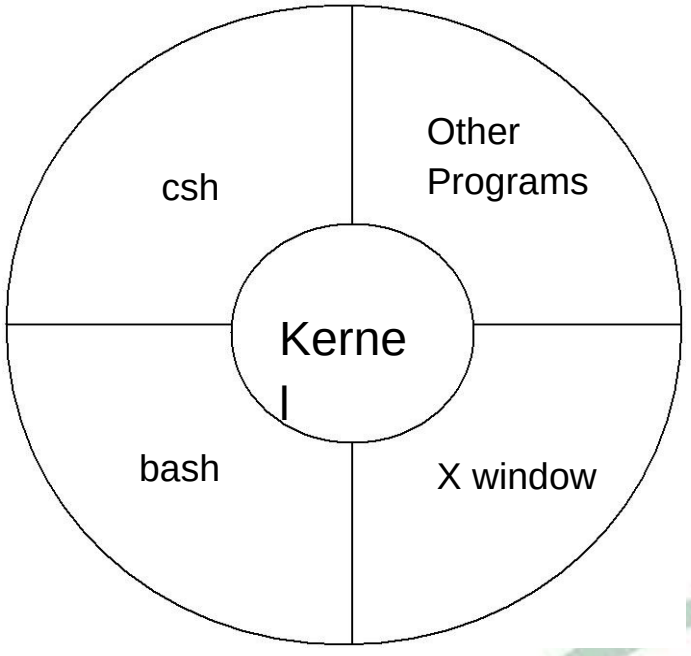
SCRIPTS DE SHELL

Bash

¿Qué es el Shell?

- Es la interfaz entre el usuario final y el Sistema Operativo.
- *No es el S.O.*
- Existen múltiples versiones y podemos averiguar cual tenemos instalada haciendo:

```
$ /bin/sh -version
```



Las *Shells*

En UNIX existen **múltiples** Shells:

– Bourne shell (sh), C shell (csh), Korn shell (ksh), TC shell (tcsh), Bourne Again shell (bash).
La más popular es la "bash" shell.

– Bash incorpora las prestaciones más útiles de la Korn shell (ksh) y la C shell (csh).

– Ofrece mejoras funcionales sobre otras *shells* desde el punto de vista de programación y de su uso interactivo.

¿Programación o scripting?

Bash no es únicamente una excelente shell por línea de comandos. También es un lenguaje de scripting en sí mismo.

El shell scripting permite utilizar las capacidades de la shell para automatizar múltitud de tareas que, de otra forma, requerirían múltiples comandos introducidos de forma manual.

Lenguaje de programación vs scripting:

– Los lenguajes de programación son, en general, más potentes y mucho más rápidos que los lenguajes de scripting.

– Los lenguajes de programación comienzan desde el código fuente, que se compilan para crear los ejecutables (lo que permite que los programas sean fácilmente portables entre diferentes SSOO).

¿Programación o scripting?

Un lenguaje de scripting también comienza por el código fuente, pero no se compila en un ejecutable.

En su lugar, un intérprete lee las instrucciones del fichero fuente y las ejecuta secuecialmente.

Los programas interpretados sin, en general, más lentos que los compilados.

La principal ventaja reside en que el fichero de código fuente es fácilmente portable a cualquier sistema operativo.

El primer programa bash

Necesitaremos acceso a un editor de textos:
– gedit, nano, vi, emacs .

Arrancamos el editor de textos preferido

```
$ nano
```

y escribimos el siguiente código:

```
#!/bin/bash
echo “Hola Mundo”
```

Guardamos el fichero como hola.sh La primera línea indica a Linux que debe utilizar el intérprete bash para ejecutar el script. Hacemos que el script sea ejecutable

```
$ chmod 700 hola.sh
$ ls –l
rwx------ hola.sh
```

El primer programa bash

Ejecutemos el programa.

```
$ hola.sh
bash: hola.sh: no se encontró la orden
```

El directorio home (el lugar donde está ubicado el programa hola.sh) no está en la variable de entorno PATH.

```
$echo $PATH
:bin:/usr/bin:…
```

Debemos especificar el path de hola.sh

```
$/home/alumno/hola.sh
$./hola.sh
```

Si queremos ejecutar el script sin necesidad de dar permisos podemos usar:

```
source hola.sh
```

Sirve para probarlo antes de dar permisos de ejecución.

El segundo programa bash

Supongamos que cierto proceso genera un montón de archivos que queremos eliminar (o guardar).

Escribamos un programa que *copie* todos los ficheros especificados en un directorio y lo borre junto con todo su contenido …

El siguiente código permite hacer esto:

```
$ mkdir papelera
$ cp *.log papelera
$ rm -rf papelera
```

El segundo programa bash

Si lo ejecutamos veremos que da muchos mensajes de error.

- Si ya existe un archivo
- Si no existe un archivo
- ...

Sería bueno poder introducir controles para decidir que hacer según lo que pase.

Variables

Como en cualquier lenguaje de programación, en shell scripting se pueden utilizar variables.
Todos los valores son almacenados como cadenas de texto. Si hacemos:

```
NUMERO=4
echo $NUMERO + 3
```

```
Mostrará 4+3
```

También hay operadores matemáticos que convierten las variables en números para el cálculo.

No es necesario declarar una variable, simplemente asignándole un valor a su referencia será suficiente para crearla.

Variables

Ejemplo

```
#!/bin/bash
SALUDO="Hola Mundo!" # sin espacios en blanco
echo $SALUDO
```

La línea 2 crea una variable denominada SALUDO y le asigna la cadena ‘Hola Mundo’.

El valor de esta variable se recupera anteponiendo un símbolo ‘$’ a su nombre.

Variables

Atención !

El lenguaje de programación de la shell no hace comprobaciones de los tipos de las variables.

Esto significa que una misma variable puede contener datos numéricos o de texto.

```
contador=0
contador=Domingo
```

La conmutación del TIPO de una variable puede conllevar a confusión.

Buena práctica: asociar siempre el mismo tipo de dato a una variable en el contexto de un mismo script.

Variables

Atención !

El caracter de escape de la bash es \ y preserva el valor literal del siguiente caracter al que acompaña:

```
$ ls \*
ls: *: No existe el archive o el directorio
```

Interpreta el símbolo de forma literal como *, no como comodín, que es su significado.

Si queremos escribir una línea en varias, usaremos \ al final de línea. Ejemplo:

```
echo esto \
es una sola línea \
aunque ocupe tres en pantalla
```

Comillas simples y dobles

Cuando se asignen datos caracter que contengan espacios en blanco o caracteres especiales, se deberá encerrar entre comillas simples o dobles.

Las dobles comillas harán que si, en su contenido se referencia una variable, ésta sea resuelta a su valor

```
var="cadena de prueba"
nuevavar="Valor de var es $var"
echo $nuevavar
```

```
Valor de var es cadena de prueba
```

Hacer un script que muestre por pantalla:

```
Valor de ‘var’ es “cadena de prueba”
```

Comillas simples y dobles

Las comillas simples mostrarán una cadena de caracteres de forma literal sin resolución de variables

```
var='cadena de prueba'
nuevavar='Valor de var es $var'
echo $nuevavar
Valor de var es $var
```

Expansiones

En linux podemos usar varios tipos de expansiones:

1. **$(orden)**
2. **`orden`**
3. **$((expresión))**
4. **$[expresión]**

NOTA: (hay que dejar blancos antes y después de los corchetes)

Expansiones - $(orden)

Permite ejecutar lo que se encuentre entre paréntesis y devuelve su salida.

$(orden)

Ejemplos:

```
echo $(pwd)
```
#ejecuta la orden pwd y muestra en pantalla su resultado.

Las siguientes órdenes copian el fichero de interfaces en el directorio actual con el nombre red010115 suponiendo que estamos a 1 de Enero de 2015.

```
NOMBRE-FICH=”red”$(date +%d%m%y)”.conf”
cp /etc/network/interfaces $NOMBRE-FICH
```

Expansiones - `orden`

Igualmente permite ejecutar “orden” y devuelve su salida.

`orden`

Ejemplos:

```
echo `pwd`  #ejecuta la orden pwd y muestra en pantalla su resultado.
```

```
echo “Tenemos `find /home/usuario –iname “*.sh”|wc –l` scripts en nuestro directorio de usuario”
```

Expansiones – $((expresión)) $[expresión]

Igual que las anteriores resuelve lo indicado entre los paréntesis.

Ejemplos:

```
NUMERO=4
echo $(($NUMERO+3) #sería lo mismo poner echo $[$NUMERO+3]
Obtenemos el valor de 7.
```

El comando let permite realizar operaciones aritméticas como la anterior sin poner la expansión ni el dolar para las variables.

```
NUMERO=4
let SUMA=NUMERO+3
echo $SUMA
```

EJERCICIOS

Crea un script media.sh que calcule la media de tres notas.

Usa el comando bc para mostrar el resultado con dos decimales.

Variables de Entorno

Hay dos tipos de variables de entorno:

- Variables locales
- Variables del entorno

Las variables de entorno se establecen por el sistema y se pueden encontrar utilizando el comando env.

Las variables de entorno contiene valores especiales. Por ejemplo:

```
$ echo $SHELL
/bin/bash
$ echo $PATH
/usr/X11R6/bin:/usr/local/bin:/bin:/usr/bin
```

Variables de Entorno

Las variables de entorno se definen en /etc/profile, /etc/profile.d/ y ~/.bash_profile.

Estos ficheros son de inicialización y son leídos cuando se invoca la bash shell.

Cuando la login shell sale, la bash lee ~/.bash_logout

Variables de Entorno

HOME: argumento por defecto (directorio home) del comando cd.

PATH: el path de búsqueda de comandos. Es una lista de directorios separados por ':' en los que se buscar cuando se teclea cualquier comando.

Normalmente, introducimos los comandos de la siguiente manera:

$ ./papelera.sh

Estableciendo **PATH=$PATH:.** nuestro directorio de trabajo se incluye en path de búsqueda de los comando y simplemente podremos introducir:

$ papelera.sh

Variables de Entorno

LOGNAME: contiene el nombre de usuario

HOSTNAME: contiene el nombre de la máquina

MACHTYPE: sistema hardware

PS1: secuencia de caracteres mostrados antes del prompt

- \t hora
- \d fecha
- \w directorio actual
- \W última parte del directorio actual
- \u nombre de usuario
- \$ caracter del prompt

UID: contiene el id del usuario que no puede ser modificado

SHLVL: contiene el nivel de anidamiento de la shell

http://www.linuxtotal.com.mx/index.php?cont=info_tips_017

Variables Intrínsecas

$#: número argumentos

$*: todos los argumentos de la shell

$@: semejante a la variable anterior

$-: opciones suministradas a la shell

$?: devolver valor de la última orden ejecutada

$!: identificación del proceso de la última orden que comenzó con &

Comando `read echo`

El comando read nos permite solicitar un valor de entrada para almacenarlo en una variable

El comando echo muestra en pantalla lo mostrado a continuación

```
echo –e “Escriba su nombre: \c” #Sin salto de línea
read nombre
echo –e “Escriba su nombre: \n” #Con  salto de línea
read nombre
```

Ejemplo (`leer.sh`)

```
#!/bin/bash
echo -n “Introduzca nombre de fichero a borrar: ”
read fichero
rm -i $fichero
echo “Fichero $fichero borrado!"
```

Comando `read echo`

Opciones

read –s (no hace echo de la entrada)

read –nN (acepta sólo N caracteres de entrada)

read –p “mensaje” (muestra un mensaje)

read –tT (acepta una entrada por un tiempo máximo de T segundos)

Ejemplo (pregunta.sh)

```
$ read –s –n1 -p “si (S) o no (N)?”
$ ./pregunta
si (S) o no (N) ? S
```

Sustitución de comandos

El símbolo “`” tiene un uso diferente de “ ´ ”. Se utiliza para sustitución de instrucciones. Es decir si dentro de un script aparece el texto “ `comando` ” entonces se ejecutará la orden que está entre las “`” .

```
$ LISTA=`ls`
$ echo $LISTA # Lista los archivos
hola.sh leer.sh
```

Otra forma de realizar la sustitución de comandos: $(comando)

```
$ LISTA=$(ls)
$ echo $LISTA
hola.sh leer.sh
```

Operadores aritméticos

+	suma
-	resta
*	multiplicación
/	división
**	exponenciación
%	módulo

Ejemplo (expresiones matemáticas sin evaluar por bash)

```
$ a=(5+2)*3
$ echo $a
$ B=2**3
$ echo $a+$b
```

Evaluación aritmética

Ejemplo (operaciones.sh)

```
#!/bin/bash
echo -n “Introduzca un primer número: ”; read x
echo -n “Introduzca un segundo número : ”;read y
suma=$(($x + $y))
resta=$(($ x - $y)) mul=$(($x * $y))
div=$(($x / $y))
mod=$(($x % $y))
# imprimimos las respuestas:
echo “Suma: $suma”
echo “Resta: $resta”
echo “Multiplicación: $mul”
echo “División: $div”
echo “Módulo: $mod”
```

NOTA: modificarlo usando bc donde sea necesario

Estructuras de control 1 - condicional

La forma más básica es:

```
if [expresión]; then
   instrucciones elif [expresión];
   then instrucciones
   else instrucciones
   fi
fi
```

Las secciones elif(else if) y else son opcionales

Expresiones

Una expresión puede ser: comparación de cadenas, comparación numérica, operadores de fichero y operadores lógicos y se representa mediante [expresión]:

Comparación de cadenas:

- = Igual
- != No igual
- -n evalúa si la longitud de la cadena es superior a 0
- -z evalúa si la longitud de la cadena es igual a 0

Ejemplos:

- [s1 = s2] (true si s1 es igual a s2, sino false)
- [s1 != s2] (true si s1 no es igual a s2, sino false)
- [s1] (true si s1 no está vacía, sino false)
- [-n s1] (true si s1 tiene longitud mayor que 0, sino false)
- [-z s2] (true si s2 tiene longitud 0, sino false)

Expresiones

Comparación numérica - Operadores:

- -eq (n1 igual a n2)
- -ge (n1 mayor o igual a n2)
- -le (n1 menor o igual a n2)
- -ne (n1 no igual a n2)
- -gt (n1 mayor que n2)
- -lt (n1 menor que n2)

Ejemplos:

- [n1 -eq n2]
- [n1 -ge n2]
- [n1 -le n2]
- [n1 -ne n2]
- [n1 -gt n2]
- [n1 -lt n2]

Ejemplo

Ejemplo(comparacion0.sh)

```
#!/bin/bash

echo -n “Introduzca su nombre de usuario: "
read nombre

if [ $nombre = $USER ];
then
echo “Hola, $nombre. Cómo está hoy?”
else
    echo “Tú no eres $nombre !!!”
fi
```

Ejemplo

Ejemplo(comparacion1.sh)

```
#!/bin/bash
echo -n “Introduzca un número 1<x<10: "
read num

if [ $num -lt 10 ]; then
if [ $num -gt 1 ]; then
echo “$num*$num=$(($num*$num))”
else
echo “Número fuera de rango !”
fi
else
echo “Número fuera de rango !”
fi
```

EJERCICIOS

1.- Programa que indique si un nº es par o impar.

2.- Programa que pida tres cifras y diga si forman un nº capicua o no.

3.- Programa que indique si tienes 1,2,3 o ningún hijos

4.- Programa que diga si estamos en viernes.

5.- Programa que indique si estamos en la primera o segunda quincena del mes.

Expresiones

Operadores de archivos:

- -d nombre_fichero verifica si el path dado es un directorio
- -a nombre_fichero verifica si el path dado es un fichero
- -f nombre_fichero verifica si el path dado es un archivo regular
- -h nombre_fichero verifica si el path dado en un link simbólico
- -e nombre_fichero verifica si el fichero existe
- -s nombre_fichero verifica si el fichero tiene un tamaño mayor a 0
- -r nombre_fichero verifica si el fichero tiene permiso de lectura
- -w nombre_fichero verifica si el fichero tiene permiso de escritura
- -x nombre_fichero verifica si el fichero tiene permiso de ejecución

- fichero1 -nt fichero2 verifica si el fichero1 es más nuevo que fichero2
- fichero1 -ot fichero2 verifica si el fichero1 es más viejo que fichero2

Ejemplo

Ejemplo(comparacion_archivos.s)

```
#!/bin/bash
if [ -f /etc/fstab ]; then cp /etc/fstab .
echo “Hecho.”
else
echo “Archivo /etc/fstab no existe.”
exit 1
fi
```

Expresiones

Operadores lógicos:

! NOT

-a AND

-o OR

Ejemplo(comparacion_logica1.sh)

```
#!/bin/bash
echo -n “Introduzca un número entre 1<x<10:”
read num
if [ $num -gt 1 –a $num -lt 10 ]; then
  echo “$num*$num=$(($num*$num))”
else
   echo “Número introducido incorrecto !”
fi
```

Expresiones

Operadores lógicos:

&& AND

||OR

Ejemplo(comparacion_logica2.sh)

```
#!/bin/bash
echo -n “Introduzca un número 1<x<10: " read num
if[ $num -gt 1 ] && [ $num -lt 10 ]; then
echo $num*$num=$(($num*$num))” else
echo “Número introducido incorrecto !”
fi
```

EJERCICIOS

1.- Programa que indique si en el directorio actual hay más de 10 ficheros.

2.- Programa que nos pida la edad y nos diga en qué década nacimos (la de los 80,90,etc) Suponemos que todo el mundo tiene más de 15 años.

3.- Script que pida por pantalla el nombre de un mes (enero, febrero, marzo…) y nos diga el número de días que tiene. Suponemos que no existen los años bisiestas.

"30 días trae noviembre con abril, junio y septiembre; los demás 31 menos febrerillo el loco con sus días, 28"

Parámetros de la shell

Los parámetros posicionales se asignan desde la shell cuando se invoca. Parámetro posicional "N" se referencia como "${N}", o "$N" cuando "N" lo forma un sólo dígito

Parámetros especiales

- $# número de parámetros pasados
- $0 devuelve el nombre del shell script que se está ejecutando y su ubicación en el sistema de archivos
- $* devuelve en una cadena de caracteres todos los parámetros pasados al script
- $@ devuelve un array con los parámetros pasados al script

Ejemplo(parametros.sh)

```
#!/bin/bash
echo "$#; $0; $1; $2; $*; $@"
```

```
$ ./parametros.sh estudiante1 estudiante2
2; ./parametros.sh; estudiante1; estudiante2; estudiante1 estudiante2; estudiante1
estudiante2
```

Parámetros de la shell

Cuando el número de parámetros supera a 9, deberemos usar la orden shift para tratarlos todos.

Ejemplo(parametros1.sh)

```
while [“S1” != “ “ ]; do
  echo $1
  shift
done
```

Instrucción Case

```
case $var in
 val1)
      instrucciones;;
 val2)
      instrucciones;;
 *)
     Instrucciones;;
esac
```

Ejemplo

Ejemplo(case.sh)

```
#!/bin/bash
echo -n “Introduzca un número entre 1 < x < 10: ” read x
case $x in
   1) echo “Valor de x es 1.”;;
   2) echo “Valor de x es 2.”;;
   3) echo “Valor de x es 3.”;;
   4) echo “Valor de x es 4.”;;
   5) echo “Valor de x es 5.”;;
   6) echo “Valor de x es 6.”;;
   7) echo “Valor de x es 7.”;;
   8) echo “Valor de x es 8.”;;
   9) echo “Valor de x es 9.”;;
   0 | 10) echo “Número incorrecto.”;; *) echo “Valor no
      reconocido.”;;
esac
```

EJERCICIOS

1.- Programa que pida los tres dígitos primeros del código postal e indique la ciudad donde nos encontramos.

2.- Programa que pida una letra en minúsculas y nos diga si es una vocal, consonante o es un caracter especial.

Instrucción Select

Select VARIABLE in Cjto_opciones; do

aquí la variable toma los valores del menú presentado en pantalla

done

Ejemplo(select.sh)

```
#!/bin/bash
select OPCION in hola adios Salir: do
  if [ $OPCION = “hola” ]; then
       echo “Hola, buenos días”
  elif [ $OPCION = “adios” ]; then
     echo “Adiós, hasta mañana”
  else
     echo “Nos vamos”
     break
  fi
done
```

Instrucción for

```
for var in lista
do
  instrucciones …
done
```

Ejemplo(for1.sh)

```
#!/bin/bash
let sum=0
for num in 1 2 3 4 5
do
  let “sum = $sum + $num”
done
echo $sum
```

Instrucción for

Ejemplo(for2.sh)

```
#!/bin/bash
for x in papel lapiz boli; do
    echo “El valor de la variable x es: $x”
    sleep 1
done
```

Ejemplo(for3.sh)

```
#!/bin/bash
for x in “papel A4” “lapiz STADTLER” “boli BIC”; do
     echo “El valor de la variable x es: $x”
     sleep 1
done
```

Ejemplo(for4.sh)

```
#!/bin/bash
lista=“antonio luis maria pepa”
for x in $lista
do
  echo “El valor de la variable x es: $x”;sleep 1
done
```

Instrucción for

Ejemplo(for5.sh)

```
#!/bin/bash
# Lista todos los ficheros del directorio actual
for x in *
do
    ls -l $x
    sleep 1
done
```

Ejemplo(for6.sh)

```
#!/bin/bash
Lista todos los ficheros del directorio /bin
for x in /bin
do
  ls -l $x
done
```

Instrucción for

Ejemplo(for7.sh)

```
#!/bin/bash
read –p “Introduzca el nombre de un directorio: ”
  directorio
echo “enlaces simbólicos en el directorio $directorio
  “
for fichero in $( find $directorio -type l )
 do
 echo "$fichero"
 done
```

EJERCICIOS

1.- Script que sume 100 números desde el nº 1 al 100.
2.- Script que liste todos los nombres de los ficheros que terminen en sh de nuestro directorio.
3.- Script que muestre los números pares entre el 2 y el 40.
4.- Script que haga un bucle infinito (que nunca salga) y que muestre el doble de un número pedido al usuario.
5.- Modifique el script anterior para salir del bucle (usa break).

Bucle tipo C

Estructura tipo C alternativa para for

```
for (( EXPR1 ; EXPR2 ; EXPR3 )) do
instrucciones
done
```

Ejemplo(for8.sh)

```
#!/bin/bash
echo “Introduzca un número: ”; read x
let sum=0
for (( i=1 ; $i<$x ; i=$i+1 )) ; do
   let “sum = $sum + $i”
done
echo “La suma de los primeros $x números es: $sum”
```

Estructura while

```
while expresion_evalua_a_true
do
    Instrucciones
Done
```

- Ejemplo(while.sh)

```
#!/bin/bash
echo –n “Introduzca un número: ”; read x
let sum=0; let i=1
while [ $i –le $x ]; do
let “sum = $sum + $i”
let “i = $i + 1”
done
echo “La suma de los primeros $x números es: $sum”
```

Estructura until

```
until [expression_evalua_a_true]
   do
   instrucciones
Done
```

Ejemplo(until.sh)

```
#!/bin/bash
echo “Introduzca un número: ”; read x
echo ;
until [ $x -le 0 ]; do
   echo $x
   x=$(($x – 1))
   sleep 1
done
echo “FIN”
```

Funciones

Ejemplo(func1.sh)

```
#!/bin/bash
function suma()
{
  let c=$a+$b
  echo “Suma: $c”
}
echo “Introduzca el valor del primer número: ” ; read a
echo “Introduzca el valor del segundo número: ” ; read
  b
suma
```

Comando exit

El comando exit se puede utilizar para finalizar la ejecución de un script o para devolver un valor, el cuál estará disponible al proceso padre del script.

– Cuando un script termina con `exit` sin parámetros, el estado de salida será el del último comando ejecutado en el script

```
#!/bin/bash

COMANDO_1

. . .

# sale con el estado de la ejecución
#del último comando.
ULTIMO_COMANDO

exit
```

```
#!/bin/bash

COMANDO_1

. . .

#sale con el estado de la ejecución
#del último comando.
ULTIMO_COMANDO

exit $?
```

Valores devueltos por las órdenes

Existe un parámetro especial, el $?, que nos devuelve el valor del resultado de la última orden del sistema. Tendrá valor 0 si todo ha ido bien, y otro cualquiera en caso de fallo.

Para comprobarlo haremos lo siguiente:

`cd /juegos` (hacemos un cd a un directorio que no existe)

Luego miramos el contenido del parámetro $?

```
echo $?
```

Comprobaremos que vale 1, que indica que la última orden no funcionó correctamente.

Luego haremos lo mismo para un directorio que exista y comprobaremos que el valor de $? es 0.

Arrays con bucles

Crear un array

```
$ mascota[0]=perro
$ mascota[1]=gato
$ mascota[2]=pez
$ pet=( perro gato pez )
```

Longitud máxima de un array son 1024 elementos. Para extraer una entrada del array ${array[i]}

```
$ echo ${mascota[0]} perro
$ echo ${mascota[2]} pez
```

Arrays

Para extraer todos los elementos se utiliza un asterisco:

echo ${array[*]}

Para saber cuántos elementos hay en el array:

echo ${#array[*]}

Podemos combinar los arrays con bucles utilizando for:

```
for x in ${array[*]}
  do
echo ${array[$x]}
done
```

Depuración

Bash ofrece dos formas de depurar los shell scripts

-v : muestra cada línea completa del script antes de ser ejecutada

-x : muestra cada línea abreviada del script antes de ser ejecutada

Uso: #!/bin/bash –v, o #!/bin/bash –x

Ejemplo (depuracion.sh)

```
#!/bin/bash –x
# Comprueba que un fichero exista o no
Function comp{
echo “~/$1”
if [ -e ~/$1 ]; then
return 0
else return 1
fi }
echo –n “Introduzca el nombre del archivo: ”; read f
if comp $f
 then echo “$f existe”
 else echo “$f no existe”
fi
```