U4. Administración de Sistemas Operativos Linux

Parte I. Scripts

Implantación de Sistemas Operativos

Índice

- Shell
- Scripts
- Instrucciones condicionales
  - If
  - Operadores
  - Case
- Instrucciones repetitivas
  - For
  - Forin
  - While
  - Until
- Funciones
- Depuración

La shell

- La shell o intérprete de órdenes permite ejecutar órdenes del sistema y ofrece un entorno de programación shell script.
- Tenemos distintos tipos de intérpretes de órdenes:
  - Sh: Bourne Shell (original)
  - Bash: Shell de Bourne mejorado (es el que se emplea por defecto)
- Se especifica el shell por defecto a emplear en /etc/passwd. Toda la información del usuario está en este fichero. El último campo es el Shell de ese usuario.
- Podemos lanzar otra shell en cualquier momento. Por ejemplo:
  /bin/sh
- Con otros programas, primero tenemos que compilar el programa y después ejecutarlo. El Shell directamente ejecuta, no hay que compilar. Un script es un programa interpretado y ejecutado por la shell.

La shell

Variables

- Son las variables que creamos nosotros y siempre van en minúsculas, para diferenciarlas de las variables de entorno que van en mayúsculas.
- Las variables pueden empezar por letra o _ y pueden incluir letras, números o el carácter _
- Tenemos 2 tipos de variables:
  - locales o del shell visibles con **set**
    - En este caso vemos todas las variables. Usar el comando con | less o | grep, porque el fichero es muy largo
  - globales o de entorno visibles con **env**
    - En este caso solo vemos las variables de entorno
- **Para ver el valor de una variable SIEMPRE pondremos $**

La shell

Variables

- Las operaciones con variables son:
  - Asignar valor: **variable=valor** (OJO! El igual va sin espacios)
  - Recuperar valor: **echo $variable** o en expresiones más largas ${variable}
    - Ejemplo: A=Jose; echo “Hola ${A}, ¿qué tal?”
  - Obtener longitud (nº caracteres) : echo ${#variable} (las llaves son necesarias)
  - Eliminar: unset variable
  - Definir como sólo lectura: readonly variable

- Las variables que creamos en el script se quedan en el script y no las veremos desde el terminal. Estas variables existirán mientras que se ejecute el script, después dejarán de existir.

La shell

Variables

- Si creamos una variable localmente, en la terminal, será visible en la shell en la que la hemos creado. Y dejará de existir cuando cerremos la consola.
- Para que una variable sea visible en las terminales hijas de aquella terminal donde la hemos creado tendremos que utilizar el comando export.
    - Ej: **export saludo = “Hola”;**
- Para crear consolas hijas gnome-terminal.

Recordatorio:

Man → manuales (para buscar dentro de un manual shift+7 (n next, p inicio, q salir)

--help → ayuda

La shell

Variables de entorno

- Cada shell ejecuta un entorno de ejecución determinado, visible para ese shell y para sus hijos. Las variables de entorno definen las condiciones del entorno del shell. Son las variables de sistema y que se cargan cuando arranca éste.
- Siempre se escriben en mayúsculas.

HOME	Directorio del usuario	**MANPATH**	Ruta de páginas de ayuda
SHELL	Shell por defecto	**PS1/PS2**	Prompt primario y secundario
USERNAME	Nombre del usuario	**LANG**	Ubicación y codificación
PWD	Directorio actual	**LC_***	Localización
PATH	Ruta de ejecutables	**LANGUAGE**	Idioma

La shell

Variables de entorno

Las más importantes son:

- $HOME → siempre haremos referencia al home del usuario.
- PATH → muy importante porque es la ruta de la carpeta donde están los comandos que se están ejecutando.
    - echo $PATH → nos muestra la ruta de la carpeta donde se buscan los comandos

La shell

Uso de comas y contrabarra

- La contrabarra sirve para escapar (quitar la función especial) un carácter especial. Ej: ***, \(, \)**
- El texto siempre irá entre comillas simples o dobles.
- Un texto entre comas simples ignorará **todos** los caracteres especiales.
- Un texto entre comas dobles ignorará todos los caracteres especiales excepto **$, ` y **
- La sintaxis **$(orden)** u **`orden`** permiten sustituir una orden por su salida.

echo “Hoy es `date`”
echo “Hoy es $(date)”

La shell

Uso de comas y contrabarra. Ejemplos

minombre=marina

echo mi nombre es $minombre

echo “mi nombre es $minombre”

echo 'mi nombre es $minombre'

echo mi nombre es $minombre y esto es un *

echo 'mi nombre es $minombre y esto es un *'

echo “mi nombre es $minombre y esto es un *”

Cread un archivo con 6 líneas: prueba

echo “el numero de líneas es `wc -l prueba`”

echo “el numero de líneas es `wc -l prueba | cut -d ' ' -f 1`”

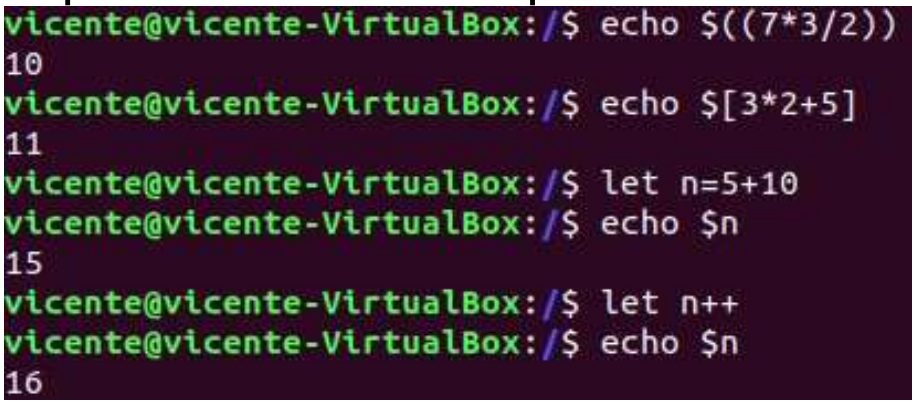
numlineas=`wc -l prueba | cut -d ' ' -f 1`

Echo “el número de líneas es $numlineas”

La shell

Operaciones aritméticas en la shell

- Es posible realizar operaciones aritméticas en la shell empleando los operadores + (suma) , - (resta), * (multiplicación), / (división), **(exponenciación), % (módulo), ++(sumo 1 a la variable), -- (resto 1 a la variable), +=(opero y asigno en un único comando), etc.
- Podemos emplear 3 opciones para sustituir una expresión aritmética por su resultado:
  $((expresion))
  $[expresion]
  let var=expresion



La shell

Operaciones aritméticas en la shell

- La división es entera: echo $((7*3/2)) = 10
- El módulo es el resto de la división $((7*3%2)) = 1
- Con el comando **let** creamos una variable let n=5+10, crea la variable n=15, si luego hacemos echo $n, nos saldrá 15.

- El comando let, existe en bash, pero no en shell. Porque pasa esto con este y otros comandos, siempre ejecutaremos los scripts en bash.

Shell scripting

- El script o programa shell es un fichero ejecutable que contiene órdenes internas y externas ejecutadas línea a línea.
- Los elementos del programa serán:
  - Variables
  - Comandos
  - Funciones
  - Comentarios
  - Estructuras de control de flujo y de repetición.
- El *shell scripting* permite utilizar las capacidades de la shell para automatizar multitud de tareas que, de otra forma, requerirían múltiples comandos introducidos de forma manual.
- El lenguaje scripting es un lenguaje interpretado por lo que no requiere de compilación para ejecutarse.
- Los programas compilados generalmente serán más rápidos que los interpretados siempre que estén ya compilados, si hay que compilarlos, es posible que sea más rápido el script.

Shell scripting

- El script se tiene que ejecutar desde la misma carpeta donde está el fichero, si no tendremos que poner la ruta absoluta.
- El script se sigue ejecutando aunque haya un error.
- Si ponemos un exit en el script, éste se saldrá de la ejecución aunque haya más comandos detrás. La orden exit permite finalizar un script especificando el código de salida. Si no hacemos uso de exit, el resultado de un script será el de la última orden que se ejecutó en el script.
- **$?** muestra el valor del resultado de lo último que he ejecutado, indicando si ha sido correcto o no. Por convenio, el valor 0 indicará que ha finalizado correctamente y un valor distinto a 0 que no ha finalizado correctamente.
  - Si OK $? = 0
  - Si NOK $? != 0
  - Probad con los comados ls y sl para ver los resultados.

Primer script

- Podemos crear scripts en cualquier editor de texto: gedit, emacs, nano, etc.
  - **nano fichero.sh** si el fichero no está, lo crea.
- Los scripts comienzan por #! (número mágico) , seguido del programa que interpreta el script. Es decir será:
  - **#! /bin/bash**
- Ejemplo: realiza tu primer script y ejecútalo

```
#! /bin/bash
echo “Hola mundo”
```

- Para ejecutar un script, primero hay que darle permisos de ejecución y luego ejecutarlo.

```
chmod 777
./fichero.sh
```

Primer script

1. Además hay que dar permisos de ejecución al script
2. Una vez convertido en ejecutable, se lanza como un comando:

chmod +x miscript.sh

./miscript.sh

- Con ./ indicamos que se encuentra y se ejecuta en la ubicación actual. Es necesario hacerlo si la ubicación del script no está incluida en PATH
- Como normalmente no estará, ejecutaremos siempre con ./

Comando read

- Permite que el usuario pueda introducir variables por teclado. Es decir, lee de la pantalla, cuando queremos preguntarle algo al usuario.
  - Ejemplo:

```
#!/bin/bash
echo “Hola, ¿como te llamas?
read nombre
echo “tu has escrito $nombre”
```

- Cuando el usuario escriba y le de a enter, se guardará el valor de la variable nombre. Como en este caso vamos a “escribir” la variable no usamos $.
- Cambiamos permisos y ejecutamos.
- ¿que pasa si ahora hago echo $nombre?

17

Comando read

- No aparece el valor de la variable porque esa variable está dentro del script y en cuanto se ejecuta éste, desaparece. En cambio si que aparece al ejecutar el script.

- Si la variable se crea en la consola, entonces si que se verá cuando se ejecute el echo $nombre. Pero esta variable también desaparecerá cuando se cierre la consola.

- Los parámetros que admite son:

  -s no hace eco en pantalla de la respuesta escogida (p.e. contraseñas)

  -nN: el número máximo de caracteres que se pueden introducir será de N

  -p “frase”: permite incluir una frase aclaratoria (para poner la pregunta “¿cual es tu nombre?” nombre (variable) en la misma linea)

  -tT: acepta una entrada por un tiempo máximo de T segundos

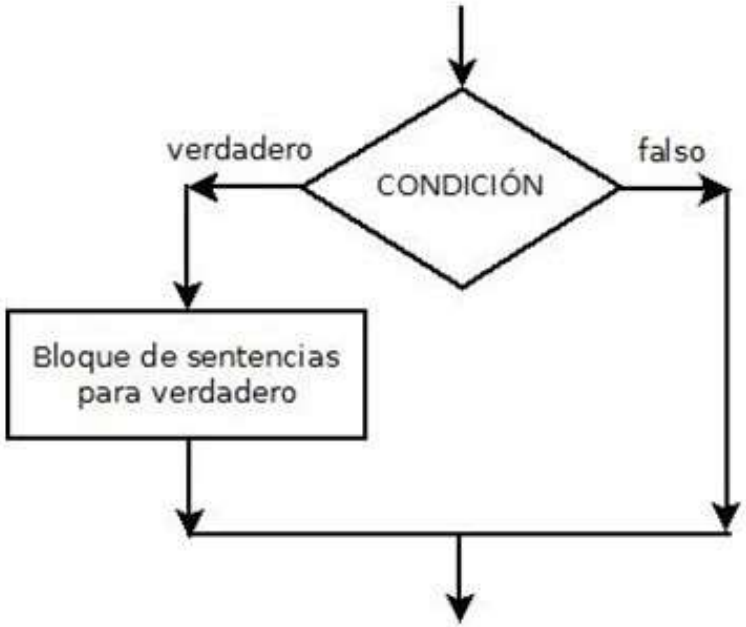
Estructuras de control condicional: if

if [expresión]

then

acciones

fi



If abre la estructura y **fi** la cierra

[condicion] → verdadero → ejecuto 1

→ falso → ejecuto 2

If evalúa la salida de una orden por lo que podemos usar:

test expresión → Esta es equivalente a la de abajo, por lo que usaremos los []

[expresión]

Estructuras de control condicional: if

Cosas que se deben tener en cuenta:

- Hay que dejar un **espacio en blanco** entre los corchetes.
- Podemos hacer condiciones usando variables, **haciendo uso del $**, puesto que queremos comparar su contenido.
- Es recomendable tabular las acciones para que quede más limpio el código.
- Siempre debe terminar la estructura con **"fi"**

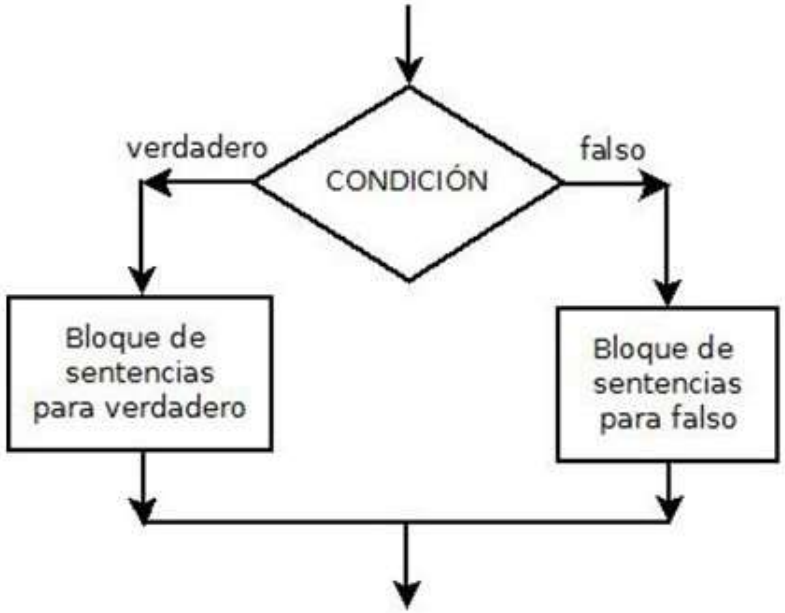
Ejemplos:
1. Haz un script que te pregunte tu nombre y si es correcto te devuelva un mensaje de bienvenida.
2. Haz un script que te pregunte por una operación aritmética y si es correcta, te devuelva un mensaje dándote la enhorabuena.

Estructuras de control condicional: If/Elif/Else

Podemos crear condiciones con alternativas, donde si no se cumple una condición se realiza otra lista de acciones:

if [condición]
then
    acciones
else
    acciones
fi

- If/Else



- Las secciones elif y else son opcionales

Estructuras de control condicional: If/Elif/Else

Se pueden anidar muchas condiciones diferentes con el elemento "**elif**":

if *[condición]*
then
    acciones

elif *[condición]*
then
    acciones

elif *[condición]*
then
    Acciones

else
    acciones
fi

Ejercicios:
1. realiza un script que te pida tu nombre. Si tu nombre es correcto, escribe un mensaje de bienvenida y crea un fichero.txt
Si no es así, mensaje de no eres bienvenido, autodestrucción y borra el fichero.

2. modifica el script anterior para que además de a tu nombre, salude a Pepe, si no, mensaje de no eres bienvenido, autodestrucción y borra fichero.

Comparadores de cadenas de texto/variables

- Una expresión puede ser una comparación de strings o cadenas de texto, números, operadores de fichero y operadores lógicos.
- Comparadores de strings o cadenas de texto:
  - Si usamos **el propio string**, el resultado será verdadero si la cadena no está vacía

```
A="Hola"
if [ A ]
then
        echo "Holaaa"
fi
```

= verdadero si es igual

!= verdadero si son diferentes

-n verdadero si la variable no está vacía

-z verdadero si la variable está vacía

Comparadores aritméticos

Comparación numérica (entre números):

- **-eq:** igual a
- **-ne**: no igual a
- **-gt**: mayor que
- **-ge**: mayor o igual que
- **-lt**: menor que
- **-le**: menor igual que

Comparadores de ficheros

- Operadores de ficheros:

  - **-e** verdadero si el fichero/directorio existe
  - **-d** verdadero si el path dado es un directorio
  - **-f** verdadero si el path dado es un archivo
  - **-r** verdadero si el fichero tiene permiso de lectura
  - **-w** verdadero si el fichero tiene permiso de escritura
  - **-x**verdadero si el fichero tiene permiso de ejecución
  - **-s** verdadero si el fichero tiene un tamaño mayor de 0 (no esta vacío)
  - **-O** verdadero si eres el propietario
  - **-G** verdadero si perteneces al grupo propietario

  - Todos estos comparadores se usan contra una variable.
  - Una variable puede ser un fichero o un directorio.

```
fichero="copia.txt"
If [ -e $fichero ]
then
        Echo "$fichero existe, pero no sé si es un fichero o un directorio"
fi
```

Si no se cumple la condición no hacemos nada porque no hay else.

Expresiones lógicas

- Operadores lógicos:

!, operador **NOT**, niega lo siguiente que le sigue
-a, &&, operador **AND** (sólo es verdadero si las 2 opciones son verdaderas)
-o, || operador **OR** (es verdadero si una de las 2 opciones es verdadera)

Ejemplos

```
#!/bin/bash
echo "Introduzca un número entre 1 y 10:"
read num
If [ "$num" -gt 1 -a "$num" -lt 10 ]; then
echo "el numero es correcto"
fi
```

```
if [ "$num" -gt 1 ] && [ "$num" -lt 10 ]; then
```

Anidar expresiones

- Si queremos comparar varios elementos en un if usando comparadores lógicos y necesitamos agruparlos con paréntesis tenemos 2 opciones:
  - Usando [] es necesario escapar los paréntesis. Ejemplo
    A=“alumno”
    edad=20
    if [\($A=“alumno” && “$edad” –ge 18 \) || \($A = “exalumno” \)]

  - Usando [[]] no es necesario escapar los paréntesis. Ejemplo
    A=“alumno”
    edad=20
    if [[($A=“alumno” && “$edad” –ge 18) || ($A = “exalumno”)]]

- Si sabemos que la variable es una cadena de texto la pondremos con dobles comillas para evitar errores si hay espacios, *, u otros caracteres especiales.