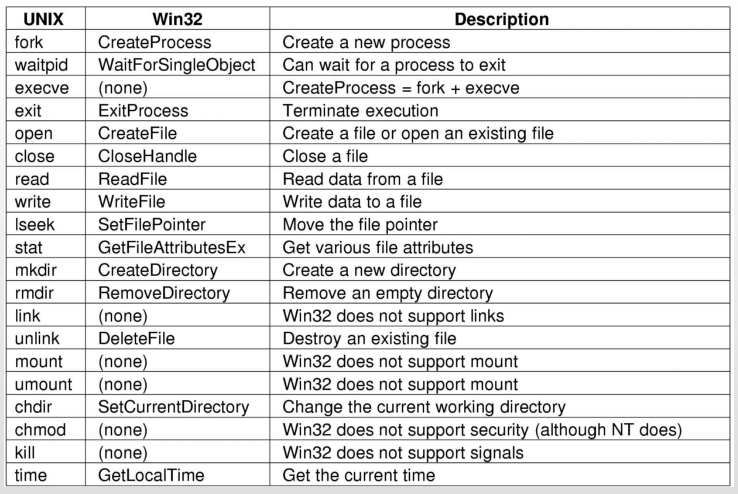
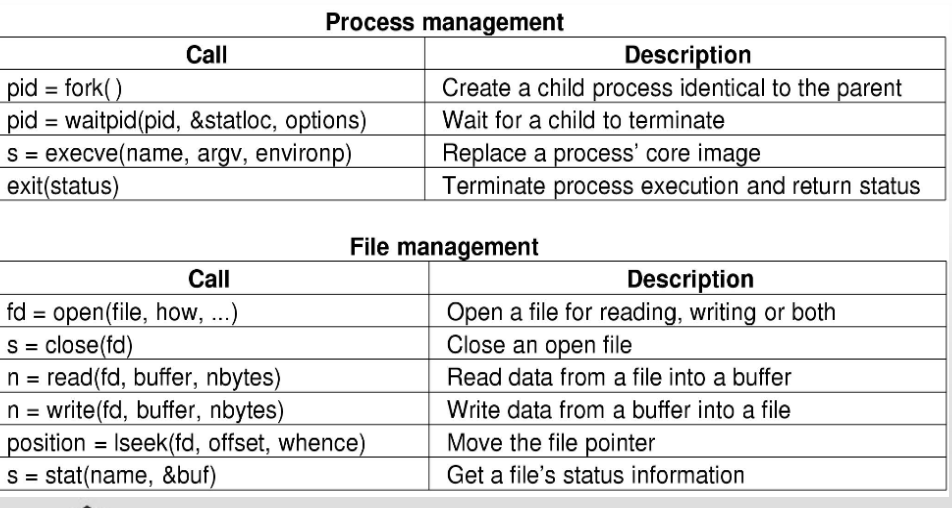
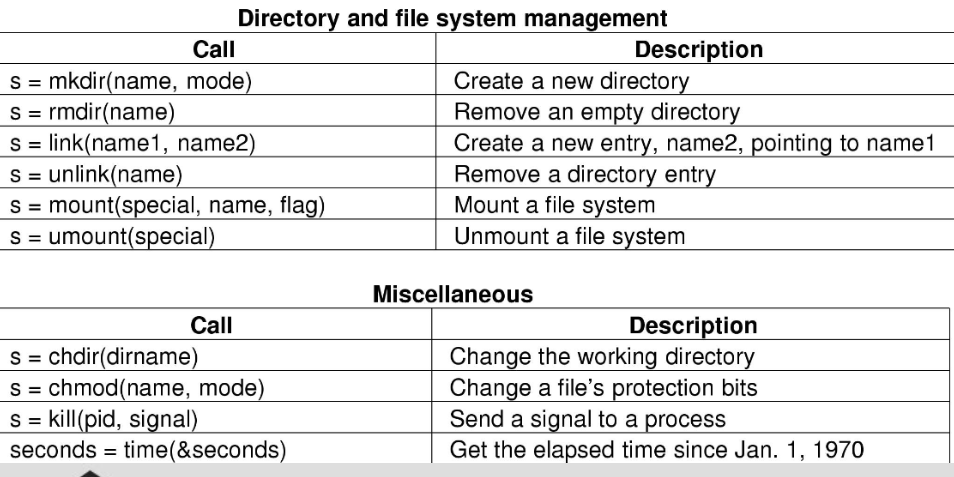
**Codigos de teoria**





**EXPR**

El comando `expr` en Unix/Linux permite la evaluación de expresionesaritméticas y de cadenas de texto. A continuación, se presentan los tipos de operaciones que se puedenutilizar con `expr`:

1. \*\*OperacionesAritméticas:\*\*

- `+`: Suma.

- `-`: Resta.

- `\*`: Multiplicación.

- `/`: División entera.

- `%`: Módulo (resto de la división entera).

- `:`: División real (puedeproducirnúmerosdecimales).

- `=`: Comparasi dos números son iguales (retorna 1 si son iguales, 0 si no lo son).

2. \*\*Operaciones de Cadenas de Texto:\*\*

- `:`: Busca un patrónen una cadena y devuelve la longitud de la coincidencia.

- `index`: Busca la posición de una subcadenaen una cadena.

- `length`: Devuelve la longitud de una cadena.

- `match`: Compara una cadena con una expresión regular.

3. \*\*OperadoresLógicos:\*\*

- `<`: Compruebasi arg1 es menor que arg2.

- `<=`: Compruebasi arg1 es menor o igual que arg2.

- `>`: Compruebasi arg1 es mayor que arg2.

- `>=`: Compruebasi arg1 es mayor o igual que arg2.

- `!=` o `<>`: Compruebasi arg1 no es igual a arg2.

Es importantedestacar que las expresiones dentro de `expr` debenestarseparadas por espacios y, enalgunoscasos, los operadorespuedennecesitar ser escapados con un backslash (\) para evitarproblemas de interpretación por parte del shell.

**TEST**

El comando `test` en Unix/Linux permiteevaluarexpresiones y generar un valor de retorno, que suele ser 0 (true) o 1 (false). Además, comomencionaste, es comúnmentereemplazado por eluso de corchetes `[ ]`, lo que hace que la sintaxis sea más legible. Las expresiones que se pueden usar con `test` o `[ ]` se puedendividirentrescategoríasprincipales:

1. \*\*Evaluación de archivos:\*\*

- `-e archivo`: Compruebasielarchivoexiste.

- `-f archivo`: Compruebasielarchivo es un archivo regular (no undirectorioni un dispositivo).

- `-d directorio`: Compruebasieldirectorioexiste.

- `-s archivo`: Compruebasielarchivotiene un tamaño mayor que cero.

- `-r archivo`: Compruebasielarchivo es legible.

- `-w archivo`: Compruebasielarchivo es escribible.

- `-x archivo`: Compruebasielarchivo es ejecutable.

- `-ntarchivo`: Compruebasielarchivo es másreciente que otroarchivo.

- `-otarchivo`: Compruebasielarchivo es másantiguo que otroarchivo.

2. \*\*Evaluación de cadenas de caracteres:\*\*

- `cadena1 = cadena2`: Compruebasi las dos cadenas son iguales.

- `cadena1 != cadena2`: Compruebasi las dos cadenas son diferentes.

- `-z cadena`: Compruebasi la cadenaestávacía (longitudigual a cero).

- `-n cadena`: Compruebasi la cadena no estávacía (longitud mayor que cero).

3. \*\*Evaluacionesnuméricas:\*\*

- `n1 -eq n2`: Comprueba si los números son iguales.

- `n1 -ne n2`: Comprueba si los números son diferentes.

- `n1 -lt n2`: Compruebasi n1 es menor que n2.

- `n1 -le n2`: Compruebasi n1 es menor o igual que n2.

- `n1 -gt n2`: Compruebasi n1 es mayor que n2.

- `n1 -ge n2`: Compruebasi n1 es mayor o igual que n2.

Ten encuenta que en las expresionesnuméricas, se utilizaeloperador `-` antes de los operadores de comparación para indicar que se trata de operacionesnuméricas. Además, las cadenas de caracteres y las variables debenestarencerradasencomillasdobles para manejarcorrectamente los casosen los que contienenespacios o caracteresespeciales.

**Estrucuras de control**

Las estructuras de control en shell scripting permitencontrolarelflujo de ejecución de un script, y aquíestá la sintaxis de las estructuras de control máscomunesen Bash:

1. \*\*Estructura `if`\*\*

```bash

if [ condición ]; then

# Código a ejecutarsi la condición es verdadera

elif [ otra\_condición ]; then

# Código a ejecutarsi la otracondición es verdadera (opcional)

else

# Código a ejecutarsininguna de las condiciones es verdadera (opcional)

fi

```

2. \*\*Estructura `case`\*\*

```bash

case valor in

patrón1)

# Código a ejecutarsi valor coincide con patrón1

;;

patrón2)

# Código a ejecutarsi valor coincide con patrón2

;;

\*)

# Código a ejecutarsininguno de los patrones coincide (opcional)

;;

esac

```

3. \*\*Estructura `while`\*\*

```bash

while [ condición ]; do

# Código a ejecutarmientras la condición sea verdadera

done

```

4. \*\*Estructura `for`\*\*

```bash

for variable in lista\_de\_elementos; do

# Código a ejecutar para cadaelementoen la lista

done

```

5. \*\*Estructura `select`\*\*

```bash

select variable in lista\_de\_elementos; do

# Código a ejecutar para la opciónseleccionada

done

```

**BREAK**

En Bash y otroslenguajes de programación, las sentencias `break` y `continue` se utilizan dentro de bucles para controlarelflujo de ejecución.

La sentencia `break` se utiliza para salirinmediatamente de un bucle y continuar con la ejecución del códigodespués del bucle. Cuando se ejecuta `break`, elbucle se detiene por completo, y el control pasa al siguientecomandodespués del bucle.

for i in 1 2 3 4 5; do

if [ "$i" -eq 3 ]; then

break # Sale del buclecuandoi es igual a 3

fi

echo "Iteración $i"

done

```

Enelejemplo anterior, elbucle `for` se detendrácuando `i` sea igual a 3, y no se imprimirán las iteracionesposteriores.

**CONTINUE**

La sentencia `continue` se utiliza para saltarel resto del ciclo actual y pasar a la siguienteiteración del bucle. Enotras palabras, cuando se ejecuta `continue`, se omiten las instruccionesrestantesenesaiteración y se pasa a la siguienteiteración del bucle.

for i in 1 2 3 4 5; do

if [ "$i" -eq 3 ]; then

continue # Salta la iteracióncuandoi es igual a 3

fi

echo "Iteración $i"

done

Enelejemplo anterior, cuando `i` es igual a 3, se ejecuta `continue`, lo que hace que la iteración actual se salte y se continúe con la siguienteiteración.

***VARIABLES***

Enelcontexto de shell scripting, las variables puedenclasificarseen los siguientestipos:

1. Variables de cadena/ Variables numéricas (numeric variables):Aunque las variables en shell scripting son generalmentetratadascomocadenas, puedesrealizaroperacionesnuméricassi es necesarioutilizandocomandoscomo `expr`.

numero=42

3.Variables de arreglo (array variables): Permitenalmacenarmúltiplesvaloresen una sola variable. Los shells como Bash admitenarreglosunidimensionales y pueden ser declarados y utilizados de la siguientemanera:

# Declarar un arreglo

mi\_arreglo=("valor1" "valor2" "valor3")

# Acceder a elementos del arreglo

primer\_elemento="${mi\_arreglo[0]}"

#array length  
array\_length= $(#array[@])

4.Variables de entorno (environment variables): Son variables especiales que almacenaninformaciónsobreelentorno del sistema, elusuario actual, y otrasconfiguraciones. Algunosejemploscomunesincluyen `HOME`, `PATH`, `USER`, entre otros.

Ej:

echo "Mi directorio de inicio es $HOME"

Enresumen, en shell scripting, puedestrabajar con variables de cadena, numéricas y de arreglo. Además, el shell es débilmentetipado, lo que significa que no se requiere una declaraciónexplícita de tipos y las conversiones se realizan de maneraautomática.

**FUNCIONES**

Para definir una función, utiliza la siguientesintaxis:

sumar() {

resultado=$(($1 + $2))

echo "La suma de $1 y $2 es igual a $resultado"

}

sumar 5 3

Encuanto al paso de parámetros de una función a otra, puedeshacerlo de la mismamanera que se pasanparámetros a una funciónen la línea de comandos. Dentro de una función, puedes acceder a los parámetrosutilizando las variables especiales `$1`, `$2`, `$3`, etc., que representanel primer, segundo, tercerparámetro, y asísucesivamente.

|  |  |
| --- | --- |
| USERS | |
| who | Muestra usuarios del Sistema |
| w | Who pero con mas detalles |
| whoami | El nombre de tu usuario actual |
| finger user | Muestra lso datos de “user” |

|  |  |
| --- | --- |
| PARAMETROS DEL SISTEMA | |
| $# | Cantidad de parametrosrecibidos |
| $1 .. $X | Parametros recibidos de 0 hasta X |
|  |  |

|  |  |
| --- | --- |
| TIME | |
| Tiempo=$(date) | Decuelve hora actual (date +%s hora ensegundos) |
|  |  |

|  |  |
| --- | --- |
| COMANDOS COMUNES | |
| $HOME | Devuelve home del usuario |
| PWD | Devuelve path actual (Print Working Directory) |
| Mkdir <nombre Archivo> | Crea directorio |
| Touch <nombre archivo> | Crea archivo |
| Ls | list |
| Echo “resultado: $(var)” | Write |
| Read var | Read () |
|  |  |

|  |  |
| --- | --- |
| ARRAY | |
| Array=() | Inicializar |
| dimL=${#Array[@]} | dimL / largo |
| array+=”$dato” | Push / insertarFinal |
| Unset ‘Array[$dimL-1]’ | Pop |
| Dato=${Array[x]} | Getter |
|  |  |

|  |  |
| --- | --- |
| ARRAY ASOSIATIVO | |
| Declare -A array | Inicializa array |
| Array[“$claveArchivo”]=$archivo | Inserta en el indice ‘clave’ |
| unset array[“$key”] | Da de baja el dato del array |
| ${!array[@]} | Devuelve todas las llaves |
| ${array[@]} | Devuelve los contenidos del array |
| if [ -v array[$key] ] then | Corrobora si la $key esta usada |

|  |  |
| --- | --- |
| Filtros / seleccionadores | |
| Ext=$( echo “$dir” | cut -d “.” -f 2) | Corta Str, -d (divisor “.”), -f (fraction “2”) |
| Echo $var | tr ’1234’ ’abcd’ (| tr ‘Str1’ ‘Str2’) | Cambia cada N° con la letra que tiene misma Pos en Str2 |
| grep |  |
| Awk | -F (separador) |
| cut |  |

|  |  |
| --- | --- |
| Directorios | |
| usr/passwd | Name : psw : UID : GID : INFO : HOME : prog inicio |
| usr/shadow (super user) | Name : Psw : FechaCamPsw::::(7pos) dias sin conectarse: |