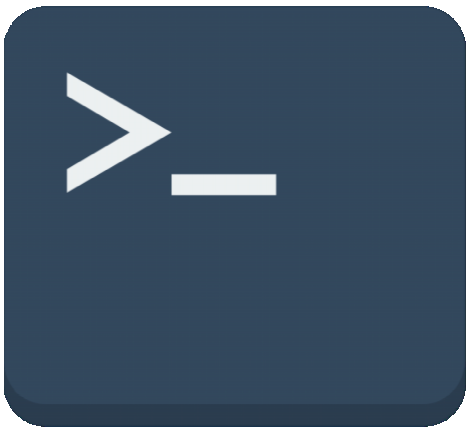
ADMINISTRACIÓN DE SISTEMAS OPERATIVOS

PROGRAMACIÓN SHELL SCRIPT



Leonardo Marco Verdú

ÍNDICE DE CONTENIDOS

1. PROGRAMACIÓN BÁSICA DE SHELL SCRIPT

1.1. BASH SCRIPT

SCRIPTS EN BASH
○ Un script en Bash es un fichero con una lista de comandos que se ejecutan secuencialmente. ○ El script debe empezar con la línea: **#!/bin/bash** - Al ejecutar el script como un fichero ejecutable (**./script.sh**) se llamará al intérprete indicado en dicha línea para ejecutar el script. - Si se ejecuta el script llamándolo como parámetro de un intérprete se ejecutará con dicho intérprete: **bash script.sh** (se ejecuta con bash) ○ La mayoría de distros tienen el enlace **sh** a un intérprete (no siempre el bash).

#texto comentario	Texto de comentario que no se interpreta Puede ser una línea entera o ir a mitad de una línea comando1 #El comando1 se ejecuta pero esta línea se obvia

GUÍA DE PROGRAMACIÓN Y ESTILO EN BASH
○ Utilizar comentarios: de fichero, en líneas, en secciones, en funciones, etc. ○ Utilizar sangría según el nivel de profundidad de las estructuras ○ Utilizar comandos internos y expansiones cuando sea posible - Cada comando externo crea un nuevo proceso (en bucles puede afectar al rendimiento) ○ https://lug.fh-swf.de/vim/vim-bash/StyleGuideShell.en.pdf

1.2. VARIABLES

ASIGNACIÓN DE VARIABLES	
nom_var=valor	Sin espacios entre el símbolo = El nom_var sin $ (si valor es variable sí llevaría $) Caracteres nombre: a-zA-Z0-9_ (sólo puede empezar _a-zA-Z) **a=$b**
nom_var=valor comando	Llama al comando con la variable configurada para esa ejecución **LD_LIBRARY_PATH=/usr/lib/mozilla firefox**

USO DE VARIABLES	
$variable	Recupera el valor de la variable para utilizarlo
${variable}	Sintaxis más conveniente para evitar ambigüedades **a=1; b=2; ab=3;** **echo $ab** #Salida: 3 **echo ${a}b** #Salida: 1b
"$variable"	En ocasiones hay que colocar "" para tener en cuenta los valores con espacios **a="1 3"** **echo "$a"** #Salida: 1 3 **echo $a** #Salida: 1 3
'$variable'	Entre comillas simples no se resuelven las variables

	echo La variable '$PPID tiene el valor' "$PPID"
unset variable	Borra una variable **unset a**
export variable	Mete una variable en el entorno Será **visible** por el shell actual y sus hijos Por convención se usan nombres en **mayúsculas** Configurar variable **global permanente** al usuario hacer export en: ~/.bash_profile (ó ~/.profile) #Al hacer login ~./bashrc #Al abrir cada nueva instancia de bash Configurar variable global permanente a todos los usuarios: /etc/bashrc /etc/profile /etc/enviroment
env	Muestra la lista de variables de entorno del usuario actual

ENTRADA DE DATOS POR TECLADO	
read v1 v2 v3	Lee una línea de la entrada estándar y mete en cada variable una palabra. Si sobran palabras se mete todo lo restante en la última variable Si no se incluye variable **$REPLY** guarda la respuesta
read -p msg v1 v2 v3 **read -t s v1 v2 v3** **read -n l v1 v2 v3**	Escribe un mensaje antes de solicitar la entrada. Tempriza s segundos la espera Acepta un máximo de l caracteres
read lista_var < fich	En lugar de la entrada estándar utiliza el contenido del fichero Cuando encuentra un salto de línea termina de leer.
while read m; do… **done < fichero**	Lee línea a línea el fichero y lo mete en la variable m
echo "valor" | read v	No funciona debido que la tubería crea un shell diferente

VARIABLES POSICIONALES	
$0	Nombre del script con ruta incluida (con la que se ejecutó)
$1 **$2** **...** **${10}** **...**	Primer parámetro Segundo parámetro … Décimo parámetro ...
$#	Número de parámetros pasado al script
shift n	Mueve a la izquierda la lista completa de variables n posiciones Se **pierden** los n primeros valores y no pueden recuperarse
$*	Lista completa de parámetros separados por palabras. Elimina todos los espacios y cada **palabra** es un elemento de la lista **p1 p2 "p3a p3b" → |p1|p2|p3a|p3b|**
"$*"	Lista con un solo elemento con todos los parámetros con espacios

	p1 p2 "p3a p3b" → |p1 p2 p3a p3b|
$@	Ídem a $*
"$@"	Mantiene los **espacios** y tiene en cuenta los **parámetros compuestos** Suele ser la opción más adecuada para recorrer los parámetros **p1 p2 "p3a p3b" → |p1|p2|p3a p3b|**

CÓDIGOS DE SALIDA	
Al finalizar cada programa devuelve un número según el estado de terminación. 0: finalización sin problemas !=0: otro estado Ejemplo grep: 0 (ok y encontrado), 1 (ok y no encontrado), 2 (error al ejecutar)	
$?	valor de salida del último programa ejecutado.
exit n	Sale del script y devuelve el código de salida n.

VARIABLES DE ENTORNO	
env	Muestra las variables de entorno
$HOME	Directorio personal del usuario actual
$PWD	Directorio actual
$USER	Nombre del usuario actual
$SECONDS	Número de segundos desde que empezó el shell
$PPID	PID del proceso padre En caso de scripts coincide con el bash que lo ejecutó
$$	PID proceso actual
$PATH	Valor del path del usuario
$RANDOM	Número aleatorio entre 0 y 32767 Para aleatorios entre otro rango utilizar el operador módulo %
$IFS	Separadores utilizados por read, for, etc. Por defecto: espacio, tabulación y nueva línea Para ver el contenido: set | grep ^IFS=

1.3. MATRICES

MATRICES	
○ Son arrays unidimensionales sin tamaño fijo ○ El índice es numérico sin importar el orden o continuidad	
a[0]=valor	Asigna el valor a la posición 0 al array a
a=(valor1 valor2 valor3)	Crea un nuevo array desde la posición 0 con valor1...
a=([0]=val1 [4]=val2 [7]=val3)	Crea un nuevo array a en las posiciones indicadas
echo ${a[0]}	Accede a la posición 0 del array a
echo ${a[*]}	Obtiene una lista con todos los valores del array a
echo ${a[@]}	Obtiene una lista con todos los valores del array a
echo ${#a[*]}	Obtiene el número de elementos del array a
echo ${a[@]:n}	Obtiene desde la n-ésima posición hasta el final

echo ${a[@]:n:m}	Obtiene desde la n-ésima posición m posiciones

1.4. EXPRESIONES ARITMÉTICAS

COMANDO LET	
○ Realiza una operación aritmética de asignación. ○ No devuelve nada a la salida estándar.	
+, -, *, /, **, % **+=, -=, *=, /=, &=**	Suma, resta, producto, división, exp, módulo **a+=$b** #Equivalente a: a=a+$b
let c=$a+$b*3 **let "c = $a+$b * 3"**	Sin comillas no pueden usarse espacios Con comillas pueden usarse espacios
let c+=$a	c=c+a
let c=$a%$b	c es el módulo (resto división entera) entre a y b

EXPANSIÓN CON $((expr)) ó $[expr]	
○ Realizan la expresión **aritmética** y se **sustituye** el resultado por su posición ○ Los espacios dentro de la expresión no influyen	
$((expr)) **a=$(((i+10)*j))**	Realiza el cálculo y se sustituye por el resultado a=(i+10)*j
$[expr] **echo $[(i+10)*j]**	Realiza el cálculo y se sustituye por el resultado a=(i+10)*j

1.5. EXPRESIONES CONDICIONALES

○ Obtienen un valor lógico como resultado de una expresión de comparación Si al terminar el resultado es **VERDADERO** hacen **$?=0** Si al terminar el resultado es **FALSE** hacen **$?=1** ○ CADENAS VACÍAS: si alguna variable puede estar vacía es mejor usarla entre comillas **test $a = "$b"** #Si $b está vacía y no se ponen comillas dará error **[-f "$a"]** #Si $a está vacío y no se ponen las comillas se evaluará como cierto	
test expr	Evalúa expr **condicional** y actualiza $? según si es cierta o falsa Puede utilizar todos las opciones que se detallan a continuación
[expr]	Sinónimo de test Debe dejarse un **espacio** entre expr y los corchetes Debe utilizarse con las opciones que se detallan a continuación No confundir con $[expr] que se sustituye por el resultado
[[expr]]	Para evaluar algunas **condiciones** más complejas Puede utilizar la sintaxis típica de programación: &&, ||, ... Puede utilizar las expresiones que se detallan a continuación **[[-f f1 && -f f2]]**
((expr))	Evalúa expr **condicional** y devuelve el resultado en $? Pueden utilizarse los operadores **aritmético-lógicos** típicos (++,>, <, ==, >=, <=, !...) No funcionan las opciones de test (-a -d -eq...) ni cadenas No importan los espacios No confundir con $((expr)) que se sustituye por el resultado
true	Únicamente devuelve una salida exitosa $?=0

false	Únicamente devuelve una salida no exitosa $?=1

FICHEROS	
-a fichero	El fichero o directorio existe
-d fichero	El fichero existe y es un directorio
-f fichero	El fichero existe y es un fichero regular
-s fichero	El fichero existe y tiene un tamaño >0
-r fichero	El usuario tiene permiso de lectura sobre el fichero
-w fichero	El usuario tiene permiso de escritura sobre el fichero
-x fichero	El usuario tiene permiso de ejecución sobre el fichero
-h fichero	El fichero es un enlace duro
-L fichero	El fichero es un enlace simbólico
-O fichero	El usuario es el propietario del fichero
-G fichero	El usuario pertenece al grupo del fichero
-N fichero	El fichero se modificó desde su última lectura
f1 -nt f2	El fichero f1 es más nuevo que el f2
f1 -ot f2	El fichero f1 es más antiguo que el f2
f1 -ef f2	El fichero f1 y f2 son enlaces duros al mismo archivo

NÚMEROS	
n1 -eq n2	n1 es igual a n2
n1 -ne n2	n1 no es igual a n2
n1 -lt n2	n1 es menor a n2
n1 -gt n2	n1 es mayor a n2
n1 -ge n2	n1 es mayor o igual a n2
n1 -le n2	n1 es menor o igual a n2

CADENA DE CARACTERES	
c1 = c2	c1 es igual a c2 carácter a carácter. Obligatorios los **espacios** entre =
c1 != c2	n1 no es igual a n2
-n c1 **cadena**	la longitud de la cadena no es 0
-z cadena	la longitud de la cadena es 0
c1 > c2	c1 precede alfabéticamente a c2
c1 < c2	c1 antecede alfabéticamente a c2

COMBINACIÓN DE EXPRESIONES	
test ! expr **[! expr]**	Niega una expresión
[! expr1] && [! expr2]	Combinación de AND y negaciones

[! expr1] || [! expr2]	Combinación de OR y negaciones
((expr)) && ((expr))	Combinación de AND con (())

1.6. EXPANSIONES Y SUSTITUCIONES

EXPANSIÓN DE FICHEROS	
○ Al ejecutar un comando si se encuentra algún comodín se intenta resolver por los ficheros encontrados en ./ ó si se comenzó a escribir ruta por los de esa ruta ○ Si no se encuentran ficheros se deja el texto con el comodín incluido ○ Si se quiere omitir la expansión debe entrecomillarse o escaparse el comodín	
*****	Sustituye a cualquier cadena (incluida cadena vacía)
?	Sustituye a un carácter cualquier (tiene que ser 1)
[caracteres]	Sustituye a uno de los caracteres entre [] (tiene que ser 1)
[!caracteres]	Sustituye a un carácter que no esté entre [] (tiene que ser 1)

EXPANSIÓN DE COMANDOS	
`comando`	Sustitución de la salida del comando en su posición No anidable
$(comando)	Sustitución de la salida del comando en su posición Anidable: **lista=$(ls $(cat directorios.txt))**

EXPANSIÓN DE LLAVES	
echo a{b,c,d}e	Sustituye por: abe, ace y ade
mkdir a{1,2,3}	Sustituye por: a1, a2 y a3 (crea 3 directorios)
mkdir a{1..10}	Sustituye por de a1 a a10 (crea 10 directorios)

EXPANSIÓN DE CARACTERES ESPECIALES	
$'texto'	Se sustituye por el texto considerando los caracteres especiales \a, \b, \n, \t, \v, \\

EXPANSIÓN DE DIRECTORIOS	
~nombre_usu	Directorio personal del usuario usu (o del actual si no se indica)
~+	Directorio actual
~-	Directorio anterior

EXPANSIÓN DE VARIABLES	
${!prefijo*}	Obtiene una lista de todas las variables que empiezan por prefijo.
${#v1}	Obtiene el tamaño de los caracteres de la variable v1
${#m1[*]}	Obtiene el tamaño de una matriz

	matriz[${#matriz[*]}]="v1" #Añade v1 como elemento al final.
${v1:n}** **${v1: -4} **${v1:n:m}**	Obtiene v1 desde el n-ésimo carácter hasta el final n puede ser un número negativo comenzando desde el final Obtiene v1 desde el n-ésimo carácter hasta longitud m
${a1[@]:n}** **${a1[@]:n:m}	Obtiene del array a1 desde la n-ésima posición hasta el final Obtiene del array a1 desde la n-ésima posición m posiciones
${v1/patrón/cadena}** **${v1//patrón/cadena} **${v1//patrón/}**	Reemplaza en v1 la 1ª aparición de patrón por cadena Reemplaza en v1 todas las apariciones de patrón por cadena Elimina el patrón de v1
${v1#patrón}	Devuelve el valor v1 quitando el comienzo que coincida de forma mínima con el patrón. Si v1=a:b:c:d => **${v1#*:}** devolverá b:c:d
${v1##patrón}	Devuelve el valor v1 quitando el comienzo que coincida de forma máxima con el patrón. Si v1=a:b:c:d => **${v1##*:}** devolverá d
${v1%patrón}	Devuelve el valor v1 quitando el final que coincida de forma mínima con el patrón. Si v1=a:b:c:d => **${a%:*}** devolverá a:b:c
${v1%%patrón}	Devuelve el valor v1 quitando el final que coincida de forma máxima con el patrón. Si v1=a:b:c:d => **${a%%:*}** devolverá a
${v1^^}	Devuelve v1 en mayúsculas
${v1,,}	Devuelve v1 en minúsculas
${!v1}	Indirección: devuelve el valor de la var con nombre = valor en v1
a=${b:+valor}	Si b es nulo a=nulo. Sino a=valor.
a=${b:-valor}	Si b es nulo a=valor y b sigue siendo nulo. Sino a=$b
a=${b:=valor}	Si b es nulo a=valor y b=valor. Sino a=$b
a=${b:?"mensaje"}	Si b no es nulo a=$b. Sino imprime el mensaje y sale.

EVALUACIÓN DE EXPANSIONES	
○ En ocasiones cuando se utilizan variables es necesario evaluar una orden en dos pasadas: una primera para resolver las variables y otra para las expansiones	
eval orden	Evalúa la orden resolviendo variables y expansiones y luego ejecuta la orden **$u=user1; eval echo ~$u**

1.7. OTROS COMANDOS ÚTILES

OTROS COMANDOS	
echo "texto" 1>&2	Envía un mensaje desde el script a la salida de error
sleep n[s/m/h/d] sleep **0.5**	Espera n segundos(s)/minutos(m)/horas(h)/días(d) Espera 0.5 segundos

trap "c1;c2;c3..." s1 s2 s3...	Captura las señales s1 s2 s3... y cuando se reciben ejecuta los comandos c1;c2;c3... en lugar del comportamiento habitual ○ Utilizado cuando un script crea ficheros temporales y tiene que borrarlos antes de salir al recibir una señal ○ Las señales pueden indicarse con números o símbolos ○ Las mas habituales son: HUP(1), INT(2), QUIT(3), TERM(15) ○ La señal KILL(9) no puede capturarse ○ La señal INT corresponde a CTRL+C ○ La señal TSTP corresponde a CTRL+Z no siempre puede capturarse ○ Puede utilizarse como comando una llamada a función --- **trap 'rm -f $fichero_temp; exit' INT TERM HUP**
trap **-l** trap **- s1 s2 s3...**	Muestra la lista de señales Restaura el comportamiento de las señales s1, s2, s3...
mktemp	Crea un fichero del tipo del tipo: /tmp/tmp.hAEYKM8hNK y devuelve su nombre
seq inicio fin seq **inicio incremento fin**	Imprime una secuencia de números desde inicio a fin Imprime una secuencia de números según un incremento

Cursor e interfaz de gráfica en modo texto	
whiptail	Crea diálogos gráficos en modo texto Muestra un mensaje con título t y mensaje m **whiptail --title "t" --msgbox "m" h w** Muestra un mensaje con pregunta yes/no Devuelve la opción en $? **whiptail --title "t" --yesno "m"h w** http://xmodulo.com/create-dialog-boxes-interactive-shell-script.html
tput tput **cup x y** tput **sc** tput **rc** tput **lines** tput **cols** tput **cub n** tput **cuf n** tput **cuu n** tput **cud n** tput **clear** tput **el1** tput **el** tput **ed** tput **ich n** tput **il n** tput **bold** tput **dim** tput **smul**	Controla el movimiento del cursor en un terminal Mueve el cursor a la posición x y Guarda la posición del cursor Recupera la posición del cursor Muestra las líneas del terminal Muestra las columnas del terminal Mueve el cursor n posiciones a la izquierda Mueve el cursor n posiciones a la derecha Mueve el cursor n posiciones arriba Mueve el cursor n posiciones abajo Borra la pantalla Borra hasta el inicio de la línea Borra hasta el final de la línea Borra hasta el final de la pantalla Inserta n caracteres (desplazando el resto a la izquierda) Inserta n líneas (desplazando el resto hacia abajo) Entra en modo negrita Entra en modo brillo medio Entra en modo subrayado

tput **sgr0**	Sale de todos los modos
tput **setab [0-8]**	Cambia el color de fondo
tput **setaf [0-8]**	Cambia el color de la letra

2. PROGRAMACIÓN SHELL ESTRUCTURADA

2.1. ESTRUCTURAS CONDICIONALES

- Como **condición** puede utilizarse cualquier comando
- Se tendrá en cuenta su salida en $? para determinar si es cierta (0) o falsa (!=0)
- Se pueden escribir de forma lineal: hay que omitir el ; al separar las líneas que terminan en una palabra clave (then, elfi, else, do, etc.)

IF THEN

```
if condición
then
   comando 1
   comando 2
   …
fi
```

```
if test $# -eq 4
then
   echo "Demasiados parámetros"
fi
```

```
if [ $# -eq 4 ]
then
   echo "Demasiados parámetros"
   exit 2
fi
```

```
if  ! grep bash $1
then
   echo "El fichero pasado no es correcto"
   exit 1
fi
```

OPERADOR &&

En ocasiones es más sencillo utilizar el **operador &&** a un if (sin else o anidamientos)

```
if [ condición]; then
c1; c2; c3
fi
```

```
#Es equivalente a:
[ condición ] && (c1; c2; c3)
```

IF THEN-ELSE

```
if condición
then
   comando 1
   comando 2
   …
else
   comando a
   comando b
   ...
fi
```

```
if ((var<10))
then
   echo "El valor es menor que 10"
else
   echo "El valor es mayor a 10"
fi
```

IF THEN-ELIF-ELSE

```
if condición1
then
   comando 1
```

```
if ((var<10))
then
   echo "El valor es menor que 10"
elif ((var>20))
```

comando 2 … **elif** condición2 **then** comando a comando b … **else** comando x comando y ... **fi**	then echo "El valor es mayor a 20" else echo "El está entre 10 y 20" fi
CASE	
case valor **in** patrón1**)** c1 c2 .. **;;** patrón2**)** ca cb .. ;; ***)** cx cy .. **esac**	○ El patrón puede usar expresiones regulares **b*)** #Empieza por b **[Yy]* | OK | ok)** #Empieza por Y ó y ó es OK ó es ok ○ La última opción ***)** es opcional y se entra si no coincide con ninguna otra. --- case $v in A|a) echo "Ha introducido A" ;; B|b) echo "Ha introducido B" ;; C|c) echo "Ha introducido C" ;; *) echo "Opción incorrecta" ;; esac

2.2. ESTRUCTURAS REPETITIVAS

BUCLE FOR (estilo C)	
for ((ini;cond;act**))** **do** comando1 comando2 … **done**	for ((i=0;i<=10;i++)) do echo $i done
BUCLE FOR-IN	
for var **in** lista_valores **do** comando1 comando2 … **done**	○ El delimitador para los elementos de la lista es **$IFS.** ○ Puede cambiarse por otro pero conviene **restaurarlo.** ○ Al recorrer los argumentos lo más recomendable es **"$@"** --- for dia in **Lun Mar Mie Jue Vie Sab Dom** #lista de valores do echo $dia

	done
	--- 	
	for i in ***.sh** #Todos los ficheros de ./ terminados en .sh
	do
	  echo $i
	done
	--- 	
	for i in **$(ls /)** #Todos los ficheros de /
	do
	  echo $i
	done
	--- 	
	IFS=":"
	for i in **$PATH**
	do
	  echo $i
	done
	--- 	
	for i in **{1..100}.txt** #Genera los números del 1 al 100
	do
	  touch ${i}
	done

BUCLE WHILE	

while condición
do
  comando1
  comando2
…
done
	○ Ejecuta el código **mientras** se cumpla la condición
	○ Si al comenzar no se cumple la condición no se ejecuta
	--- 	
	while [$limite -gt $i]
	do
	  echo "Valor $i"
	  i=$(($i+1))
	done
	--- 	
	i=1
	while read linea
	do
	  echo "Línea $i: $linea"
	  i=$((i+1))
	done < fichero #Redirección a la función
	--- 	
	path=$PATH #Bash es case-sensitive al nombrar variables
	while [$path]
	do
	  dir_cola=${path%%:*}
	  echo $dir_cola
	  if [$dir_cola = $path]; then break; fi
	  path=${path#*:}
	done

BUCLE UNTIL	
until condición	○ Ejecuta el código **hasta** que se cumpla la condición

do
  comando1
  comando2
  …
done
	○ Si al comenzar se cumple la condición no se ejecuta
	--- 	
	echo "Adivina el número entre 1 y 100"
	num_secreto=$((1+$RANDOM%100+1))
	until [$num_secreto = "$num_leido"]
	do
	  read -p "Introduzca número: " num_leido
	done

BREAK Y CONTINUE	

break
 	
continue
	Dentro de un bucle sale de inmediato y sigue por el comando siguiente al done
	Dentro de un bucle termina la vuelta actual del bucle y continúa por la primera instrucción del bucle
	--- 	
	#Imprime del 1 al día mes actual saltando los múltiplos 3
	dia_mes=`date +%d` #¿Aquí o dentro del for?
	for ((i=0;i<=31;i++))
	do
	  if (($dia_mes==$i)); then break; fi
	  if (($i%3==0)); then continue; fi
	  echo $i
	done

SELECT	
select var **in** lista_valores **do** comando1 comando2 … **done**	Muestra un menú con la lista de valores numerada y un prompt (configurable en $PS3) para solicitar uno de ellos al usuario. Cuando se inserta un número se almacena en $var el valor correspondiente y se ejecutan los comandos. El proceso se repite hasta que se sale con un break. --- PS3="Introduzca opción: " select i in rojo blanco verde azul amarillo do if [$i]; then echo "El color elegido es: $2"; break; fi done

2.3. TRATAMIENTO DE PARÁMETROS

REGLAS ESTÁNDAR PARA LA SINTAXIS DE PARÁMETROS
○ El **orden** relativo de las opciones es indiferente ○ Cada opción va **precedida** por un símbolo - ○ El identificador de cada opción es de una **única letra** en su versión corta ○ Cada identificador puede tener una versión más larga que irá precedida por -- ○ Las opciones sin argumentos pueden **agruparse** en cualquier orden con un único - ○ Si una opción tiene un **argumento** debe preceder a la opción separado por un espacio
PROCESADO DE PARÁMETROS

- Una forma de procesar los parámetros de un script es utilizar múltiples estructuras repetitivas y comparativas para evaluar el valor de los parámetros recibidos.
- Si se tienen múltiples parámetros puede ser bastante engorroso.
- Para simplificar se utiliza el comando interno **getopts**.
  - No soporta parámetros en versión larga (--param).
- Existe un comando externo llamado **getopt** con funcionalidad similar.

getopts lista_opc var [args]

```
while getopts "hr:v" option
do
   case $option in
   h)
      hflag=1
      ;;
   r)
      rvalue=$OPTARG
      ;;
   v)
      vflag=1
   \?)
      echo "$0: opción no
soportada $OPTARG"
      exit 2
      ;;
   esac
done
```

Cada vez que se llama a getopts recorre la lista de parámetros recibidos buscando la siguiente opción

[args]: si se utiliza en lugar de utilizar los parámetros del script ($1, $2...) se utilizan estos.

lista_opc: contiene los caracteres que representan los posibles parámetros.
- Si alguno de ellos puede recibir un argumento se coloca detrás el símbolo :
- Si se coloca al principio de la lista el símbolo : getopts no muestra los errores

var: el nombre de la variable en que se guardará el valor del siguiente parámetro

$OPTIND: guarda en cada llamada el índice del argumento siguiente.
- Si encuentra una opción no soportada guarda el símbolo ?

$OPTARG: guarda en cada llamada
- en caso de opciones con argumentos, su valor

2.4. FUNCIONES

FUNCIONES

```
function nom_fun [()]
{
   comando1
   comando2
   ...
   [return [valor]]
}
...
nom_fun a1 a2 a3
```

```
nom_fun ()
{
   comando1
   comando2
   ...
   [return [valor]]
}
...
nom_fun a1 a2 a3
```

- Una función es similar a una llamada a otro script
- A diferencia de una llamada a otro script:
  - Se ejecutan dentro del mismo entorno Bash
  - Su código tiene que cargarse aunque no se invoquen
- Los **parámetros** que recibe se tratan igual que los de un script ($1, $2, etc.)
- $0 no cambia. En su lugar **$FUNCNAME** tiene el nombre de la función
- El comando **return** termina la función y devuelve el valor numérico pasado en **$?**
- El comando **exit** dentro de la función termina el script
- Todas las variables por defecto son **globales** y compartidas
- Si se quiere una variable **local** debe declararse como: **local** nombre_variable=valor
- La llamada a función puede redireccionarse a un fichero

source fichero_funciones	Carga las funciones definidas en fichero_funciones en el script actual