Shell Scripting

Varios

Argumentos

$#	Number of arguments
$*	All arguments
$@	All arguments, starting from first
$1	First argument

Sustitución de comandos

```
$(comando)

`comando`
```

por ejemplo

```
$ echo $(pwd)
/home/darthmendez
```

Variables

```
NOMBRE="Mariano"
```

Con el $<nombre variable> se obtiene el valor:

```
$ echo $NOMBRE
Mariano
```

Se puede obtener el valor de una variable dentro de un string, usando la forma canonica del signo $:

```
$ echo "Mi nombre es $NOMBRE"
Mi nombre es Mariano
```

O se puede obtener nediante el operador de sustutución que es ${}:

```
$ echo "Mi nombre es ${NOMBRE}"
Mi nombre es Mariano
```

```
$ echo 'Mi nombre es $NOMBRE'
Mi nombre es $NOMBRE
```

```
$ NOMBRE+=" Mendez"
$ echo $NOMBRE
Mariano Mendez
```

Arreglos

```
foo[0]="primero"  # asigna al primer elemento  "first"
foo[1]="segundo"  # asigna al segundo elemento "second"
```

Otra forma de asignar es la asignacion por extensión:

```
$ array=(a b c)
$ echo $array       # prints a
$ echo ${array[@]}  # prints a b c
$ echo ${array[*]}  # prints a b c
```

Estructuras de Control

test

Este comando chequea el tipo de archivos y compara valores (man test)

```
test expression
[ expresion ]
```

ejemplo

```
[ $V -eq 0 ]
```

Valores Numéricos:

n1 -eq n2	n1 = n2
n1 -ne n2	n1 ≠ n1
n1 -gt n2	n1 > n2
n1 –ge n2	n1 ≥ n2
n1 -lt n2	n1 < n2
n1 -le n2	n1 ≤ n2

Strings:

“$VAR” = “cad”	$VAR vale “cad”.
“$VAR” != “cad”	$VAR vale algo distinto de “cad”.
-z “$VAR” “$VAR”	$VAR está vacía. Equivale a “$VAR” = “”
-n “$VAR”	$VAR no está vacía. Equivale a “$VAR” != “” o ! -z

Archivos:

-e “$FILE”	$FILE existe.
-f “$FILE”	$FILE existe y es regular.
-h “$FILE”	$FILE existe y es un enlace simbólico
-d “$DIR”	$DIR existe y es un archivo de tipo directorio
-p “$FILE”	$FILE existe y es un archivo especial tubería (pipe)
-b “$FILE”	$FILE existe y es un archivo especial de bloques
-c “$FILE”	$FILE existe y es un archivo especial de caracteres
-r “$FILE”	$FILE existe y puede leerse
-w “$FILE”	$FILE existe y puede modificarse
-x “$FILE”	$FILE existe y puede ejecutarse
-s “$FILE”	$FILE existe y su tamaño es mayor de cero bytes

if-fi

If ejecuta comando, si el valor de retorno de la ejecución es 0 se ejecuta comando1 sino se ejecuta comando2

```
if comando;
then
    comando1
else
    comando2
fi
```

ejemplo:

```
VALOR=1
if [$VALOR=1];then
  comando1
  comando2
else
  comando3
  comando4
fi
```

ejemplo más complejo:

```
if command ; then
   comando1
   comando2
elif comando3 ; then
   comando4
   comando5
   comando6
else
  echo 'todo fallo'
  exit 1 # termina el script con valor no 0
fi
```

Loops

for

El for tiene dos sintaxis, la primera:

```
for name [ [ in [ word ... ] ] ; ] do list ; done
```

El funcionamiento es el siguiente, se crea una lista expandiendo lo que se encuentra despues del in, la varianle con el nombre name tome el valor de cada elemento de la lista, y se ejecuta la lista list de comandos para cada valor de name.

```
for file in *.txt ; do
   mv "$file" "$file.bak"
done
```

la segunda forma de sintaxis de for es la misma que la del lenguaje C:

```
for (( init ; cond ; incr )) ; do list ; done
```

While

La sintaxis del while es :

```
while comando; do
    comando1
    comando2
    comando3
done
```

ejemplo:

```
while [[ -e wait.txt ]] ; do
    sleep 3 # "sleep" for three seconds
done
```

Until

La sintaxis de until es muy similar a la de while:

```
until comando ; do
    comando1
    comando2
done
```

```
until [[ -e proceed.txt ]] ; do
    sleep 3 # "sleep" for three seconds
done
```

Funciones

A diferencia de las funciones de los lenguajes de programación “reales”, las funciones Bash no pueden devolver un valor cuando se las llama. Cuando se completa una función bash, su valor de retorno es el estado de la última instrucción ejecutada en la función, 0 para el éxito y un número decimal distinto de cero en el rango de 1 a 255 para el fracaso.

El estado de retorno puede especificarse utilizando la palabra clave return y se asigna a la variable $?. La utilización de return termina la función.

En un script bash se pueden crear funciones, la sintaxis es la siguiente:

```
myfunc() {
    echo "hello $1"
}

# sintaxis alternativa
function myfunc() {
    echo "hello $1"
}
```

Valore de retorno:

```
myfunc() {
    local myresult='some value'
    echo $myresult
}

result="$(myfunc)"
```

Expresiones regulares

Las expresiones regulares son un medio para describir patrones de texto

puntos de anclaje:

^	inicio de línea
$	fin de línea
<	principio de palabra
>	fin de palabra
b	límite de palabra

cuantificadores:

?	el carácter aparece ninguna o una vez.

*	cero, una o varias veces.
+	al menos una vez.
{4}	cuatro veces.
{4,10}	entre 4 y 10 veces

tipos de caracteres:

- [:alnum:] [A-Za-z0-9] Caracteres alfanuméricos (letras y números)
- [:word:] [A-Za-z0-9_] Caracteres alfanuméricos y "_"
- [:alpha:] [A-Za-z] Caracteres alfabéticos
- [:blank:] [t] Espacio y tabulador
- [:space:] [trnvf] Espacios
- [:digit:] [0-9] Dígitos
- [:lower:] [a-z] Letras minúsculas
- [:upper:] [A-Z] Letras mayúsculas
- [:punct:] [][!"#$%&'()*,./:;<=>?@^_`{}~-] Caracteres de puntuación

    - Obtener todas las palabras que contengan alguna letra "a"

      $ grep -E –color 'a' hobbit.txt

- Ontener todas las palabras que contengan el string "ar"

  $ grep -E –color 'ar' hobbit.txt

- Obtener todas las palabras que tengan exactamente una letra "a", entre 3 y 7 letras y no empiecen con una a

  $ grep -E –color '<[^a]{2,7}a[^a]>' hobbit.txt

- Obtener todas las palabras que empiecen con h y terminen con s

  $ grep -E –color 'bh[A-Za-z]*sb' hobbit.txt