

UD3 – LINUX – ADMINISTRACIÓ I CONFIGURACIÓ-IV

1º DAW - CFGS

Prof. Manuel Enguidanos
menguidanos@fpmislata.com

3.9. VARIABLES



3.9. Variables

Una variable es un lugar en la memoria donde el sistema o un usuario guardan cierta información que después será necesario recuperar.

Las variables pueden ser locales, de *shell* o de entorno. Además de estas variables están las propias del usuario que este puede utilizar en sus *scripts*, como se verá en el próximo apartado.

El sistema utiliza las variables para almacenar valores relativos a su configuración o necesarios para que este funcione de forma correcta.

Las variables de entorno son utilizadas por los procesos ejecutados en un servidor y sirven para los clientes que accedan a este. Las variables locales y las de *shell* solo afectan a la sesión activa en cada equipo.

3.9. Variables

Variables de shell

Uso de variables:

- control del entorno (*environment control*)
- programación shell

Dos tipos

- variables locales: visibles sólo desde el shell actual
- variables globales o de entorno: visibles en todos los shells

El comando **set** permite ver las variables definidas en nuestra shell

- El nombre de las variables debe:
 - empezar por una letra o `_`
 - seguida por cero o mas letras, números o `_` (sin espacios en blanco)

3.9. Variables

Variables de shell

Uso de las variables

- Asignar un valor: *nombre_variable=valor*

```
$ un_numero=15  
$ nombre="Pepe Pota"
```

- Acceder a las variables: *\${nombre_variable}* o *\$nombre_variable*

```
$ echo $nombre  
Pepe Pota
```

- Número de caracteres de una variable

```
$ echo ${#un_numero}  
2
```

- Eliminar una variable: *unset nombre_variable*

```
$ unset nombre  
$ echo ${nombre}mo  
mo
```

- Variables de solo lectura: *readonly nombre_variable*

```
$ readonly nombre  
$ unset nombre  
bash: unset: nombre: cannot unset: readonly variable
```

3.9. Variables

Para ver además otras variables, como las variables de *shell*, se puede utilizar el comando `set`.

Tabla 3.9. Ejemplos de variables de *shell* con su nombre y su contenido

Variable	Contenido
HISTFILE	Nombre del fichero que guardará el historial de los comandos.
HISTFILESIZE	Número de comandos que se pueden guardar en el fichero <code>.bash_history</code> .
HISTSIZE	Número de comandos que están en la caché.
HOSTNAME	Nombre del equipo.
PS1	<i>Prompt</i> primario del sistema.
PS2	<i>Prompt</i> secundario del sistema.

Las variables se cargan en los ficheros de inicio y fin de sesión del sistema y de cada usuario.

El usuario puede crear y utilizar sus propias variables:

`variable=valor`

Ejemplo: `num=7; echo $num`

`variable=`comando``

Ejemplo: `hoy=`date +%D`; echo $hoy`

`variable=$(comando)`

Ejemplo: `hora=$(date +%H:%M:%S); echo $hora`

3.9. Variables

Variables de entorno

Cada shell se ejecuta en un *entorno* (*environment*)

- el entorno de ejecución especifica aspectos del funcionamiento del shell
- esto se consigue a través de la definición de variables de entorno (o variables globales)
- algunas variables son:

Nombre	Propósito
HOME	directorio base del usuario
SHELL	shell por defecto
USERNAME	el nombre de usuario
PWD	el directorio actual
PATH	el <i>path</i> para los ejecutables
MANPATH	el <i>path</i> para las páginas de manual
PS1/PS2	<i>prompts</i> primario y secundario
LANG	aspectos de localización geográfica e idioma
LC_*	aspectos particulares de loc. geográfica e idioma

3.9. Variables

Variables de entorno

Cada shell se ejecuta en un *entorno* (*environment*)

- Para definir una nueva variable de entorno: `export`

```
$ nombre="Pepe Pota"      # Define una variable de shell
$ echo $nombre            # Usa la variable en el shell
Pepe Pota                 # padre
$ export nombre           # Exporta la variable
$ bash                    # Inicia un nuevo shell
$ echo Mi nombre es $nombre # Intenta usar la variable
Mi nombre es Pepe Pota    # del shell padre
$
```

- La variable exportada (variable de entorno) es visible en el shell hijo
 - el shell hijo crea una copia local de la variable y la usa
 - las modificaciones de esa copia no afectan al shell padre
- Para ver las variables de entorno definidas usar `env` o `printenv`

3.9. Variables

Algunas de las variables más usadas son las detalladas en la Tabla 3.8.

Tabla 3.8. Ejemplos de variables de entorno con su nombre y su contenido

Variable	Contenido
HOME	Ruta hacia el directorio personal del usuario.
SHELL	Ruta al intérprete de órdenes.
PWD	Ruta del directorio de trabajo actual.
LOGNAME	Nombre del usuario que está conectado en ese momento.
LANG	Configuración del idioma y localización del sistema.
Variable	Contenido
USER	Nombre del usuario que inició la sesión.
TERM	Tipo de emulación de terminal (puede ser de tipo Linux o de tipo color).
OLDPWD	Ruta del directorio anterior al que hemos estado (si se usa el comando <code>cd</code> con la opción siguiente, <code>cd -</code> , se irá al directorio indicado en la variable).
PATH	Rutas (separadas por el carácter <code>:</code>) donde el intérprete de órdenes buscará los comandos que se vayan a ejecutar, si no se especifica una ruta hacia él.

3.10. SCRIPTS DE LINUX



3.10. Introducción a los *scripts* de Linux

Para automatizar tareas del sistema operativo y no tener que repetirlas se utilizan los *scripts* (guiones), que son ficheros que contienen una serie de comandos que se ejecutarán cada vez que se ejecute el *script*. Se suelen utilizar para automatizar tareas de administración del sistema, aunque se pueden utilizar para cualquier utilidad.

Existen una serie de *scripts* del sistema que se ejecutan cada vez que el usuario inicia o cierra la sesión: `.bash_profile`, `.bash_login` y `.profile` en el directorio personal del usuario. Al cerrar la sesión el *script* que se ejecuta es `.bash_logout`. El sistema utiliza estos *scripts* para configurar ciertos parámetros de la cuenta del usuario y el mismo usuario puede modificarlos. En estos ficheros se añaden los alias de los usuarios.

3.10. Introducción a los *scripts* de Linux

Por defecto, los *scripts* tienen la extensión `.sh`, aunque esta extensión no es obligatoria, sino que es sobre todo para que el usuario reconozca estos ficheros y poder especificar por ejemplo que se abran con una determinada aplicación.

Lo que sí es necesario es que para que se puedan ejecutar tengan permiso de ejecución (Apartado 3.4). Además, habrá que indicar al sistema la ruta para llegar a estos archivos (ya sea absoluta o relativa), o bien incluirlos en un directorio que esté dentro de la ruta de la variable **PATH** (Apartado 3.9).

Dentro de cada *script* habrá que incluir los comandos que se desea ejecutar cada vez que se invoque al *script*. La primera línea del *script* deberá ser la que indique al sistema la *shell* con la que se quiere ejecutar; en el caso de la *shell* **bash** será:

```
#!/bin/bash
```


3.10. Introducción a los *scripts* de Linux

Una vez creado el fichero, debemos darle permisos de ejecución, mediante el comando:

chmod a+x primero.sh


Posteriormente para ejecutarlo debemos llamarlo como *./primero.sh* (el punto barra es para indicarle que lo busque en el directorio actual, ya que dicho directorio no estará seguramente incluido en el PATH del sistema).

Escribimos script



```
#!/bin/bash

echo '¡Hola, mundo!'
```



```
team@itsfoss:~/scripts$ cat > hola.sh
#!/bin/bash

echo '¡Hola, mundo!'
```

```
team@itsfoss:~/scripts$ chmod u+x hola.sh
```

Damos permiso de ejecución

```
team@itsfoss:~/scripts$ ./hola.sh
¡Hola, mundo!
```

Ejecutamos

■ 3.10. Introducción a los *scripts* de Linux

■■■ Comentarios

Una línea que empiece por el carácter **#** será considerada como un comentario. Por ejemplo:

```
#!/bin/bash  
#primer script
```



Realitzar Activitats Resoltes





Actividad resuelta 3.25

Crea un *script* de Linux que borre la pantalla y muestre la fecha. Añade un comentario que indique lo que hace el *script*.

Solución

```
#!/bin/bash
#script que borra la pantalla, muestra la fecha y te saluda
clear
date
echo "Hola, $LOGNAME"
```


3.10. Introducción a los *scripts* de Linux

Variables específicas de los *scripts*

Tabla 3.10. Variables utilizadas dentro de los *scripts* con la función que realizan

Variable	Función
\$0	Nombre del <i>script</i> .
\$1, \$2, \$3...	Parámetros posicionales que se introducen en el <i>script</i> desde la línea de comandos después del nombre.
Variable	Función
\$#	Número de parámetros posicionales que tiene el <i>script</i> .
\$*	Variable que recoge el valor de todos los parámetros posicionales.
\$?	Valor devuelto por el último comando ejecutado. No solo se utiliza en <i>scripts</i> , sino en cualquier comando que se ejecute en el sistema.
\$\$	PID (<i>Process ID</i> , identificador de proceso) del <i>script</i> . Al ejecutarse, como cualquier otro programa, crea un proceso que tiene un número que lo identifica en el sistema.

- `$0` representa el nombre del script
- `$1` – `$9` los primeros nueve argumentos que se pasan a un script en Bash
- `$#` el número de argumentos que se pasan a un script
- `$@` todos los argumentos que se han pasado al script
- `$?` la salida del último proceso que se ha ejecutado
- `$$` el ID del proceso del script

3.10. Introducción a los *scripts* de Linux

Comandos

echo

Muestra una línea de texto.

`echo [opciones] [cadenas]...`

`-n`

No salta de línea al final.

`-e`

Interpreta los caracteres de escape, como `\c` (no salta de línea), `\n` (inserta un salto de línea) y `\t` (tabulador), entre otros.

```
$ echo Esto es un ejemplo
Esto es un ejemplo
```

```
$ echo "Esto es un ejemplo"
Esto es un ejemplo
```

3.10. Introducción a los *scripts* de Linux

Comandos

`read`

Lee una línea de la entrada estándar. El valor leído se puede almacenar en una variable para utilizarlo después.

`read [opciones] [nombre...]`

`-n ncars`

Lee solo los caracteres indicados en ***ncars***.

`-p prompt`

Muestra la cadena ***prompt*** por pantalla.

`-s`

No muestra la salida que se lea.

`-t tiempo`

Espera un tiempo y expira al cabo del ***tiempo*** indicado.

```
#!/bin/bash
```

```
read -p "Usuario: " usuario
```

```
read -sp "Bienvenido, $usuario, introduce tu contraseña: " password
```

```
echo -e "\nEl usuario es $usuario y la contraseña es $password"
```

```
[root@primary ~]# sh ./read.sh
```

```
Usuario: gpsos
```

```
Bienvenido, gpsos, introduce tu contraseña:
```

```
El usuario es gpsos y la contraseña es Web_gpsos
```

3.10. Introducción a los *scripts* de Linux

Estructuras condicionales y de control

if/else

Ejecuta un comando o varios dependiendo de una condición.

```
if <condición>
then
    <comando/s>
[elif <condición>
then
    <comando/s>]
[else
    <comando/s>]
fi
```

```
#!/bin/sh

a=99
b=45

if [ $a -lt $b ]
then
    echo "a is less than b"
else
    echo "a is greater than b"
fi
```

Resultado:

a is greater than b

3.10. Introducción a los *scripts* de Linux

Estructuras condicionales y de control

for

Ejecuta uno o varios comandos el número de veces indicado en **valores**.

```
for $variable in valores
do
    <comando/s>
done
```

```
#!/bin/bash
for i in 1 2 3 4 5
do
    echo "Hello $i"
done
```

La ejecución del archivo bash genera el siguiente texto:

```
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
```

3.10. Introducción a los *scripts* de Linux

Estructuras condicionales y de control

while

Repite un código mientras no se cumpla una condición.

```
while <condición>
do
    <comando/s>
done
```

```
n=1
while [ $n -le 5 ]
do
    echo "Running $n time"
    (( n++ ))
done
```

```
gonka@gonka-pcmerk:~$ bash ejemplo1-while.sh
Running 1 time
Running 2 time
Running 3 time
Running 4 time
Running 5 time
```

3.10. Introducción a los *scripts* de Linux

Estructuras condicionales y de control

until

Repite un código hasta que se cumpla una condición.

```
until <condición>
do
    <comando/s>
done
```

```
#!/bin/bash
# Bucle básico Until
contador=1
until [ $contador -gt 10 ]
do
    echo "El valor de la variable $contador es: " $contador
    ((contador++))
done
echo "Bucle finalizado. Enhorabuena!"
```

```
[raul@localhost /home/raul]$ ./script_until.sh
El valor de la variable $contador es: 1
El valor de la variable $contador es: 2
El valor de la variable $contador es: 3
El valor de la variable $contador es: 4
El valor de la variable $contador es: 5
El valor de la variable $contador es: 6
El valor de la variable $contador es: 7
El valor de la variable $contador es: 8
El valor de la variable $contador es: 9
El valor de la variable $contador es: 10
Bucle finalizado. Enhorabuena!
```

3.10. Introducción a los *scripts* de Linux

Estructuras condicionales y de control

case

Ejecuta un código u otro dependiendo de un valor.

```
case $variable in
    expr1) comando/s;;
    expr2) comando/s;;
    ...
    *) comando/s;;
esac
```

Algunas reglas para tomar en cuenta

Paréntesis de cierre) después de cada caso (condición).

Doble punto y coma ;; delimita la lista de comandos que serán ejecutados cuando se cumpla el caso (condición).

Finalmente cerrar la sentencia case con esac .

```
#!/bin/bash

echo "¿Cuál es su idioma?"

select language in Español Català Galego Euskera Other
do
    case $language in
        "Español"|"Català")
            echo "Hola, $USER."
            break
            ;;
        "Galego")
            echo "Ola, $USER."
            break
            ;;
        "Euskera")
            echo "Kaixo, $USER."
            break
            ;;
        *)
            echo "Hello, $USER."
            break
            ;;
    esac
done
echo "Fuera de select, el idioma sigue siendo $language."
```

```
¿Cuál es su idioma?
1) Español
2) Català
3) Galego
4) Euskera
5) Other
?: 1
Hola, fernando.
Fuera de select, el idioma sigue siendo Español.
```


3.10. Introducción a los *scripts* de Linux

Estructuras condicionales y de control

`select`

El valor de la variable puede ser uno de los que estén en la lista de valores. Una vez seleccionado se ejecuta el comando o los comandos que se indiquen.

```
select $variable in valores
do
    <comando/s>
done
```

```
#!/bin/bash

echo "¿Cuál es su idioma?"

select language in Español Català Galego Euskera Other
do
    case $language in
        "Español"|"Català")
            echo "Hola, $USER."
            break
            ;;
        "Galego")
            echo "Ola, $USER."
            break
            ;;
        "Euskera")
            echo "Kaixo, $USER."
            break
            ;;
        *)
            echo "Hello, $USER."
            break
            ;;
    esac
done
echo "Fuera de select, el idioma sigue siendo $language."
```

```
¿Cuál es su idioma?
1) Español
2) Català
3) Galego
4) Euskera
5) Other
#? 1
Hola, fernando.
Fuera de select, el idioma sigue siendo Español.
```

3.10. Introducción a los *scripts* de Linux

Condiciones

Para evaluar el valor que devuelven las condiciones, se puede añadir la expresión dentro de unos corchetes, `[expresión]`, que es una forma abreviada del comando `test`. Las condiciones se pueden referir a ciertas propiedades de un fichero (Tabla 3.11), pueden comparar valores de cadenas de caracteres (Tabla 3.12) o bien pueden comparar valores numéricos (Tabla 3.13).

Tabla 3.11. Principales condiciones que se pueden utilizar relacionadas con los ficheros

Condición	True o verdadero si:
<code>-f valor</code>	valor es un fichero que existe y es un fichero regular.
<code>-d valor</code>	valor existe y es un directorio.
<code>-e valor</code>	valor existe.
<code>-w valor</code>	valor tiene permiso de escritura.
<code>-r valor</code>	valor tiene permiso de lectura.
<code>-x valor</code>	valor tiene permiso de ejecución.
<code>-L valor</code>	valor es un enlace simbólico.
<code>valor1 -nt valor2</code>	valor1 es un fichero más reciente que valor2 .
<code>valor1 -ot valor2</code>	valor1 es un fichero más antiguo que valor2 .

Condicionales con archivos

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

```
#!/bin/bash
```

```
FICHERO=/etc/passwd
```

```
if [ -f $FICHERO ]
```

```
then
```

```
    echo "El fichero $FICHERO existe"
```

```
else
```

```
    echo "El fichero $FICHERO no existe"
```

```
fi
```

3.10. Introducción a los *scripts* de Linux

Condiciones

Tabla 3.12. Condiciones que se pueden utilizar para comprobar los valores de las cadenas de caracteres

Condición	True o verdadero si:
<code>valor1 = valor2</code>	<i>valor1</i> es una cadena igual que <i>valor2</i> .
<code>valor1 != valor2</code>	<i>valor1</i> es una cadena distinta a <i>valor2</i> .
<code>-n valor</code>	<i>valor</i> es una cadena de longitud mayor que 0.
<code>-z valor</code>	<i>valor</i> es una cadena vacía o de longitud igual a 0.

Tabla 3.13. Condiciones que se pueden utilizar para comparar valores numéricos

Condición	True o verdadero si:
<code>num1 -eq num2</code>	<i>num1</i> es un número igual a <i>num2</i> .
<code>num1 -ne num2</code>	<i>num1</i> es un número distinto a <i>num2</i> .
Condición	True o verdadero si:
<code>num1 -lt num2</code>	<i>num1</i> es un número menor que <i>num2</i> .
<code>num1 -gt num2</code>	<i>num1</i> es un número mayor que <i>num2</i> .
<code>num1 -le num2</code>	<i>num1</i> es un número menor o igual a <i>num2</i> .
<code>num1 -ge num2</code>	<i>num1</i> es un número mayor o igual que <i>num2</i> .

Se pueden utilizar también las operaciones lógicas `-a` o `&&` para AND, `-o` o `||` para OR y `!` para indicar NOT.

3.10. Introducción a los *scripts* de Linux

Funciones

Dentro de los *scripts* se pueden declarar funciones propias para poder llamarlas desde el código. La forma de declarar una función es la siguiente:

```
function nombre {  
    <comando/s>  
}
```

```
#!/bin/bash  
#Author: Diego Bastidas  
# Definición y uso de funciones  
  
hola_mundo () {  
    echo "Hola mundo"  
}  
  
parametros () {  
    echo "Hola soy $1 y suscribete a $2"  
}  
  
read -p "Ingrese su nombre: " nombre  
read -p "Ingrese el nombre de su canal: " canal  
  
hola_mundo  
parametros $nombre $canal
```

```
dfbastidas@dfbastidas:~/curso_scripting$ ./funciones.sh  
Ingrese su nombre: Diego  
Ingrese el nombre de su canal: Bastidas  
Hola mundo  
Hola soy Diego y suscribete a Bastidas
```



Realitzar Activitats Resoltes





Actividad resuelta 3.26

Crea un **script** llamado **total.sh** que reciba una serie de parámetros, te muestre el número total de parámetros y te muestre cada uno por pantalla, uno por **línea**.

Solución

```
sudo nano total.sh
```

```
#!/bin/bash
echo "Número de parámetros: $#"
```

```
while [ $# -ne 0 ]
do
    echo $1
    shift
done
```

```
Ctrl + O
Ctrl + S
chmod a+x total.sh
```

Para ejecutarlo, puedes probar lo siguiente:

```
./total.sh a b c
```

Salida por pantalla:

Número de parámetros: 3

a

b

c



Actividad resuelta 3.27

Crea un *script* llamado **tipo.sh** que reciba un parámetro y te indique si es el nombre de un fichero, el nombre de un directorio o no es ninguno.

Solución

```
sudo nano tipo.sh
#!/bin/bash
if [ $# -eq 0 ]; then
    echo "No hay parámetros"
else
    if [ -f $1 ]; then
        echo "$1 es un fichero"
    elif [ -d $1 ]; then
        echo "$1 es un directorio"
    else
        echo "$1 no es fichero ni directorio"
    fi
fi
Ctrl + O
Ctrl + S
chmod a+x tipo.sh
```

Para ejecutarlo, puedes probar lo siguiente:

<code>./tipo.sh .</code>	Salida: <code>.</code> es un directorio
<code>./tipo.sh tipo.sh</code>	Salida: <code>tipos.sh</code> es un fichero
<code>./tipo.sh</code>	Salida: No hay parámetros



Actividad resuelta 3.28

Crea el mismo *script* de la actividad anterior, llámalo **pregunta.sh**, pero en este caso te debe preguntar que introduzcas por teclado el nombre y te confirme si es fichero o directorio.

Solución

```
sudo nano pregunta.sh
```

```
#!/bin/bash
echo -e "Escribe el nombre de un fichero: \c"
read pregunta
if [ -f $pregunta ]; then
    echo "$pregunta es un fichero"
elif [ -d $pregunta ]; then
    echo "$pregunta es un directorio"
else
    echo "$pregunta no es fichero ni directorio"
fi
Ctrl + O
Ctrl + S
chmod a+x pregunta.sh
```

Para ejecutarlo, puedes probar lo siguiente:

```
./pregunta.sh
```

Salida por pantalla e introducción por teclado:

```
Escribe el nombre de un fichero: pregunta.sh
pregunta.sh es un fichero
```




Actividad resuelta 3.29

Crea un *script* llamado **saludo.sh** que borre la pantalla y te muestre la vaca del sistema, saludándote y diciéndote la hora. Para ello deberás instalar el programa **cowsay**.

Solución

```
sudo apt install cowsay  
sudo nano saludo.sh
```

```
#!/bin/bash  
#script que borra la pantalla, muestra la fecha y te saluda  
clear  
echo "Hola, $LOGNAME, son las `date +%R`" | cowsay
```

```
Ctrl + O  
Ctrl + S  
chmod a+x saludo.sh  
./saludo.sh
```

También es válido `$(date +%R)`

```
usuario@Equipo: ~  
$ ./saludo.sh  
< Hola, usuario, son las 01:02 >  
-----  
      ^__^  
      (oo)\_____  
      (__)\\       )\/\  
      ||----w |  
      ||     ||  
usuario@Equipo:~$  
usuario@Equipo:~$
```

Figura 3.45. Ejecución del script *saludo.sh* por pantalla.

Para profundizar más en los *scripts* de Linux se recomienda realizar la Actividad de ampliación 3.5.



Realitzar Practica 4



UD3 – LINUX – ADMINISTRACIÓ I CONFIGURACIÓ-IV

1º DAW - CFGS

Prof. Manuel Enguidanos
menguidanos@fpmislata.com