

# Introducción al Shell script



Presentado por **Ronald Hernández**  
[www.consultec-ti.com](http://www.consultec-ti.com)

# Módulo 5

## Introducción al Shell Script

### Objetivo

Conocerán las destrezas para manejar y administrar comandos básicos donde podrán ejecutarlos en consola y que permita ejecutar actividades en directorios o ficheros.



# Agenda del día



- Introducción al Shell script
- Terminología básica del Shell script
- Iniciando con Shell script
- Uso de la consola o terminal
- Uso de comandos por consola o terminal
- Comandos de archivos y directorios
- Variables
- Estructura de control
- Uso de Cron y Crontab

# Un poco de historia



Presentado por **Ronald Hernández**

[www.consultec-ti.com](http://www.consultec-ti.com)

# Un poco de historia



**Kenneth Lane  
Thompson**

1971

- Creador de la primera versión del S.O UNIX.
- Creador del primer Shell
- Características iniciales del Shell era una sintaxis compacta para la redirección de comandos entrada/salida.



**William Nelson  
Joy**

1978

- Creador del Shell C y editor de texto vi
- El Shell C permite al usuario escribir y ejecutar comandos.
- También puede leer comandos de un archivo, llamado script
- Principal objetivo del Shell C era que se pareciera al lenguaje C y fuera más interactivo



**David Korn**

1983

- Se hace cargo de algunas características de **Shell C**
- Agrega funciones avanzadas de scripting disponibles en lenguajes más avanzados como Perl
- Es compatible con el shell Bourne



**Brian J. Fox**

1988

- Autor original del Shell bourne-Again (BASH)
- Shell predeterminado en muchas distribuciones Linux, Mac OS.
- El shell más popular entre los usuarios de las distribuciones GNU/Linux

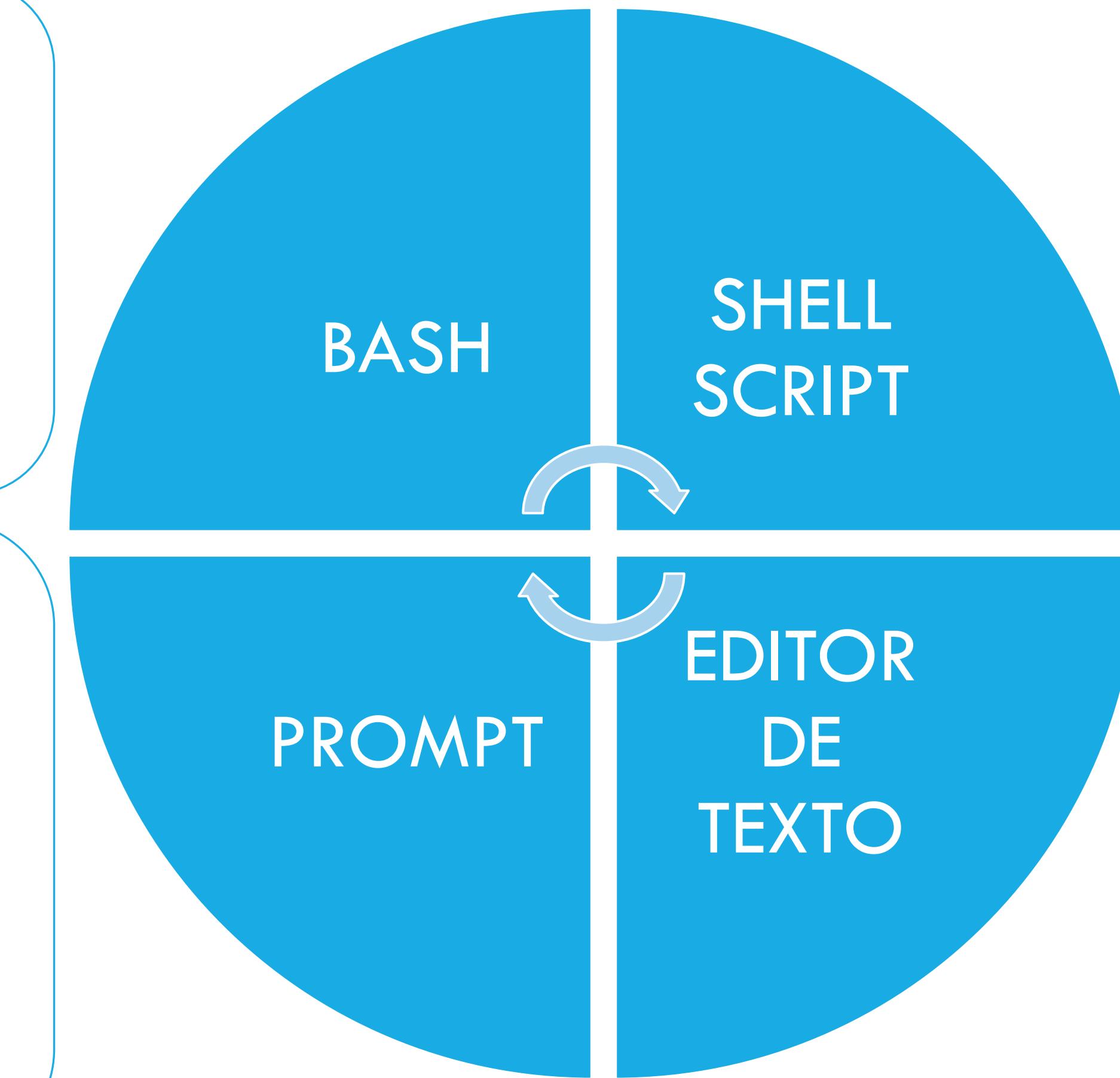
# Terminología básica de shell script



Presentado por **Ronald Hernández**  
[www.consultec-ti.com](http://www.consultec-ti.com)

# Terminología básica de shell script

- Es un intérprete de comandos, el más popular
- Actúa como interfaz entre el kernel Linux y los usuarios.
- Bash no es una terminal.
- Tampoco es la única shell disponible
- Podemos ver los intérpretes disponibles desde /etc/shells

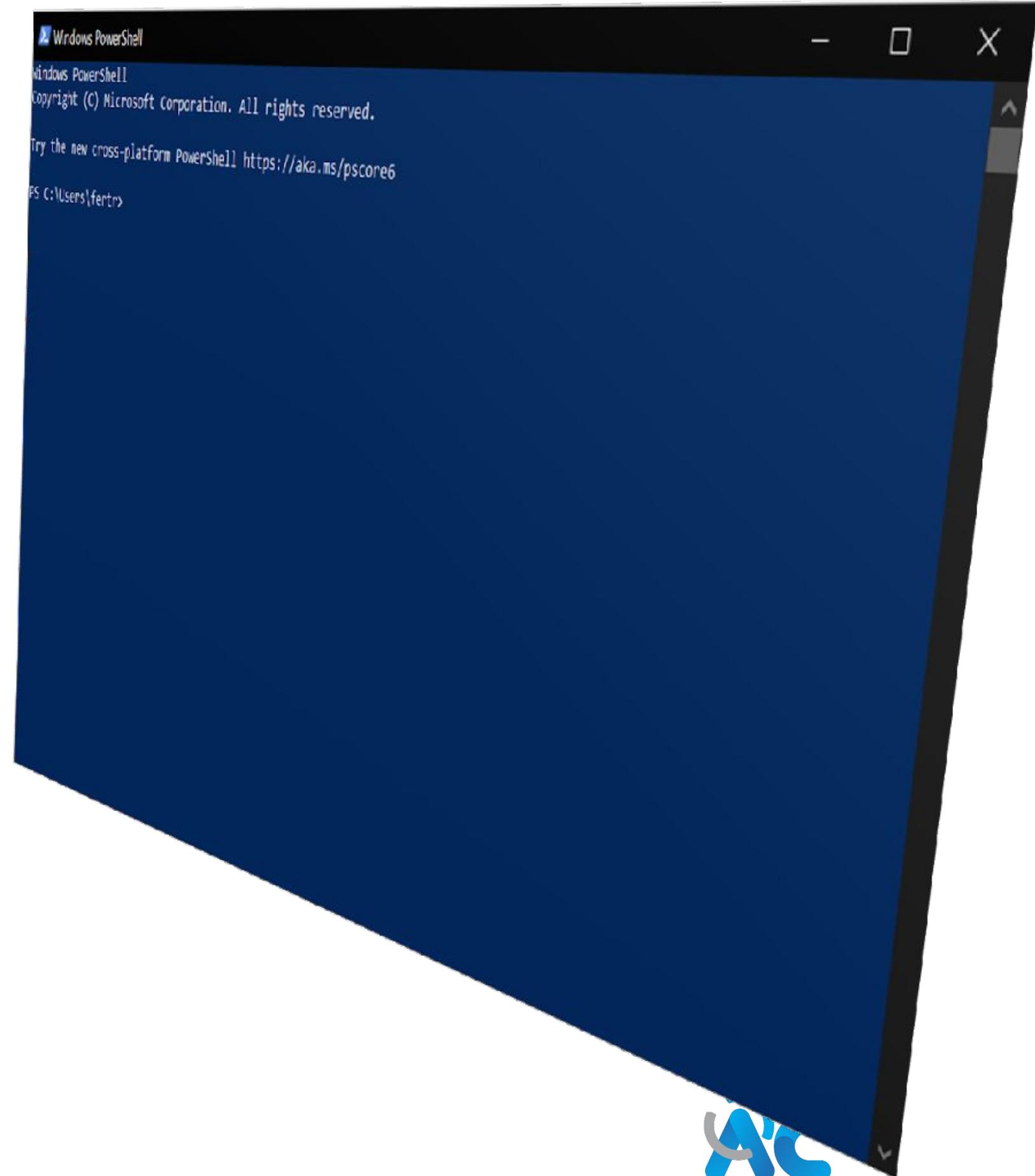


- Archivo que contiene un conjunto de órdenes para realizar una acción, con extensión .sh.
- Permite automatizar procesos y reducir probabilidades de errores
- Acelera las operaciones de forma repetitivas.
- Son ejecutadas por un Shell (CLI o intérprete de comandos) de Unix o Linux

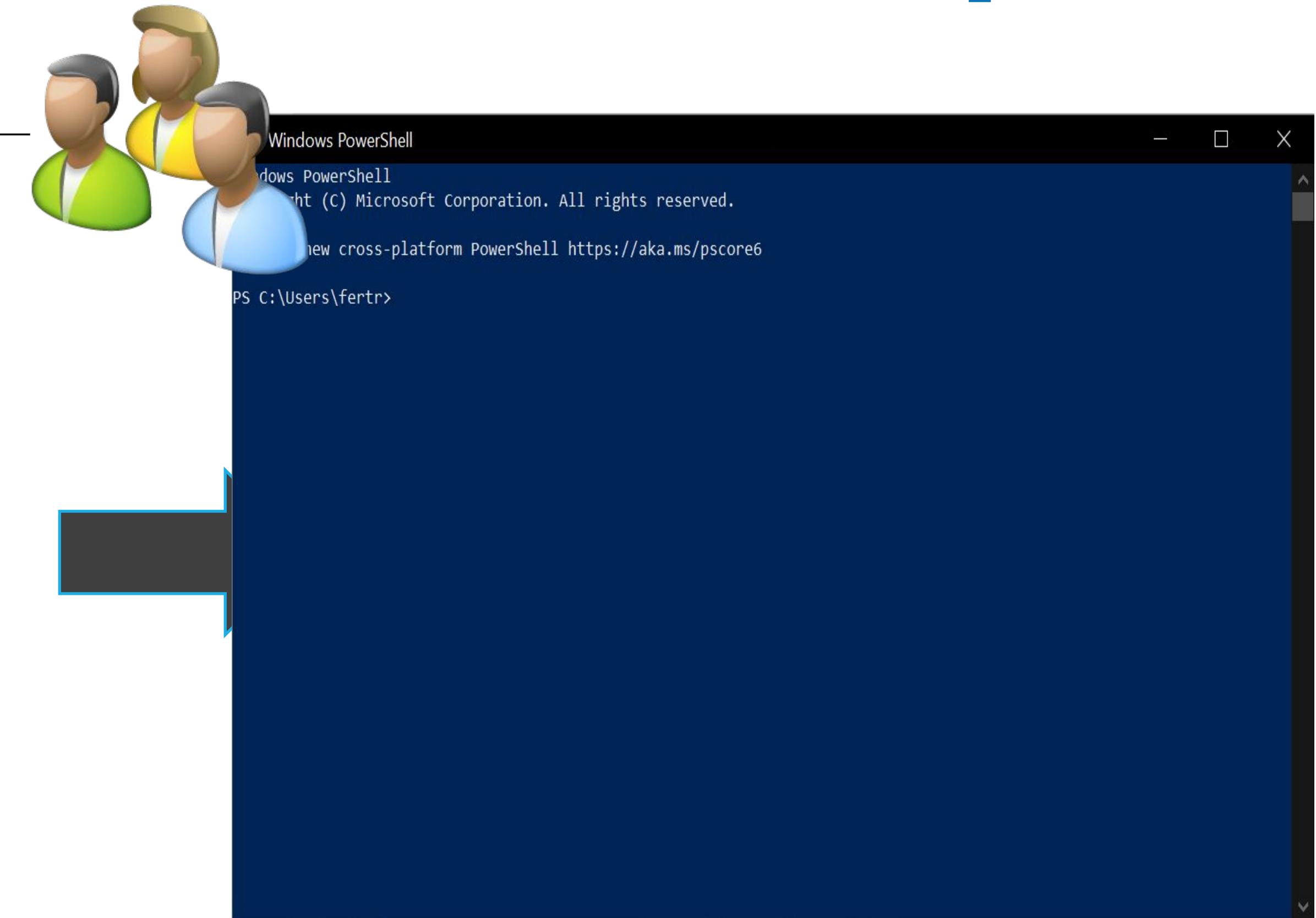
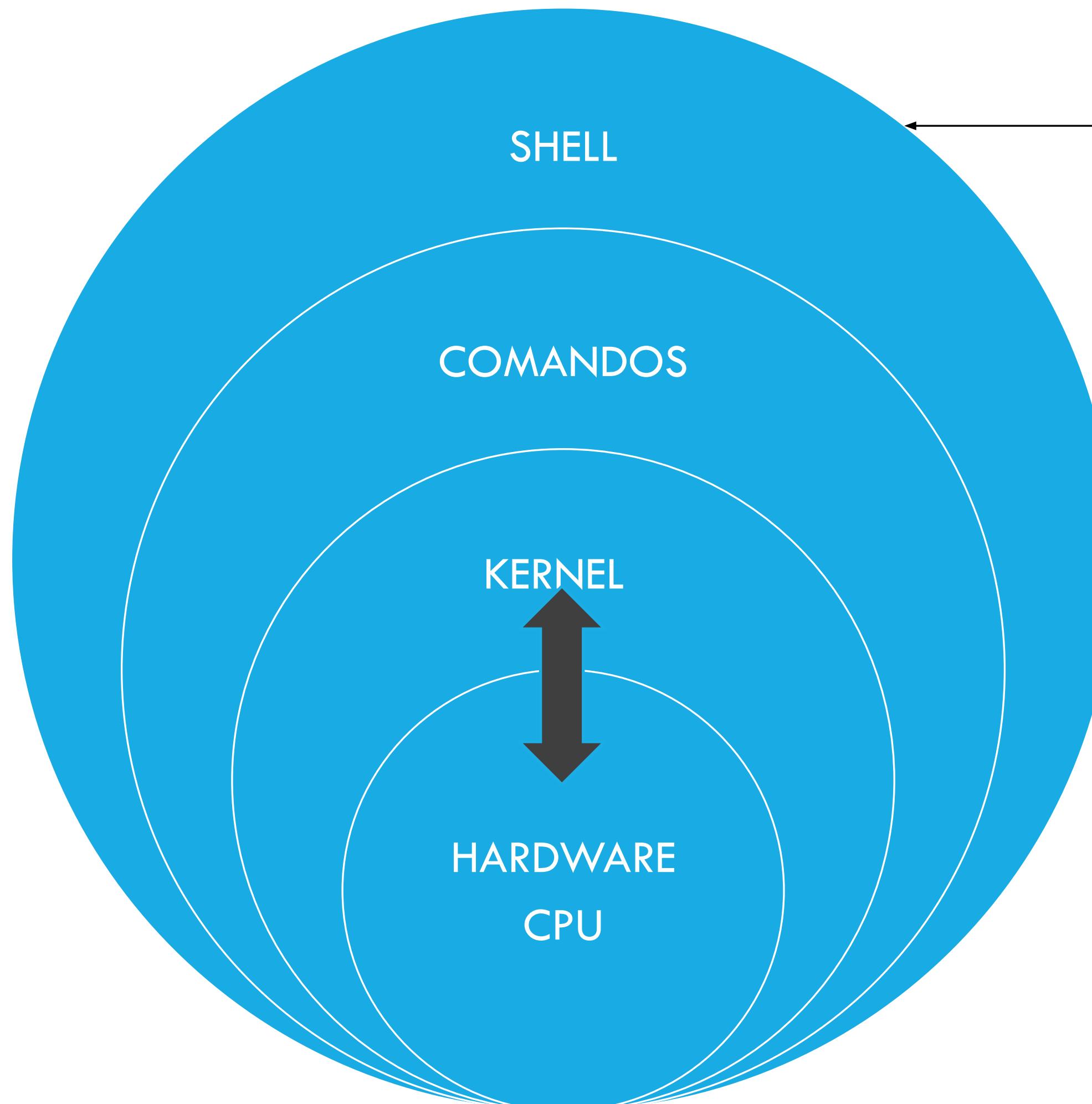
- Programa informático que permite crear y modificar archivos digitales compuestos únicamente por textos sin formato.
- Incluidos en los Sistemas Operativos.
- Por convención, los archivos creados en DOS y Windows Microsoft tienen extensión ".txt". A diferencia de Linux, el usuario tiene libertad total de colocar cualquier extensión.
- Gran variedad de editores de texto.

# Terminología básica de shell script

- **¿Qué es Shell Scripting?**
- Es la técnica (habilidad / destreza) de diseñar y crear Script (archivo de automatización de tareas) mediante un Shell (preferiblemente) de un Sistema Operativo.
- **¿Qué es la Shell?**
- Para cualquier sistema operativo se refiere al interprete de comandos del Sistema Operativo. Por lo general es la “consola”
- **¿Qué es el Bash Shell de GNU/Linux?**
- Es un programa informático cuya función consiste en interpretar órdenes.
- **¿Qué es un Shell Script?**
- Un shell script es un grupo de comandos, funciones y variables. En teoría un shell script es una forma de agrupar secuencias de comandos que corren sin necesidad de que los escribamos en el prompt.



# Terminología básica de shell script

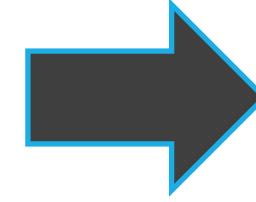
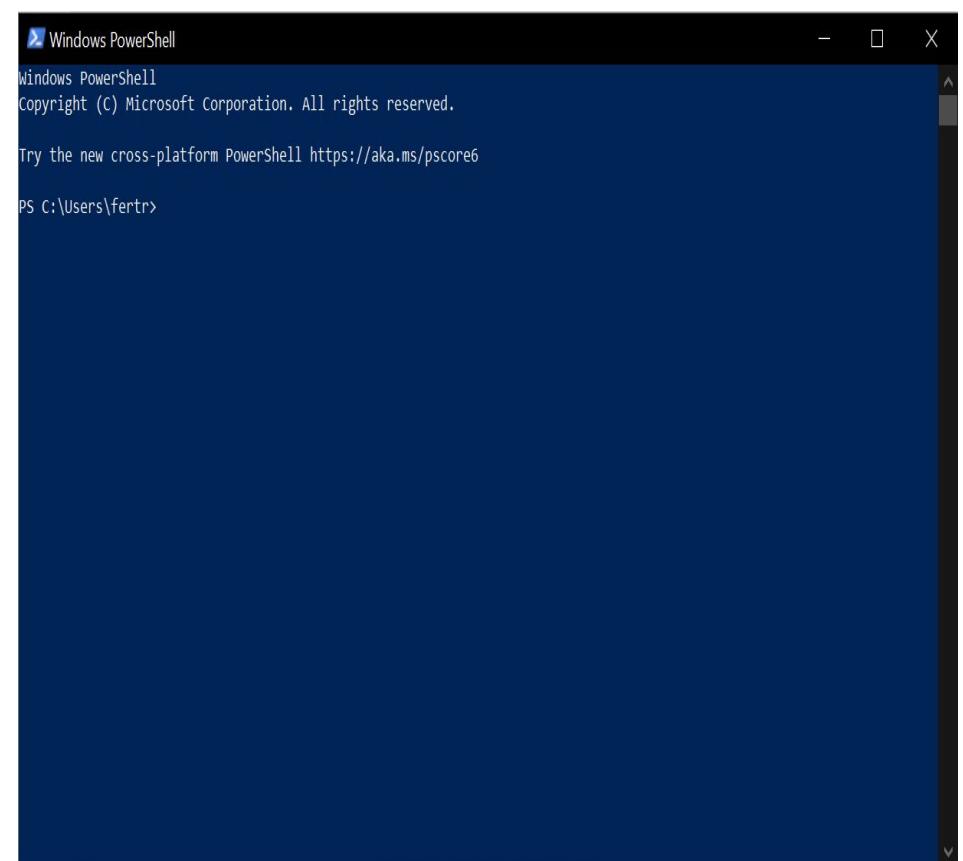


# Iniciando con shell script

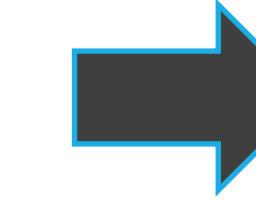
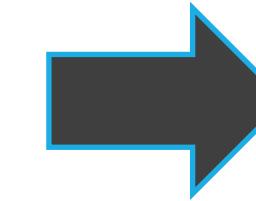


Presentado por **Ronald Hernández**  
[www.consultec-ti.com](http://www.consultec-ti.com)

# ¿Qué es un shell script?



```
#!/bin/bash
# Script de hola mundo
echo "Hola mundo"
```



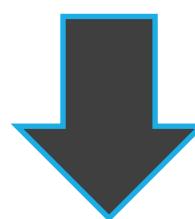
Es un simple fichero de texto que contiene uno o varios comandos. Habitualmente, los shell scripts tienen la extensión ".sh".

**script01.sh**  
**#!/bin/bash**  
**# Script #01 de hola mundo**  
echo "Hola mundo"



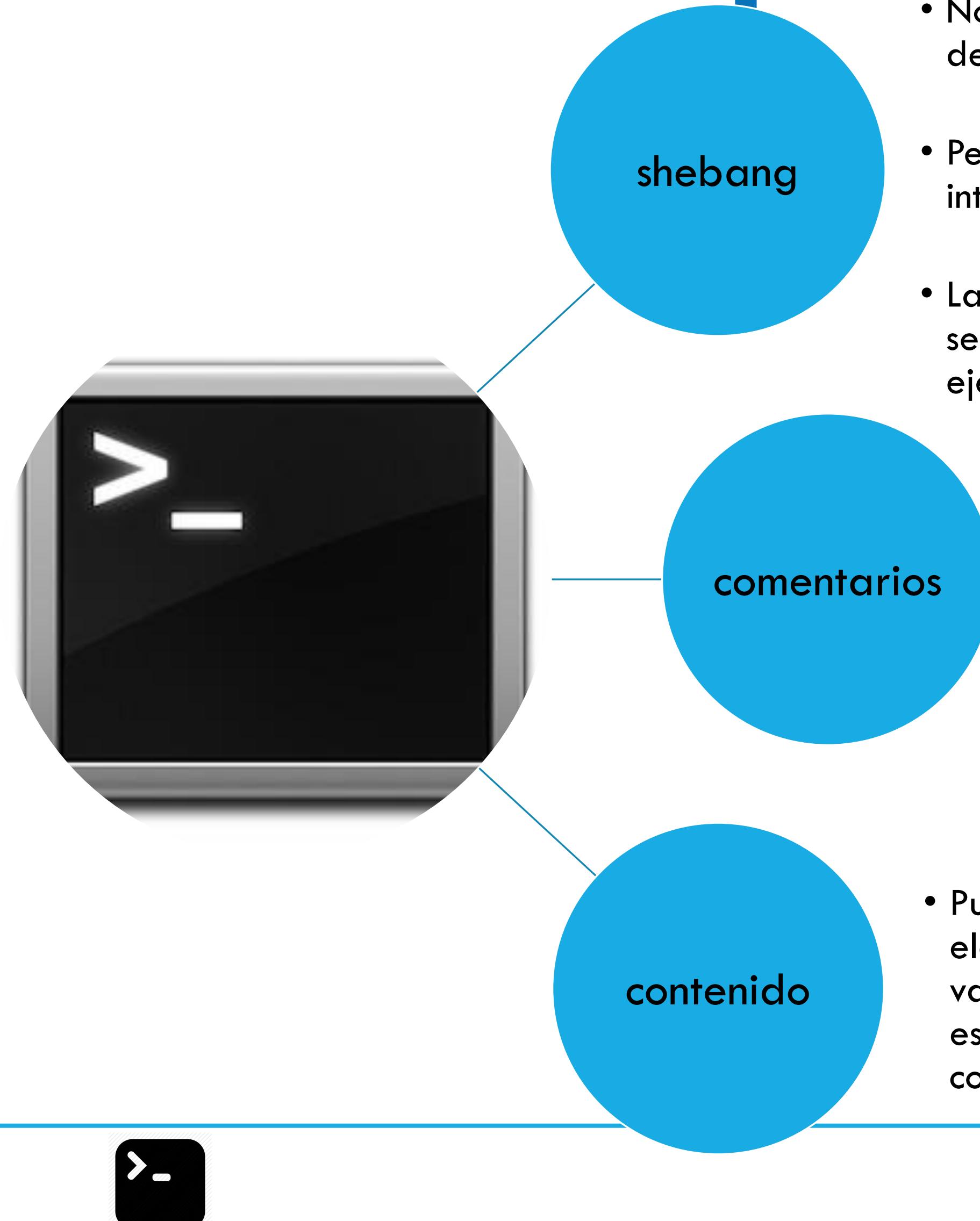
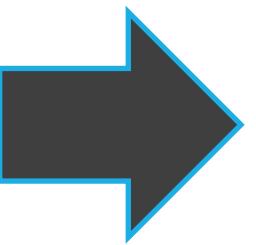
# Estructura de un shell script

La estructura básica de un shell-script es la siguiente:



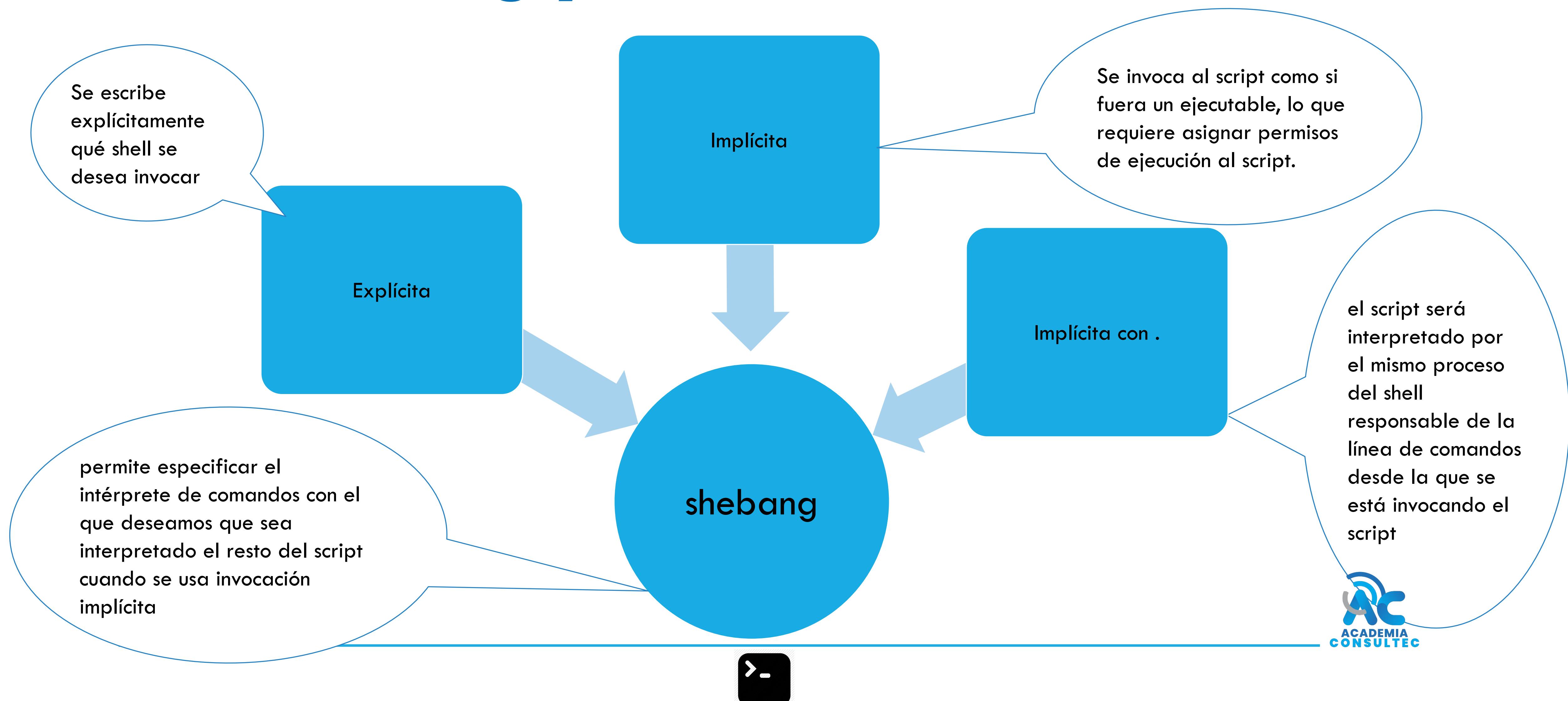
**script02.sh**

```
#!/bin/bash
# Esto no se interpreta
echo Hola
ps w
echo "Proceso lee el script:
$$"
```



- Nombre que recibe el par de caracteres #!
- Permite especificar el intérprete de comandos
- La sintaxis de esta línea es la secuencia #! seguida del ejecutable del shell deseado
- Sirve para documentar las partes de ejecución del script
- Pueden utilizarse múltiples elementos (comandos, variables, funciones, estructuras de control, comentarios)

# El shebang y como se invoca



# Rutas de interprete

1.- Crear el siguiente script

**script03.sh**

```
echo "Contenido carpeta personal:"  
ls ~/
```

2.- Compruebe que el **script03.sh** tiene permisos de ejecución generales (si no los tuviese, para asignárselos bastaría ejecutar **chmod +x script03.sh**).

3.- Invoque el script para que sea interpretado, usando por ejemplo el comando:

**./script03.sh**

# Rutas de interprete

1.- Crear el siguiente script

**script04.sh**

```
#!/bin/dash
# Esto es un comentario y no se interpreta
echo Hola
ps w
echo "Proceso lee el script: $$"
```

2.- Invoque dicho script mediante los distintos métodos de invocación **script04.sh**

- Explícita:

/bin/sh **script04.sh**

/bin/dash **script04.sh**

/bin/bash **script04.sh**

- Implícita con . :

. **script04.sh**

- Implícita: compruebe que el script tiene permiso de ejecución y ejecútelo con:

./**script04.sh**



# Break 15 minutos



Presentado por **Ronald Hernández**

[www.consultec-ti.com](http://www.consultec-ti.com)

# Uso de la consola o terminal

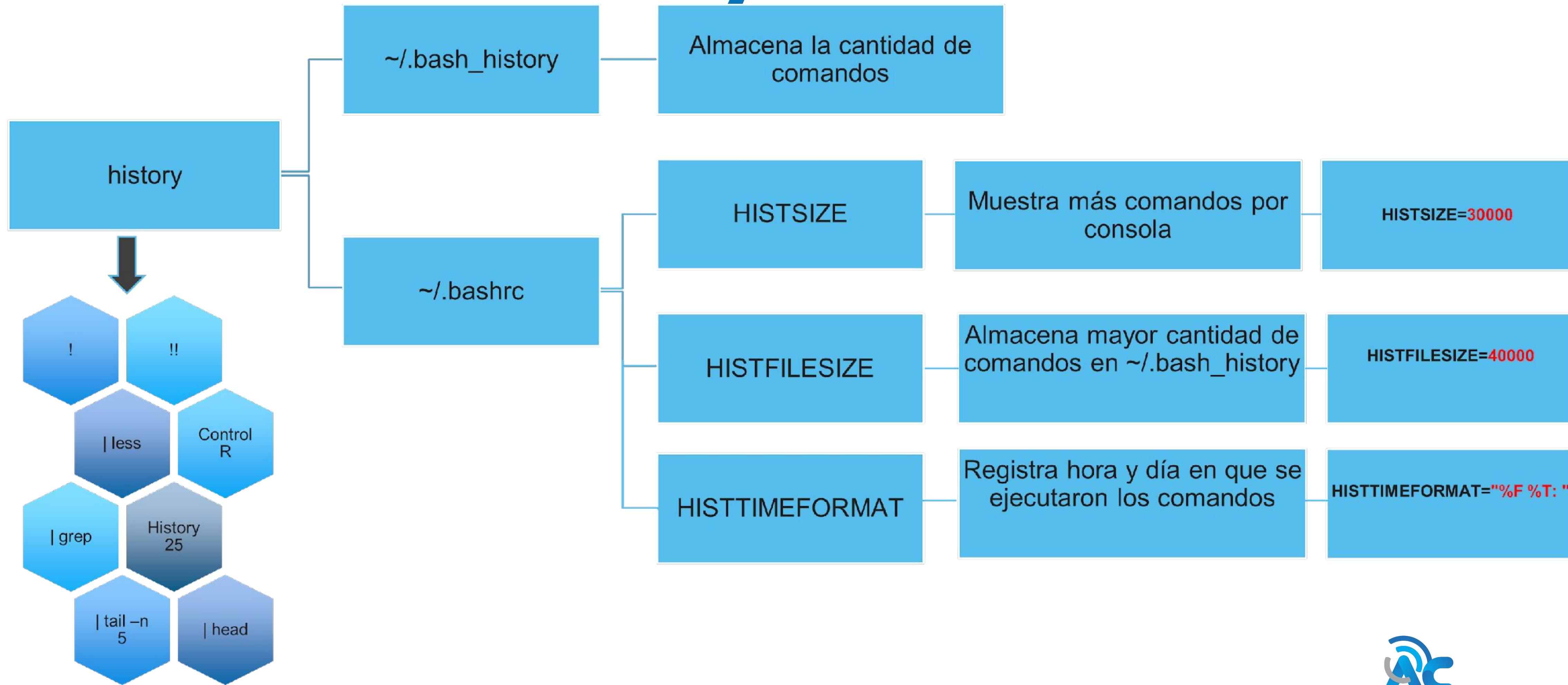


---

Presentado por **Ronald Hernández**

[www.consultec-ti.com](http://www.consultec-ti.com)

# Comando History



# Comando History

A continuación algunas formas básicas de usar este comando “history”

!! = Se ejecuta el último comando listado en el histórico.

\* Ejemplo:

!! (se ejecuta 546 history)

! = Se ejecuta alguno de los comando listado en el histórico.

\* Ejemplo:

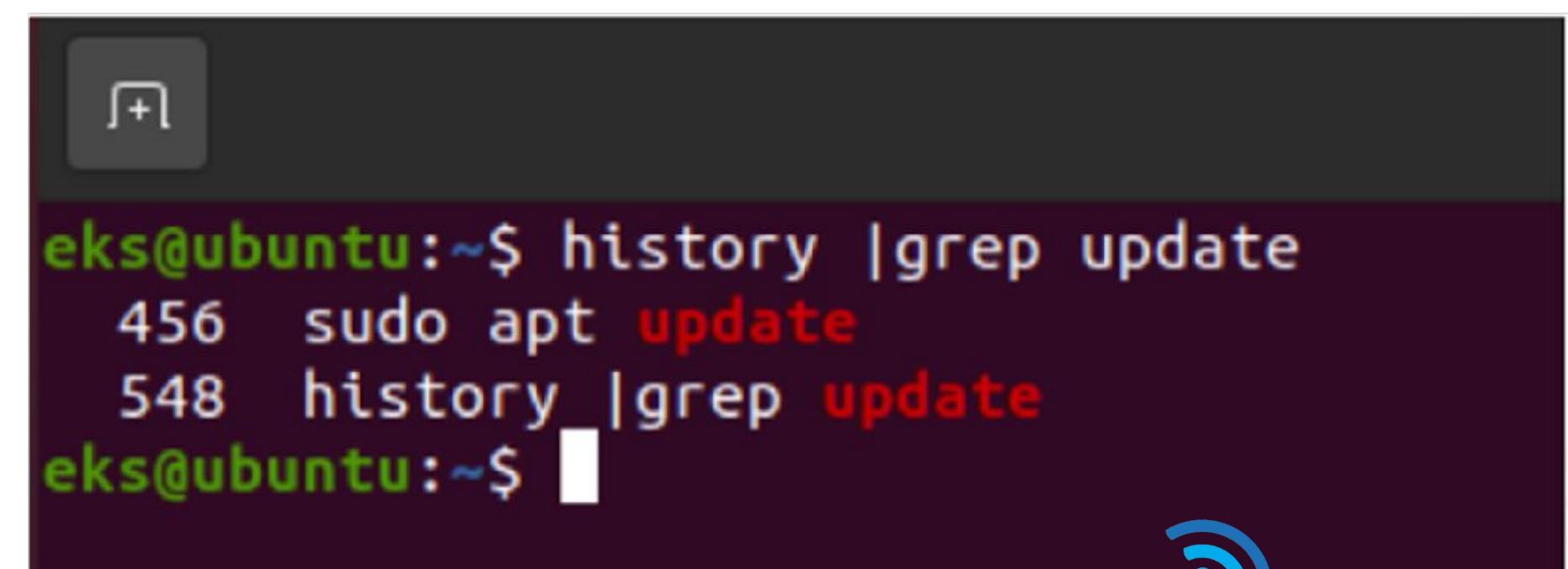
!531 (clear)

history |grep update = Se mostraran todos los comando con el termino “update”

\* Ejemplo:

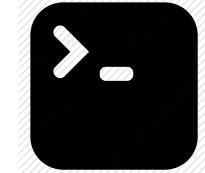
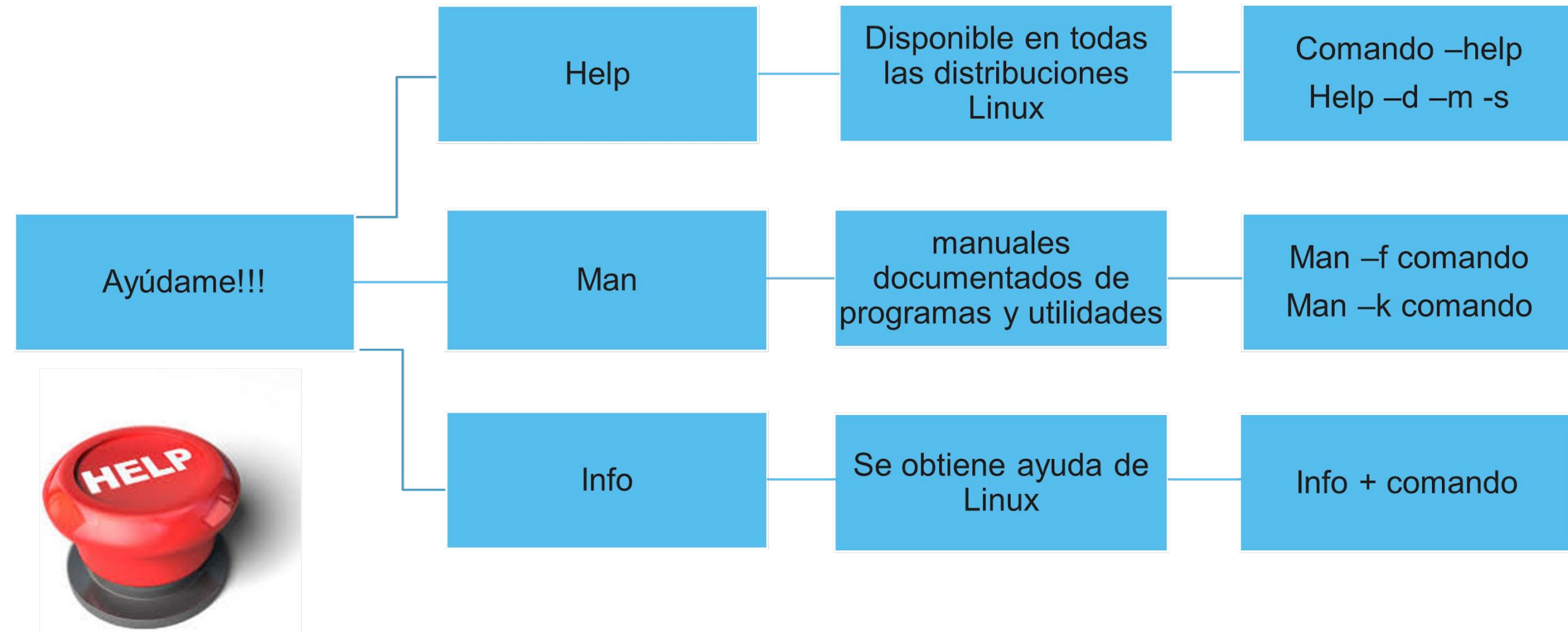
!! (se ejecuta 546 history)

```
530  history 25
531  clear
532  history 25
533  a=hola
534  echo $a
535  clear
536  hsitory
537  clear
538  history
539  cat ~/.bash_history
540  clear
541  ls -la
542  nano .bash_history
543  clear
544  cat ~/.bash_history
545  clear
546  history
eks@ubuntu:~$ !531
```



The screenshot shows a terminal window with a dark background and light-colored text. At the top, there's a small icon of a window with a plus sign. The terminal prompt is "eks@ubuntu:~\$". Below it, the command "history |grep update" is entered. The output shows two entries from the history: "456 sudo apt update" and "548 history |grep update". The word "update" is highlighted in red in both the command and the output. The terminal ends with another prompt "eks@ubuntu:~\$".

# Comando Help



# Comando Grep

## Utilidad

- Búsquedas de palabras, símbolos, letras, números y cualquier patrón dentro de un archivo

## Sintaxis

- `grep [opciones] pattern [ARCHIVO]`
- **grep**: la instrucción de comando
- **[opciones]**: modificadores del comando
- **pattern**: el patrón que queremos encontrar con la búsqueda
- **[ARCHIVO]**: el archivo en el que estás realizando la búsqueda

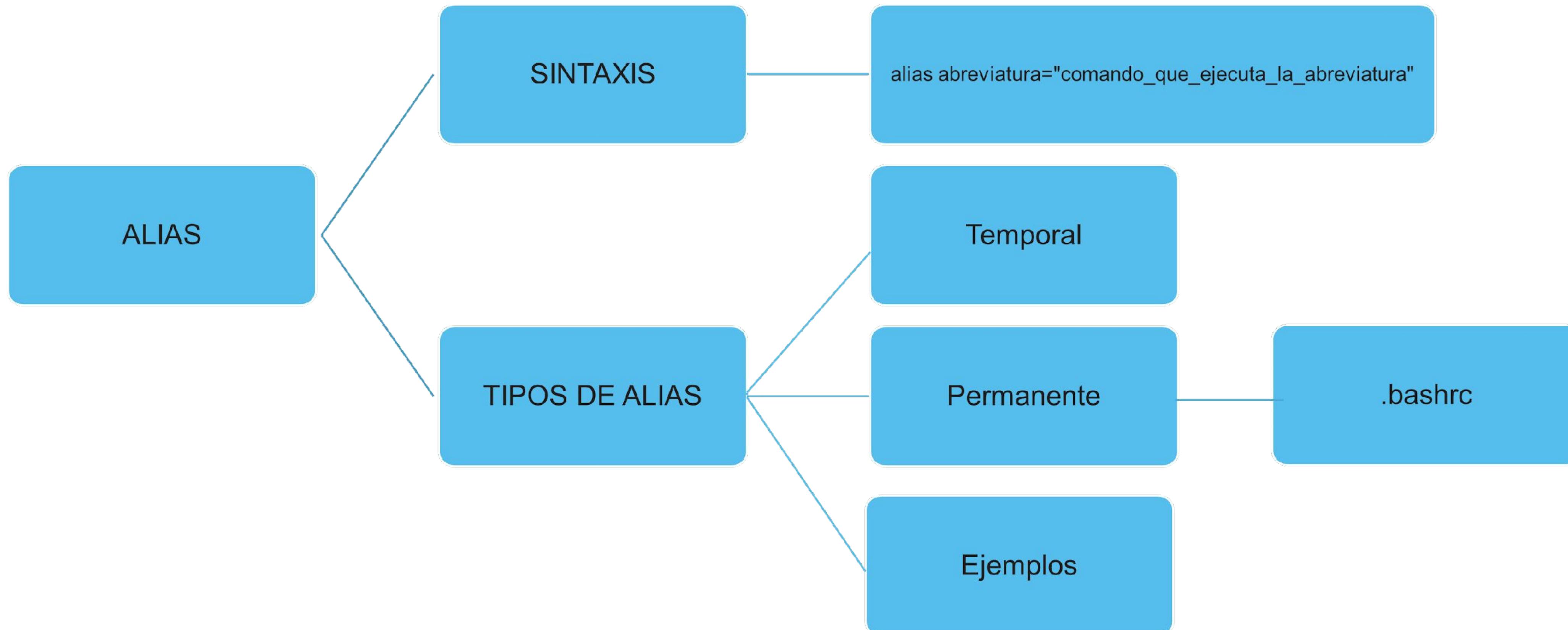
## Opciones

- **-i**: la búsqueda no distinguirá entre mayúsculas y minúsculas. Es decir, si quieres buscar la palabra «auto» será lo mismo que «AUTO»
- **-c**: solo mostrará el número de líneas que coinciden con el patrón buscado
- **-r**: habilita la búsqueda recursiva en el directorio actual
- **-n**: busca líneas y precede cada línea coincidente con un número de línea.
- **-v**: con esta opción, se nos muestran las líneas que no coinciden con el patrón que hemos buscado

## Ejemplos

- `grep búsqueda archivo`
- `grep -c búsqueda archivo`
- `grep -i búsqueda archivo`
- `grep -l palabra_a_buscar /*`

# Comando Alias



# Comando Alias

Ejemplo de un alias temporal

```
eks@ubuntu:~$ alias aliastemporal="echo Bienvenido al curso de shell script"
eks@ubuntu:~$ aliastemporal
Bienvenido al curso de shell script
eks@ubuntu:~$ █
```

Ejemplo de un alias permanente.

- Editar archivo .bashrc y agregar el alias

```
# alias de pruebas
alias aliaspermanente="echo Bienvenido al curso de shell script"
```

- Guardar cambios y cerrar la consola. Abrir una nueva y ejecutar nombre del alias

```
eks@ubuntu:~$ aliaspermanente
Bienvenido al curso de shell script
eks@ubuntu:~$ █
```

# Comandos de archivos y directorios

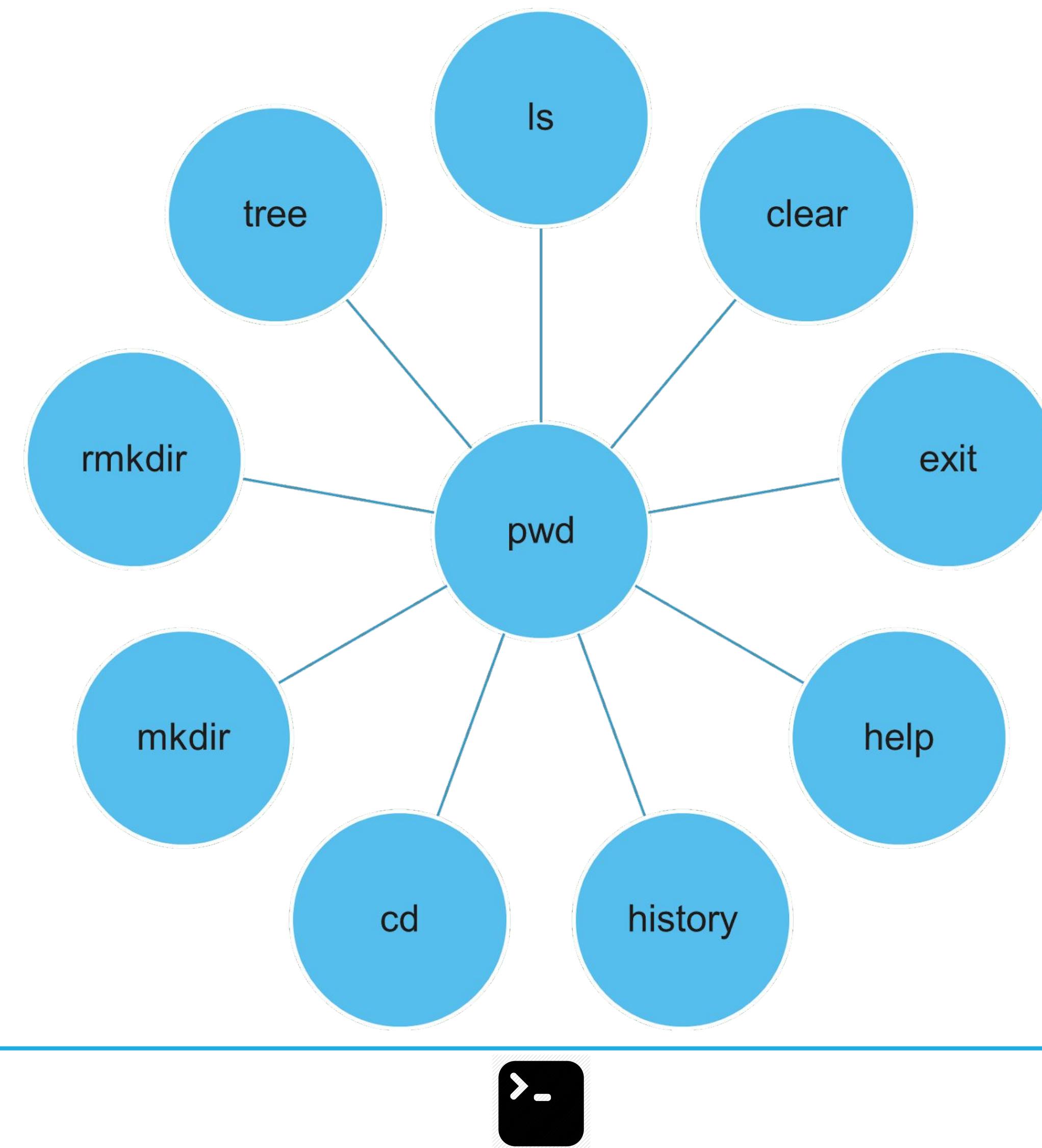


---

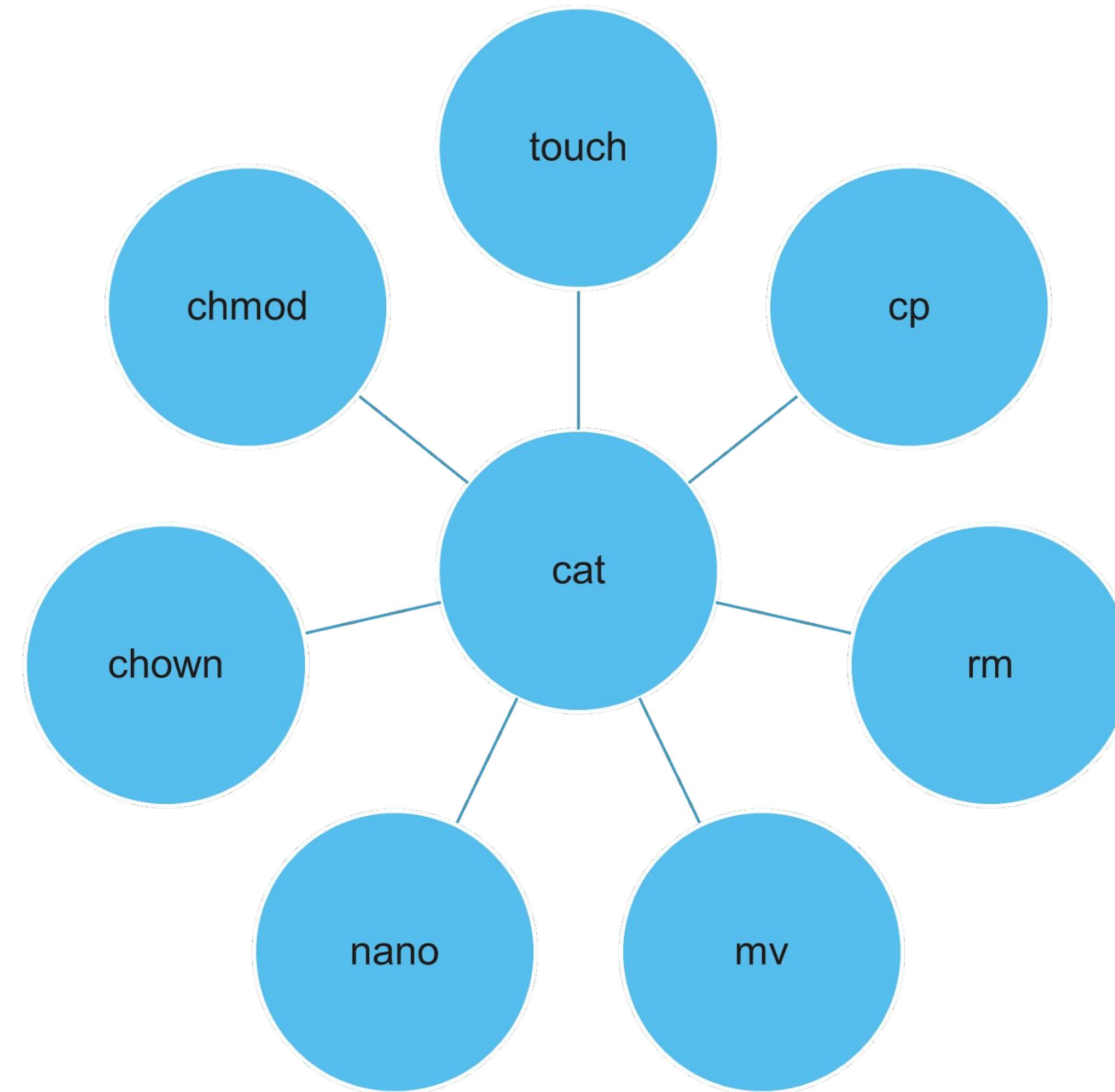
Presentado por **Ronald Hernández**

[www.consultec-ti.com](http://www.consultec-ti.com)

# Comandos para manejo de directorio



# Comandos para manejo de archivos



# Hora de almuerzo



Presentado por **Ronald Hernández**  
[www.consultec-ti.com](http://www.consultec-ti.com)

# Variables

## ¿Qué es una variable?

- lugar seguro donde guardas una información o valor

## Sintaxis

- `nombre_variable=valor_variable`
- Leer una variable
- `echo $nombre_variable`
- Recuperar valor de una variable
- `$nombre_variable`

## Nombres de una variable

- Sólo puede contener caracteres alfanuméricos y guiones bajos
- El primer carácter debe ser una letra del alfabeto o “\_” (este último caso se suele reservar para casos especiales).
- No pueden contener espacios.
- Las mayúsculas y las minúsculas importan, “a” es distinto de “A”.

## Las comillas en bash

- **Comillas simples** se guarda en la variable **literalmente** lo que hay entre ellas.
- **Comillas dobles** se **interpreta** el contenido.

# Ejemplos de Variables

## **script04.sh**

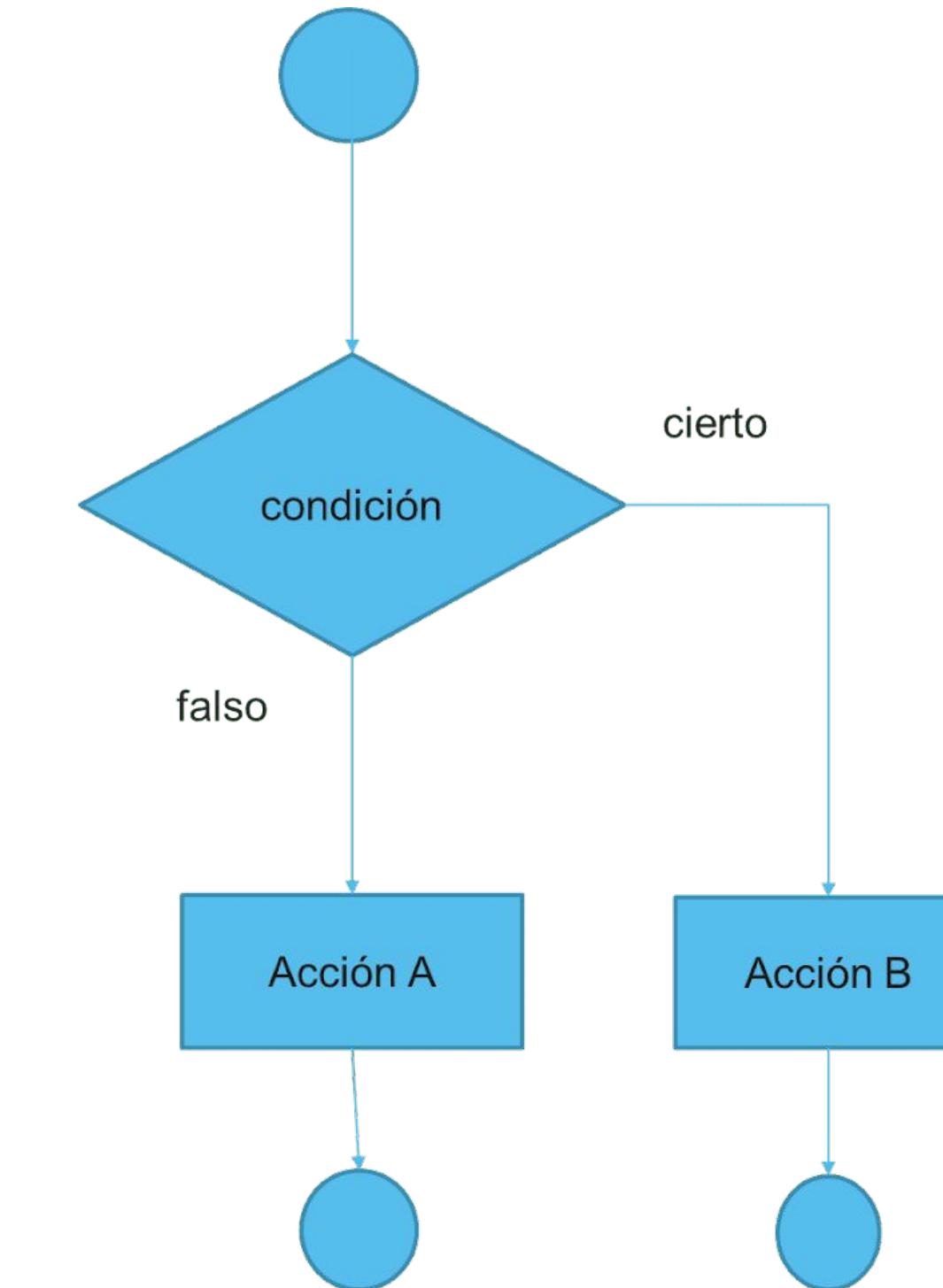
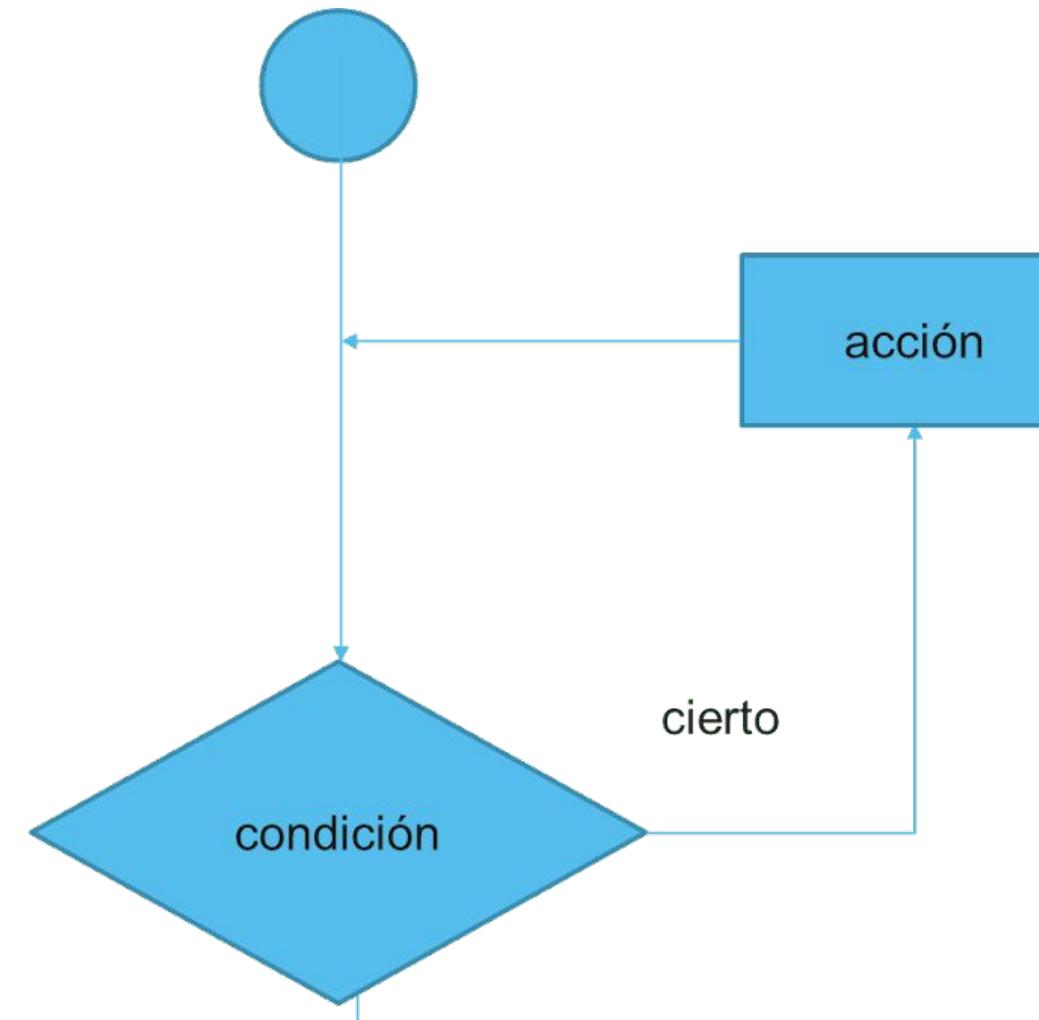
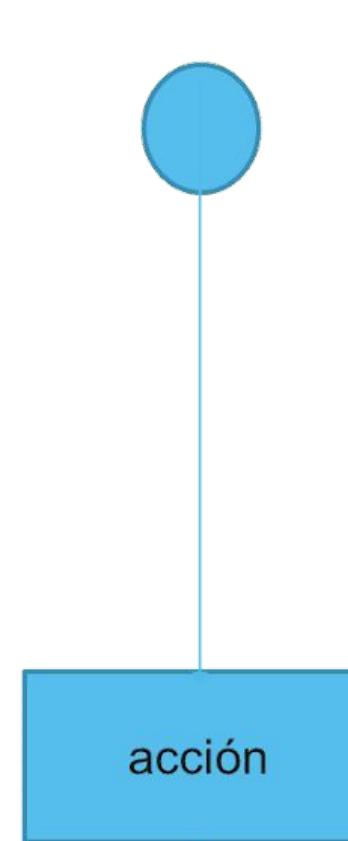
```
#!/bin/bash
variable1=juan
variable2='Esta es la casa de $variable1'
echo variable2
```

## **script05.sh**

```
#!/bin/bash
variable1=juan
variable2="Esta es la casa de $variable1"
echo variable2
```



# Control de flujos



# Control de flujo: FOR

## Bucle FOR

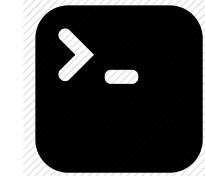
- El bucle **for** permite iterar /repetir sobre una serie de elementos de una cadena, lista, rangos, etc..

## Sintaxis

- **for** VARIABLE **in** LISTA\_VALORES;
- do
- COMANDO 1
- COMANDO 2
- ...
- COMANDO N
- done

## Lista de valores

- Puede ser un rango numérico:
- **for** VARIABLE **in** 1 2 3 4 5 6 7 8 9 10;
- **for** VARIABLE **in** {1..10};
- serie de valores:
- **for** VARIABLE **in** file1 file2 file3;



# Ejemplo: FOR

## **script06.sh**

```
#!/bin/bash
for numero in {1..20..2};
do
    echo Este es el número: $numero
done
```



# Control de flujo: IF

## Condicionales IF

- El control de flujo IF permite evaluar condiciones cuando se cumplen y no se cumplen a través de condiciones numéricos o cadenas de textos.

## Sintaxis

- Sobre un condicional:
  - **If** [[CONDICIÓN]];  - **then**
  - COMANDO 1 si se cumple la condición
  - **Fi**
- Si una condición no se cumple:
  - **If** [[CONDICIÓN]];  - **then**
  - COMANDO 1 si se cumple la condición
  - **else**
  - COMANDO 2 si no se cumple la condición
  - **Fi**
- más condiciones concatenando más if:
  - **If** [[CONDICIÓN 1]];  - **then**
  - COMANDO 1 si se cumple la condición 1
  - **Elif** [[CONDICIÓN 2 ]];  - **then**
  - COMANDO 2 si se cumple la condición 2
  - **else**
  - COMANDO 3 si no se cumple la condición 2
  - **fi**

## Condicionales

- **Numérico:**
- Operador significado
  - -lt menor que (<)
  - -gt mayor que (>)
  - -le menor o igual que (≤)
  - -ge mayor o igual que (≥)
  - -eq igual (==)
  - -ne no igual (!=)
- **Cadenas de texto:**
- Operador significado
  - = igual, las dos cadenas de texto son exactamente idénticas
  - != no igual, las cadenas de texto no son exactamente idénticas
  - < es menor que (en orden alfabéticoASCII)
  - > es mayor que (en orden alfabéticoASCII)
  - -n la cadena no está vacía
  - -z la cadena está vacía

# Control de flujo: IF

## Condicionales IF

- El control de flujo IF permite evaluar condiciones cuando se cumplen y no se cumplen a través de condiciones numéricos o cadenas de textos.

## Sintaxis

- Sobre un condicional:
  - **If** [[CONDICIÓN]];
    - then
    - COMANDO 1 si se cumple la condición
    - **Fi**
  - Si una condición no se cumple:
    - **If** [[CONDICIÓN]];
      - then
      - COMANDO 1 si se cumple la condición
      - **else**
      - COMANDO 2 si no se cumple la condición
      - **Fi**
    - más condiciones concatenando más if:
      - **If** [[CONDICIÓN 1]];
        - then
        - COMANDO 1 si se cumple la condición 1
        - **Elif** [[CONDICIÓN 2 ]];
          - then
          - COMANDO 2 si se cumple la condición 2
          - **else**
          - COMANDO 3 si no se cumple la condición 2
          - **fi**

## Condicionales

### Con archivos:

- Operador
  - -e name Devuelve true si name existe
  - -f name name es un archivo normal (no es un directorio)
  - -s name name NO tiene tamaño cero
  - -d name name es un directorio
  - -r name name tiene permiso de lectura para el user que corre el script
  - -w name name tiene permiso de escritura para el user que corre el script
  - -x name name tiene permiso de ejecución para el user que corre el script

# Ejemplo: IF\_then

## if\_then.sh

```
#!/bin/bash
set -e

echo ' Adivina el valor numerico de la
variable'
read variable #entrada de datos

if [ $variable = 1 ]; then
    echo 'Acertaste'
    exit 0
fi

echo 'No acertaste'
```



# Ejemplo: IF\_else

## **if\_else.sh**

```
#!/bin/bash

echo 'Adivina el valor numerico de la
variable'
read variable #entrada de datos

if [ $variable = 1 ];then
    echo 'Acertaste'
    exit 0
else
    echo 'No acertaste'
    exit
fi
```



# Ejemplo: IF\_then\_elif\_else



## **if\_then\_elif\_else.sh**

```
#!/bin/bash

echo ' Adivina el valor n\'umerico de la
variable'
read variable

if [ $variable = 1 ]; then
    echo 'Acertaste'
    exit 0
elif [ $variable = 2 ]; then
    echo 'casi!'
else
    echo 'No acertaste'
fi

exit 0
```



# Control de flujo: WHILE

## Bucle While

- El ciclo while verifica una condición antes de cada iteración y ejecuta el ciclo hasta que se cumpla la condición

## Sintaxis

- while [condition];
- do
- commands
- done

## Condicionales



# Ejemplo: WHILE

## **while.sh**

```
i=0
while [ $i -lt 10 ]
do
    echo 'Pruebas de control de flujo
while!!!'
    ((i++))
done
```



# Control de flujo: UNTIL

## Bucle Until

- El ciclo until se ejecutará el ciclo hasta que se cumpla la condición

## Sintaxis

- until [condition];
- do
- commands
- done

## Condicionales



# Ejemplo: UNTIL



## until.sh

```
i=10
until [ $i -lt 0 ]
do
    echo $i
    ((i--))
done
```



# Break 15 minutos



Presentado por **Ronald Hernández**

[www.consultec-ti.com](http://www.consultec-ti.com)

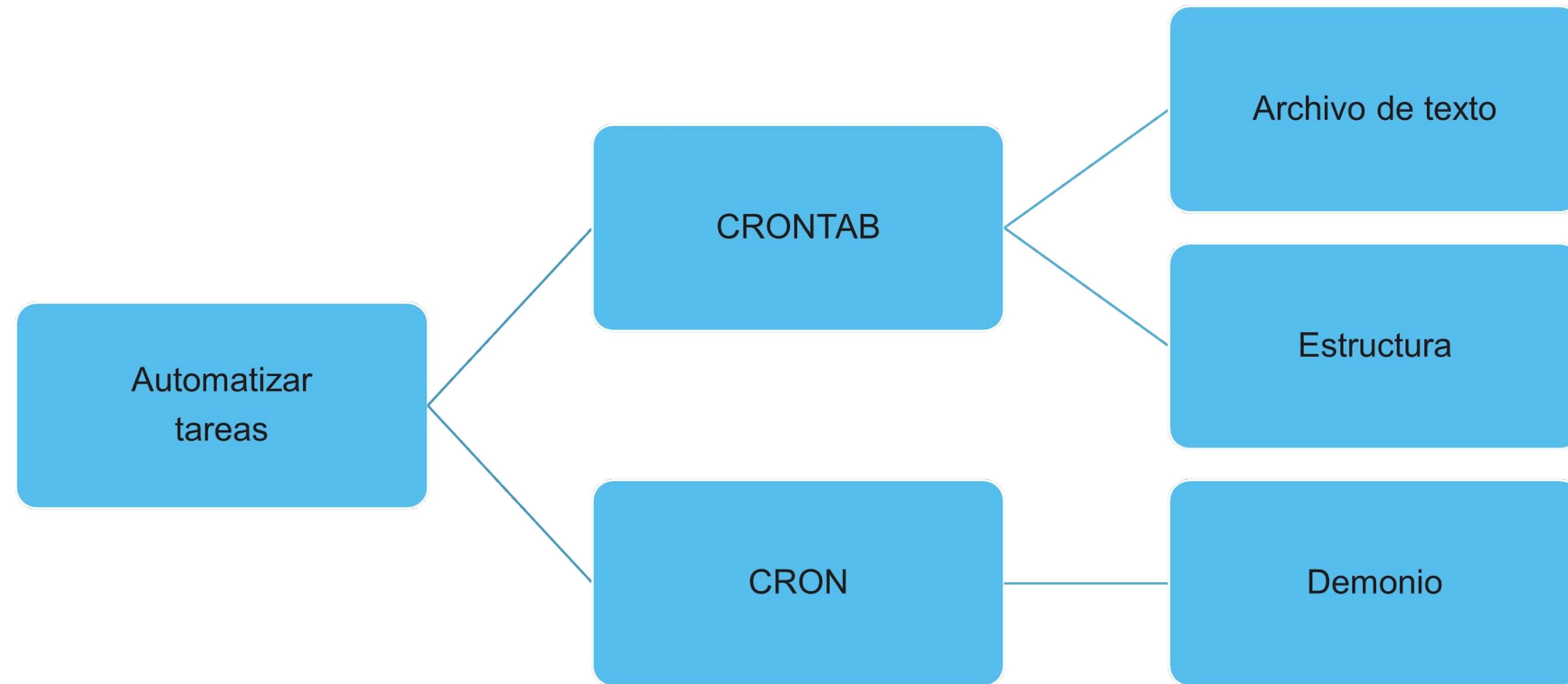
# Uso de CRON y CRONTAB



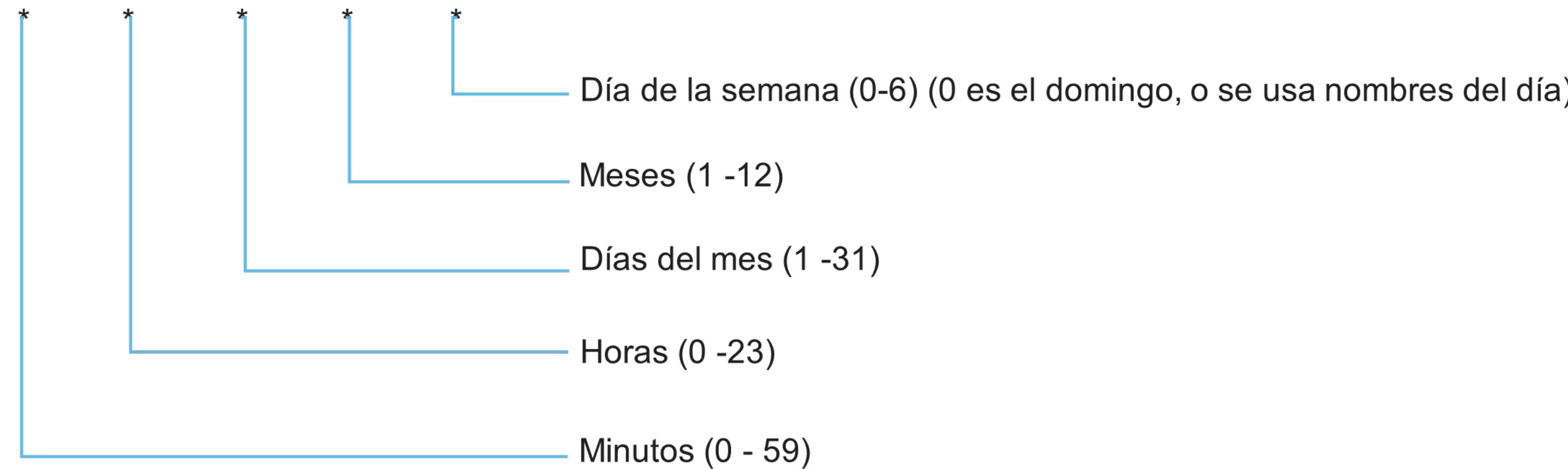
Presentado por **Ronald Hernández**

[www.consultec-ti.com](http://www.consultec-ti.com)

# Uso de CRON y CRONTAB



# Estructura de un CRONTAB



```
00 19 * * * usuario /ubicacion/del/script/consulta.sh
```

# Ejemplo: CRONTAB

**1. Programe un cron para que se ejecute a las 5 AM Todos los días**

```
0 5 * * * /scripts/job.sh
```

**2. Programe un cron para que se ejecute dos veces al día a las 6 AM y a las 6 PM.**

```
0 6,18 * * * /scripts/job.sh
```

**3. Programe un cron para que se ejecute cada minuto**

```
* * * * * /scripts/job.sh
```

**4. Programe un cron para que se ejecute en cada Monday a las 7 PM.**

```
0 19 * * mon /scripts/job.sh
```

**5. Programe un cron para que se ejecute cada 15 minutos.**

```
*/10 * * * * /scripts/job.sh
```

**6. Programe un cron para que se ejecute en los meses seleccionados**

```
* * feb,jun,oct * /script/job.sh
```

**7. Ejecute el script de shell /home/script/backup.sh el 4 de marzo a las 7:25 AM**

```
25 7 4 3 * /home/script/backup.sh
```



# Gracias

¡Nos vemos pronto!