# Práctica 3, explicación

## Shell

Una shell es un intérprete de comandos que proveen estructuras de control que permiten programar shell scripts los cuales permiten automatizar tareas, hacer aplicaciones interactivas y con interfaces gráficas.

### Tipos de shell

Hay varias shell las cuales sus diferencias están en la sintaxis.

- sh: shell por defecto de Unix.
- bash: cómoda, instalada por defecto en la mayoría de las distribuciones.
- dash: eficiente y parcialmente compatible con bash.
- csh: sintaxis incompatible con bash o dash.

Shell se diferencia con otros lenguajes porque es práctico para manejar archivos, es simple para crear procesos y manipular salidas, es independiente de la plataforma y funciona bien en cualquier SO de tipo *nix.

### Elementos del lenguaje

- Intrucciones

  **REEMPLAZO DE COMANDOS**

  Permite usar la salida de un comando como si fuese una cadena de texto normal, permite guardarlo en variables o usarlos directamente y se la puede usar de dos formas

  ```
  ${comando_valido}
  `comando_valido`
  ```

- Redirecciones y pipes: los procesos normalmente cuentan con 3 archivos abiertos (stdin, entrada estándar que normalmente es el teclado, stdout que es una salida estándar, generalmente el monitor y stderr que es un error estándar). Se identifican en el S.O con un número, el file descriptor (0, 1 y 2).

  ```
  > #redirección destructiva, en caso de que el archivo no exista es creado y si existe lo sobrescribe
  >> #redirección no destructiva, en caso que el archivo no exista es creado y si existe lo agrega al final
  2>
  2>>
  # redirigen el error estándar


  comando < archivo #archivo es entrada de comando
  < # indica la entrada de un comando
  ```

  **PIPES**

```
comando | comando2 | comando3
# conectan la salida de un comando con la entrada de otro
```

- Comentarios, empiezan con #
- Estructuras de control
  - if

```
if [ condition ]
then
  block
fi
```

  - while

```
while [condition]
do
  block
done

## do..until
until [condition] # mientras no se cumpla la condición
do
  block
done
```

  - for (2 tipos)

```
# C-style
for ((i=0;i<10; i++))
do
  block
done

# Con lista de valores (foreach)
for i in value1 value2 valueN;
do
  block
done
```

  - case

```
case $variable in
  "valor 1")
    block
  ;;
  "valor 2")
    block
    ;;
*) #para cualquier otro caso
  block
;;
esac

## menú de opciones
select variable in opcion1 opcion2 opcion3
do
```

```
  # en $variable está el valor elegido
  block
done
```

- Variables: bash soporta string y arrays, los nombres son case sensitive

```
# Creando una variable
NOMBRE="pepe" #No debe haber espacios entre =

# Para accederlas uso $
echo $NOMBRE
```

  - Strings, los nombres de las variables pueden contener mayúsculas, minúsculas, números y _ pero no pueden empezar con un número
  - Arreglos()

```
# Creación
arreglo_a=() # es creado vacío
arreglo_b=(1 2 3 4 5) # inicializado

# Asignación de un valor en una posición concreta
arreglo_b[2]=spam

# Acceso a un valor del arreglo
echo ${arreglo_b[2]}
copia= $(arreglo_b[2]}

# Acceso a todos los valores del arreglo
echo ${arreglo[@]}

# Tamaño del arreglo
${#arreglo[@]}

# Borrado de un elemento
unset arreglo[2]
```

Las comillas no son necesarias a menos que

- El string tenga espacios
- La variable puede tener un contenido con espacios
- Importantes para las condiciones de if, while, etc.

Pueden ser simples o dobles

- Funciones: permiten modularizar el comportamiento de los scripts

```
# Hay dos formas de declararlos

function nombre{block}
###
nombre() {block}

return #retorna un valor entre 0 y 255
$? #permite evaluar el valor de retorno
```

Las variables no inicializadas son reemplazadas por un valor nulo o 0, según el contexto de evaluación y por defecto las variables son globales.

- Evaluación de condiciones lógicas

| Operador | Con strings | Con números |
|---|---|---|
| Igualdad | "$nombre" = "Maria” | $edad -eq 20 |
| Desigualdad | "$nombre" != "Maria” | $edad -ne 20 |
| Mayor | A > Z | 5 -gt 20 |
| Mayor o igual | A ≥ Z | 5 -ge 20 |
| Menor | A < Z | 5 -lt 20 |
| Menor o igual | A ≤ Z | 5 -le 20 |

- Argumentos y valor de retorno: los scripts pueden recibir argumentos en su invocación, para accederlos se usan variables especiales

```
$0 #contiene la invocación al script
$1 $2 $3 #contiene cada uno de los argumentos
$# #contiene la cantidad de argumentos recibidos
$* #contiene la lista de todos los argumentos
$? #contiene en todo momento el valor de retorno del último comando ejecutado


if [ $# -ne 2 ]; then # siempre chequear que la cantidad de argumentos es el esperado
  exit 1 # Error
else
  echo "Nombre: $1, Apellido: $2"
fi
exit 0 # Funcion´o correctamente
```

Para terminar un script se usa la función exit, que causa la terminación de un script y puede devolver un valor entre 0 y 255

## Algunos comandos útiles

```
## Imprimir el contenido de un archivo
cat archivo

## Imprimir texto
echo "Hola mundo"

## Leer una línea desde entrada estándar en la variable var
read var

## Quedarme con la primer columna de un texto separado por : desde la entrada estándar
cut -d: -f1

## Contar la cantidad de líneas que se leen desde entrada estándar
wc -l

## Buscar todos los archivos que contengan la cadena pepe en el directorio /tmp
grep pepe /tmp/*

## Buscar todos los archivos dentro del home del usuario cuyo nombre termine en .doc
find $HOME -name "*.doc"

## uscar todos los archivos dentro del directorio actual que sean enlaces simbólicos
find -type l
```

```
## Empaquetado, une varios archivos en uno solo
tar -cvf archivo.tar archivo1 archivo2 archivo3
tar -xvf archivo.tar

##Compresión, redue el tamaño del archivo
gzip archivo.tar ##genera archivo.tar.gz comprimido
gzip -d archivo.tar.dz # descomprime archivo.tar

## el comando tar puede invocar a gzip con el argumento -z
tar -cvzf archivo.tar.gz archivo1 archivo2
tar -xvzf archivo.tar.gz


# Indicar intérprete en la primer línea
#!/bin/bash
# Esta línea especifica que se ejecutará un script
```