

<b>SEPTIMA SEMANA: "Operadores de cadenas I"</b> .....	<b>1</b>
<b>Miembros del grupo de estudios</b> .....	<b>1</b>
<b>Materiales de estudio</b> .....	<b>1</b>
<b>Algunos criterios</b> .....	<b>2</b>
TEMA 4 - Programación Básica del Shell .....	4
3.- Operadores de cadenas. ....	4
3.1.- Operadores de sustitución.....	4

---

## SEPTIMA SEMANA: "Operadores de cadenas I"

### Miembros del grupo de estudios

María, Fabricio, Diego --> **Argentina**

Jorge, Asterix --> **Chile**

Alejandro --> **Brasil**

Gerardo --> **España**

Miguel --> **Venezuela**

Con la reciente incorporación de:

José --> ¿?

Reinaldo --> **España**

---



### Materiales de estudio

*(Si tienen alguno más para recomendar, lo agregan)*

[Apuntes\\_bash\\_0.pdf](#) / [Apuntes\\_bash\\_1.pdf](#) / [bash.pdf](#) / [Comandos.pdf](#)

[\(GRUPO BASH\) PRIMERA SEMANA "COMODINES"](#)

[\(GRUPO BASH\) SEGUNDA SEMANA "REDIRECCIONES Y PIPES"](#)

\*\*\* [\(GRUPO BASH\) TERCERA SEMANA "CARACTERES ESPECIALES Y ENTRECORNILLADO"](#) \*\*\* **NUEVO!!!**

\*\*\* [\(GRUPO BASH\) CUARTA SEMANA "COMBINACIONES DE TECLAS"](#) \*\*\* **NUEVO!!!**

\*\*\* [\(GRUPO BASH\) QUINTA SEMANA "PARÁMETROS POSICIONALES. VARIABLES LOCALES Y GLOBALES"](#) \*\*\* **NUEVO!!!**

[Ejemplos de scripts - Ubuntu Forum](#)

[Un comando cada día - Ubuntu Forum](#)

[Bash scripting de supervivencia](#)

[Taller de programación shell](#)

[Shell Scripting](#)

[Google Docs](#)

[Redireccionamiento de Entrada-Salida](#)






[Redireccionamiento y tuberías](#)

[Combinaciones de teclas](#)

---

## Algunos criterios

- Podemos reutilizar los iconos para señalar algunas cuestiones (copiar y pegar).

	Referencia al material de estudio
	Desafío
	Script
	Atención!
	Mi Reino por un caballo!!!

	Una duda...
	Premio "Rex Tux Scripting"
	Consejo
	Cuidado!!!
	Comando

- Delimitar las intervenciones con barras horizontales.
- Firmar con el nombre resaltado con color.

María, Fabricio, Diego, Jorge, Alejandro, Asterix, Gerardo, Miguel, José, Reinaldo

- Fuente `Courier New` para los scripts y comandos. Aplicar sangrado de párrafo (ahora está en Formato-->Estilo de Párrafo).
- Si no visualizan correctamente las fuentes verdana y courier, instalen los paquetes:

```
sudo aptitude install msttcorefonts ttf-mscorefonts-installer
```

- **Poner onda con el formato** para facilitar la legibilidad del documento y el trabajo de publicación posterior.



## **TEMA 4 - Programación Básica del Shell**

### **3.- Operadores de cadenas. (página 55 )**

#### **3.1.- Operadores de sustitución.**

---

### **Sinopsis:**

"Los operadores de cadena nos permiten manipular cadenas sin necesidad de escribir complicadas rutinas de procesamiento de texto. En particular los operadores de cadena nos permiten realizar las siguientes operaciones:

- *Asegurarnos de que una variable existe (que está definida y que no es nula).*
- *Asignar valores por defecto a las variables.*
- *Tratar errores debidos a que una variable no tiene un valor asignado.*
- *Coger o eliminar partes de la cadena que cumplen un patrón.*

*Vamos a empezar viendo los operadores de sustitución, para luego ver los de búsqueda de cadenas."*

---

Esta parece ser una semana con bastante trabajo práctico. Una buena semana para aprovecharla con desafíos y escritura de scripts.

**Alejandro**

---

Tengo un problema con la ejecución del script del manual. He realizado una lista con nombres de personas y deudas, y lo he llamado como en el manual clientes. Al ejecutar en consola lo que aparece en el manual (pág 56).

```
sort -nr $1 | head -${2:-5}
```

El prompt se devuelve y se queda esperando al parecer que ingrese datos. Es decir no me toma el archivo clientes. Pero al ingresar datos, después cumple con lo solicitado.

## Grupo de estudios: Programación Bash Script

Séptima semana: del 19 al 25 de noviembre de 2009

---

```
$ sort -nr $1 | head -${2:-5}           (ingreso los datos)
45340 José Carlos Martínez
24520 Mari Carmen Gutierrez
450   Luis Garcia Santos
44    Marcos Vallido Grandes
500   Carlos de la Fuente Lopez
350   Sergio Miranda
120   Jorge Valdés
10    Mauricio Campusano
958   Luis Lorca
3564  Estela Nuñez
23456 Arian Martin
7734  Carolina Gonzalez
4500  Fredy Flores           (Finalizo con Ctrl+D, el espacio a continuación lo
agregué yo para que vean la salida)
```

```
45340 José Carlos Martínez
24520 Mari Carmen Gutierrez
23456 Arian Martin
7734  Carolina Gonzalez
4500  Fredy Flores
```

Ahora al reemplazar \$1 por el nombre del fichero pasa lo siguiente:

```
$ sort -nr clientes | head -${2:-5}     Me devuelve:
45340 José Carlos Martínez
24520 Mari Carmen Gutierrez
23456 Arian Martin
7734  Carolina Gonzalez
4500  Fredy Flores
```

Los script no hay forma de hacerlos funcionar, algo debo estar haciendo mal, o no entendiendo.

### Asterix

---

Mmmm... **Asterix**, supongo que puede ser esto:



*...si por ejemplo no pasamos el argumento \$1, con el nombre del fichero, se ejecutará el script así:*

```
sort -nr | head -$5
```

*Y sort se quedará esperando a que entren datos por su entrada estándar hasta que el usuario pulse Ctrl+D o Ctrl+C. Esto se debe a que, aunque controlamos que el segundo argumento no sea nulo, no controlamos el primero.*

- Yo creé un fichero "clientes" con "cat >clientes" y agregué la info en forma interactiva;

- Después creé el script "mejoresclientes" con gedit;
- Lo copié a mi carpeta "~/bin" (que está debidamente suscripta a PATH);
- Y lo ejecuté con el comando "~\$ mejoresclientes clientes 3"

Supongo, también, que para que tome el fichero "clientes", debemos ejecutar el script en el directorio donde está ese fichero, porque sino ¿cómo lo encuentra?

Y algo que noté es que en al ejemplo del script de la página 56 **le falta la primera línea:**

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

Yo se la agregué, y funciona. Pero se podría probar ejecutarlo así como está, con el comando "bash".

Alejandro

---

He estado estudiando las páginas del Manual y también me resultan un tanto engorrosas, por lo que he buscado otros instructivos sobre lo mismo y [ESTE](#) me ha solucionado varias dudas.

Especialmente la explicación del corte de cadenas la encuentro genial. Saludos.

Jorval

---

### Algunos experimentos para intentar entender...

- Primer experimento con `${var:-valor}`:

```
~$ echo $casa
~$ echo ${casa:-3}
3
~$ casa=5
~$ echo ${casa:-3}
5
```

¿Qué hice?

1. Intento imprimir una variable "casa" que no fue definida previamente. Resultado = nada.
2. Ahora, usando el primer caso de operador de sustitución. Como la variable no existe, es expandida con "valor" 3.
3. Asigno un valor "5" a la variable.
4. Imprimo nuevamente. Como la variable existe y no tiene valor nulo, es expandida con el valor que se le asignó "5".

**Conclusión:** en el ejemplo del manual se usa este operador de sustitución en lugar de usar directamente el parámetro posicional "\$2" porque, si no fuera así y no se le otorgase al script el argumento que tomará la variable "cuantos", listaré todos los clientes. Si yo tengo una lista de 1000 clientes, y quiero que me liste los primeros 10 mejores clientes pero, que en el caso de no colocar el segundo argumento, no liste más de 50, esa sería la forma de resolverlo.



Son cinco operadores de sustitución que presenta el libro. **Les propongo que cada uno intente explicar uno de ellos** "echando un poco de luz" sobre los ejemplos, que están bastante (coincido con Jorge)... turbios, digamos.

Pero no nos vamos a dejar ganar por unas pobres variables!!!

Alejandro

---

Hola, el otro día hablamos del "." que precedía al nombre de archivo de un script. Aquí tienen la explicación:

### 4.1. Lectura: Scripts para bash

Un script para bash es un archivo tipo texto, cuyas líneas tienen comandos que son ejecutados (interpretados) por bash. Para lograr que el intérprete de comandos interprete las líneas de un archivo puede:

- Ejecutar /bin/bash seguido del nombre del archivo (o redireccionar la entrada estándar para que provenga del archivo).
- Emplear el comando **source** seguido del nombre del archivo.
- Emplear el carácter '.' seguido de un espacio y el nombre del archivo.
- Agregar en la primera línea del archivo la cadena #!/bin/bash, dar permiso de ejecución al archivo y teclear el nombre del archivo desde el intérprete de comandos ---como si fuera un nuevo comando.

Jorval

---

Dos pequeños scripts que pueden ayudar a ver las diferencias entre `${var:?valor}` y `${var:=valor}`

#### Script nº 1:

```
#!/bin/bash
#el script operadores toma dos parámetros A y B

pa=${1:?'no fue asignado'}
pb=${2:?'no fue asignado'}
```

```
echo "el parámetro A es: $pa"  
echo "el parámetro B es: $pb"
```

El script nº 1 requiere dos parámetros. Si no recibe alguno de los dos dará un mensaje de error y detendrá el script.

### Script nº 2:

```
#!/bin/bash  
#el script operadores toma dos parámetros A y B  
  
pa=$1  
pa=${pa:='no fue asignado'}  
pb=$2  
pb=${pb:='no fue asignado'}  
  
echo "el parámetro A es: $pa"  
echo "el parámetro B es: $pb"
```

El script nº 2, también requiere dos parámetros. La diferencia es que, al no recibir alguno de los parámetros, asignará como valor a la variable el mensaje de error que definamos, y no detendrá el script.

### Alejandro

---

Gracias **Alejandro** por tu explicación, ahora me quedó super claro el como funciona el script.

Voy a tratar de explicar uno de los cinco operadores de sustitución: de los cinco he entendido hasta ahora el primero (`{var:-valor}`), y el que explico más abajo.

### **{var:?mensaje}**

Lo podemos usar para cuando queramos detectar errores producidos por una variable para cuando esta, no tenga un valor asignado; la sintaxis sería `{var:?mensaje}` que detecta si `var` no tiene valor asignado y produce un mensaje de error. Por ejemplo `{veces:?'Debe tener un valor asignado'}`

Si hay un error, imprimirá en pantalla lo siguiente:

`veces: Debe tener un valor asignado` si `veces` no tiene valor asignado.

En el ejemplo del manual; mi script es el siguiente ("mejoresclientes"):

```
#!/bin/bash  
# Script que saca los mejores clientes  
# Tiene la forma:  
#     mejoresclientes <fichero> [<cuantos>]
```

```
fichero_clientes=$1
fichero_clientes=${fichero_clientes:? 'debe ingresar el fichero'}
cuantos=$2
defecto=5
sort -nr $fichero_clientes | head -${cuantos:=defecto}
```

```
$ source mejoresclientes
bash: fichero_clientes: debe ingresar el fichero
```

Ahora de nuevo pero ingresando el fichero clientes:

```
$ source mejoresclientes clientes
45340 José Carlos Martinez
24520 Mari Carmen Gutierrez
23456 Arian Martin
7734 Carolina Gonzalez
4500 Fredy Flores
```

Al no haber error, el script nos devuelve esta vez los resultados, y no el mensaje de error.



Una duda que me acaba de nacer: Si una variable no la tenemos definida y usamos la forma `${var:-valor}`, nos retornará valor. ej.

```
$ echo ${casa2:-variable no definida}
variable no definida
```

y ahora usando la forma

```
$ echo ${casa2:? 'variable no definida'}
bash: casa2: variable no definida
```

¿Podríamos usar cualquiera de las dos formas?. A mi parecer la segunda forma tiene la ventaja de saber cual es la variable que no está definida, tal vez sea por esta ventaja que conviene usarla.

### Asterix

---

**Asterix:** creo que la diferencia sería la misma que en los ejemplos "script 1" y "script 2", es decir, en caso de usar `${var:-valor}` el script sigue su curso, porque más que un mensaje de error es un valor por defecto; en cambio, al usar `${var:? 'mensaje'}`, es un mensaje de error y el script se detiene (si no entendí mal). Entonces, según la situación, podría ser mejor uno que otro.

Respecto al tema de identificar de qué variable se trata, me parece que se puede subsanar colocando en el mensaje el nombre de la variable, así que no creo que ese sea el mayor problema.

```
$ echo ${casa2:-'variable casa2 no definida'}
```

En cuanto al "script 2" que había postado, creo que esta es una solución más razonable:

```
#!/bin/bash
#el script operadores toma dos parámetros A y B

pa=$1
pb=$2

echo "El valor del PRIMER parámetro es ${pa:-'nulo (ejecute el
script nuevamente)'}"
echo "El valor del SEGUNDO parámetro es ${pb:-'nulo (ejecute el
script nuevamente)'}"
```

Ahí, si el parámetro no fue pasado al script, las variables tendrán valor nulo, entonces se imprime el valor por defecto que es la cadena con el mensaje de error. De todas formas, lo lógico sería usar **`${var:?'mensaje'}`** y que el script se detenga, ya que la idea es que el usuario ejecute nuevamente el script pasándole los parámetros correspondientes.

Hasta ahora lo que hemos hecho son puras experiencias para tratar de entender la lógica que está detrás de un u otro operador de sustitución. Lo interesante ahora sería encontrar o pensar usos más "reales", menos "artificiales" ¿no?.

Alejandro

---

Hola estudiantes, mientras buscaba páginas relacionadas con los operadores de sustitución, he encontrado este excelente resumen que valdría la pena tener a mano siempre durante nuestras prácticas de escribir bash-scripts. Su dirección [AQUÍ](#) Espero nos sirva a todos para aclarar las dudas.

Jorval

---