Resumen Práctica 3

Shell

Programa que interpreta y ejecuta comandos en SO’s (*nix). Sirve como interfaz entre usuario y SO.

Características

Es un interprete de comandos, responde a comandos en tiempo real, es configurable en sistemas *nix y permite programar shell scripts mediante estructuras de control.

Función Shell Scripts

Permite automatizar tareas que se repiten mucho, también crear aplicaciones interactivas, y desarrollar interfaces gráficas usando comandos (ejemplo zenity).

Tipos de Shell

- SH: Shell predeterminada de Unix

- bash: Instalada por defecto en casi todas las distros.
- dash: Eficiente pero parcialmente compatible con bash.
- csh: Sintaxis distinta e incompatible con bash/dash
- Hay mas variantes pero son menos comunes pero todas permiten la ejecución de comandos y scripts

Elementos del Lenguaje en Shell

1. Instrucciones (Comandos): Existen comandos internos (built-in) incluidos en la shell (Para consultar utilizar “help”) y tambien externos, que son programas independientes (Para consultar utilizar “man comando”).
2. Redirecciones y pipes: Estos permiten redirigir la salida de comandos (<,>, |)
3. Comentarios: Comienzan con # y sirven para añadir notas sin afectar el funcionamiento del código.
4. **Estructuras de control:**
    a. if
    b. while
    c. for
    d. case
5. Variables: Almacenan valores como strings o arrelos (“()”)
6. Funciones: Conjuntos de comandos que están agrupados que se pueden reutilizar.

Comandos Útiles

Manejo de archivos y texto:

cat archivo: Mostrar contenido archivo

echo “Hola mundo”: Imprimir texto en pantalla

read var: Leer una linea desde la entrada estándar y la guarda en la variable var

wc -l: Contar numero de lineas de la entrada estandar

cut -d: -f1: Extrae primera columna de texto separado por : desde la entrada estándar.

Búsqueda de archivos:

- **grep pepe /tmp/*:** Busca archivos en /tmp que contengan “pepe”.
- **find $HOME -name “*.doc”:** Busca archivos con extensión .doc en el directorio del usuario.
- **find -type l:** Buscar enlaces simbólicos en el directorio actual

Empaquetado y Compresión:

- **tar -cvf archivo.tar archivo1 archivo2:** Crea un archivo empaquetado (**archivo.tar**) de varios usuarios.
- **tar -xvf archivo.tar:** Extrae los archivos de **archivo.tar**
- **gzip archivo.tar:** Comprime archivo.tar a **archivo.tar.gz**
- **gzip -d archivo.tar.gz:** Descomprime **archivo.tar.gz**
- **tar -cvzf archivo.tar.gz arch1 arch2 arch3:** Crea y comprime un paquete
- **tar -xvzf archivo.tar.gz:** Extrae y descomprime el paquete

Redirecciones y Pipes: stdin, stdout, stderr

- **stdin (Entrada estándar):** Teclado, representado por el descriptor 0.
- **stdout (Salida estándar):** Monitor, representado por el descriptor 1.

- **stderr (Error estándar):** Monitor también, pero específicamente para errores, representado por el descriptor 2.

Modos de redireccionamiento

- `comando > archivo`: Redirección destructiva, esto significa que si el archivo no existe, se crea, pero si existe, se sobrescribe.
- `comando >> archivo`: Redirección no destructiva, es lo mismo que la redirección destructiva, solo que si el archivo existe, el contenido se agrega al final.
- `2> y 2>>`: Redirigen salida de errores (stderr) de forma destructiva (`2>`) o no destructiva si es con `2>>`
- `< archivo`: Usa el contenido de `archivo` como entrada de comando en lugar de entrada estándar (teclado).
    - Ejemplos de esto seria `grep “Error” < log.txt`

Pipes (tuberías)

Los pipes (|) permiten conectar la salida de un comando con la entrada de otro, se utiliza para bash script.

Sintaxis

- `comando1 | comando2 | comando3`

Ejemplos

- `cat archivo | tr a-z A-Z` : Convierte contenido del archivo a mayúsculas
- `cat archivo | grep hola | cut -d, -f1` : Busca “hola” y extrae la primera columna separada por comas.
- `cat /etc/passwd | cut -d: -f1 | grep a | wc -l` : Cuenta las líneas con la letra “a” en el primer campo

Variables en Bash

- **Strings y Arreglos:** Bash distingue entre letras mayúsculas y minúsculas en los nombres de variables.

Creación y acceso:

- Definir **NOMBRE=”pepe” (sin espacios alrededor del `=`)**
- Acceder: Usar $NOMBRE, o ${NOMBRE} para evitar ambigüedades.

```
echo ${NOMBRE}texto_extra # Accede a $NOMBRE sin confusión
```

Variables y Arreglos

Para crear un arreglo se utiliza:

- `arreglo_a=()` para un arreglo vacío
- `arreglo_b( 2 3 45 6 7 )` para un arreglo inicizalizado

Asignar valores:

- `arreglo_b[2]=spam` (asigna “spam” en la pos 2)

Acceder a elementos:

- `echo ${arreglo_b[2]}` de manera Individual
- `echo ${arreglo_v[@]}` o `echo ${arreglo_b[*]}` para todos

Otros comandos:

- Tamaño del arreglo: `${#arreglo[@]}` o `${#arreglo[*]}`
- Eliminar un elemento: `unset arreglo[2]` deja posición vacía

> `Los índices de los arreglos comienzan en 0`

Tipos de comillas

- Comillas dobles (`“”`)
  - Estas permiten usar variables y resultados de comandos dentro del texto:

```
var='variables'
echo "Permiten usar $var"          # Muestra el valor de $var
echo "Y resultados de comandos $(ls)"  # Ejecuta y muestra el
resultado de ls
```

- Comillas simples (`’’`)
  - Estas no permiten el uso de variables o comandos dentro del texto; el contenido se muestra de manera literal

```
echo 'No permiten usar $var'       # Muestra "$var" literalme
nte
echo 'Tampoco resultados de comandos $(ls)'  # Muestra "$(l
s)" sin ejecutar
```

Las comillas ayudan a controlar la interpretación de variables y comandos dentro de los textos.

Reemplazo de Comandos

El **reemplazo de comandos** permite usar la salida de un comando como texto, guardándola en variables o usándola directamente en scripts.

Sintaxis

1. **Forma recomendada:** `$(comando_valido)`

2. **Alternativa**: `` `comando_valido` `` (menos clara y con reglas de anidamiento más complejas).

Ejemplos

- Guardar la salida de `ls` en una variable:

```
arch="$(ls)"        # También: arch=`ls`
```

Variables especiales para argumentos:

- `$0`: Nombre del script (la forma en que se invocó).
- `$1`, `$2`, `$3`, ...: Cada uno de los argumentos pasados.
- `$#`: Número total de argumentos recibidos.
- `$*`: Lista de todos los argumentos como una cadena.

Funciones en Bash

Las funciones ayudan a modularizar scripts en Bash

Declaración

1. **Con** `function`:

```
function nombre {
     # código
 }
```

2. **Sin** `function`:

```
nombre() {
     # código
 }
```

Características:

se usa `return` para devolver un valor entre 0 y 255, dicho valor de retorno se evalúa con `$?`, y los argumentos se reciben como `$1`, `$2`, etc