **s**. Usamos -r para los [].

```
i72jivem@VTS3:~/PAS/p2$ cat ejemplo.txt | sed  -r -e 's/[Ll]a/El/'
Este es otro ejemplo de expresiones reguElres
El segunda parte ya la veremos
,,,,adios,hoEl
```