

QUINTA SEMANA	1
Miembros del grupo de estudios	1
Materiales de estudio	1
Algunos criterios	2
TEMA 4 - Programación Básica del Shell	4
1.- Scripts y funciones	4
2.1.- Los parámetros posicionales.....	4
2.2.- Variables locales y globales	4
Sinopsis:	4
Resumen 1ra parte: ¿cómo ejecutar un script?	5
Funciones	6
1. La ruta absoluta.....	11
2. La ruta relativa	11
3. Modificar la variable PATH	11
Script Nº 1 (Asterix)	12
GRAN PREMIO TUX SCRIPTING para Asterix!!!	13
Cerrar una shell	15
(hiper-resumen) Parámetros posicionales. Variables locales y globales.....	16

QUINTA SEMANA

Miembros del grupo de estudios

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

Jorge, Asterix --> **Chile**

Alejandro --> **Brasil**

Gerardo --> **(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"](#)

[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](#) *** **NUEVO!!!****

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.
- No usar comentarios (en la publicación web no aparecen)
- Firmar con el nombre resaltado con color.

María, Fabricio, Diego, Jorge, Alejandro, Asterix, Gerardo

- 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

1.- Scripts y funciones

2.1.- Los parámetros posicionales

2.2.- Variables locales y globales

Sinopsis:

Este tema introduce los conceptos más básicos de la programación de scripts y funciones con Bash.

En este momento debería de ser capaz de manejarse con el terminal (el modo interactivo) con más soltura que antes de leer este tutorial. Ahora pretendemos empezar a entender los scripts que configuran su máquina, y a que sea capaz de crear sus propios scripts.

El tema supone que el lector conoce los conceptos básicos de programación en algún lenguaje estructurado (C, Pascal, Basic). Bash no usa los conceptos de programación orientada a objetos, pero si conoce este tipo de programación (p.e. Java) también le servirá aquí, ya que la programación estructurada es un subconjunto de la orientada a objetos.

Bash es un lenguaje muy críptico, con lo que sería recomendable que se fijara bien en la sintaxis, y que intentara hacer los ejercicios que proponemos en su ordenador con el fin de consolidar las ideas básicas que vamos a desarrollar en este tema.

Hola amigos, al leer la sinopsis de arriba, me preocupé porque si bien soy capaz de interactuar con el terminal con más soltura que antes, no tengo idea de lo que son los lenguajes estructurados de programación, pero haciendo clic [AQUI](#) encontré que no era tan peliagudo como creía. Saludos

Jorval

Sí, no creo que haya que preocuparse. Es lo que vinimos haciendo durante estas últimas cuatro semanas. Aprender la sintaxis y la lógica de un lenguaje estructurado. Es más de lo mismo... pero cada vez más difícil...

Alejandro

Como forma de arrancar, les propongo hacer un resumen comentado de los puntos que consideramos más importantes. Como el tema 4 es largo (y la actividad del grupo, últimamente, es corta) propongo que veamos solamente los puntos 1 y 2: "scripts y funciones" y "las variables del shell". Creo que la actividad de resumen, tratando de hacer más comprensible cada punto, ayuda a entrar en el tema -que, en principio, aparece bastante críptico-; y también sirve como control y autocontrol de la comprensión. La idea no sería simplemente Ctrl-C y Ctrl-V, sino tratar de explicar lo que uno consiguió entender del punto en cuestión e intentar hacerlo entendible para el resto.

Comienzo con algo y vemos...

Resumen 1ra parte: ¿cómo ejecutar un script?

Habría dos formas de ejecutar un script:

1.- como si fuese un programa ejecutable en C.

```
$ source nombre_script
```

2.- otorgándole permisos de ejecución.

```
$ chmod +x nombre_script
$ nombre_script
```

Siempre y cuando el escript esté alojado en los siguientes directorios:

```
alekde@kubuntu:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

Sino, se podrá ejecutar con:

```
$ ./nombre_script
```

Diferencias:

*"...Cuando ejecutamos un script con `source`, éste se ejecuta dentro del proceso del shell, mientras que si lo ejecutamos como un **fichero ejecutable** se ejecuta en un subshell. Esto implica que **sólo las variables de entorno exportadas** son conocidas por el nuevo subshell que ejecuta el programa."*

Podemos conocer la lista de **variables exportadas** usando el comando `env`.

Ampliación:

"Las variables de entorno que definimos dentro de una sesión de Bash no están disponibles para los subprocesos que lanza Bash a no ser que las exportemos con el comando `export`."

"...cuando ejecutamos un script se abre una nueva shell que es la encargada de ir interpretando las diferentes órdenes y de mantener el valor de las variables definidas. Esta subshell tiene como tiempo de vida el tiempo de ejecución del script. Además esta subshell hereda el valor de parte de las variables de entorno, pero en su propio espacio de memoria, por lo que las modificaciones de cualquier variable de la nueva shell no tendrá repercusión en la shell padre."

En una lista de correos aparece la siguiente consulta (como para ver las derivaciones que tiene el tema):

"Hola, compañeros:

Tengo una duda que me intriga bastante:

Parece ser que cuando ponemos un comando en la línea de comandos y pulsamos intro, incluso cuando este comando es un programa compilado, se hace un duplicado de la shell (una subshell) y dicho comando se ejecuta dentro de esta subshell.

Ahora bien, cuando escribimos varios comandos en la línea de comandos antes de pulsar intro:

--> Si el separador es punto y coma ";", ¿se abre una subshell para cada comando o se ejecutan ambos secuencialmente en la misma subshell?

--> ¿Qué ocurre si el separador es la barra vertical "|"? ¿Se abre una subshell para cada comando de la tubería?

--> Cuando el separador es ampersand "&" los comandos se ejecutan de forma concurrente. ¿También en este caso se abre una subshell para cada comando?

--> Cuando introducimos unos cuantos comandos entre paréntesis "()", se abre una subshell para todo el contenido de los paréntesis. ¿En este caso también se abre una subshell de esa subshell para cada uno de los comandos que están entre paréntesis?

□Gracias de antemano!

PRIMITIVO PAJARES."

No encontré la respuesta que le dieron a Primitivo Pajares.

¿Alguien se anima a seguir con el ítem funciones?

Alejandro

Funciones

(Resumen, páginas: 44 - 45)

Éstas, a diferencia de los scripts, se ejecutan dentro de la memoria del propio proceso de Bash, con lo que son más eficientes que ejecutar scripts aparte, pero tienen el inconveniente de que tienen que estar siempre cargadas en la memoria del proceso Bash para poder usarse.

Definir una función:

```
function nombre_funcion {  
  ...  
  sentencias bash  
  ...  
}
```

ó

```
nombre_funcion() {  
  ...  
  sentencias bash  
  ...  
}
```

Borrar una función:

```
unset -f nombre_funcion
```

Para ejecutar la función simplemente escribimos su nombre seguido de argumentos, como cuando ejecutamos un comando. Los argumentos actúan como parámetros de la función.

Si una función tiene el mismo nombre que un script o ejecutable, la función tiene preferencia.

Fabricio



Perdonen mi ignorancia, pero hasta ahora siempre he relacionado un script con abrir un editor de texto, ingresar una secuencia de comandos. Guardarlo con una extensión .sh, darle un permiso de ejecución y correrlo.

En cuanto a una función ¿están preestablecidas? ¿se ejecutan dentro (complementan) de un script? ¿hay que crearlas con un editor de texto? ¿cómo se ejecutan?

Asterix

Tras buscar un rato en san google respondo mi propia pregunta, copio en forma textual una definición que me agradó "Una función en Bash (denominada subrutina o procedimiento en otros lenguajes de programación) se podría definir como un script dentro de un script. Sirve para organizar un script en unidades lógicas de manera que sea más fácil mantenerlo y programarlo".

¿Qué es una función?

Esta rutina (o subrutina), es un grupo o bloque de instrucciones llamadas y definidas parecido a una variable.

Al igual que un shell script, pero reside en la memoria. Permite minimizar la cantidad de líneas en el programa, debido a que se repite menos código y además podemos usar nuestra amada recursividad.

Para utilizarlas simplemente hacemos un llamado a la función, por medio de su nombre, las funciones pueden estar en el mismo shell script o en uno aparte.

Si una función esta en un archivo aparte, debe importarse el script con el "." para que cargue las funciones en memoria, sino, no estarán disponibles.

Asterix

Hola, mi estudio hasta ahora:

- 1.- Creé un script con gedit y le puse por nombre: Hola_mundo
- 2.- Lo guardé en mi /home/jorval/Bash_Colaborativo
- 3.- Me coloqué en \$ cd Bash_Colaborativo
- 4.- Lo ejecuté primero con \$ source Hola_mundo. Esto cargó el archivo en la memoria de Bash y lo ejecutó.
- 5.- Le dí al fichero permiso de ejecución \$ chmod +x Hola_mundo para ejecutarlo por otros métodos.
- 6.- Ahora ejecuté el script de 3 maneras distintas:
 - a.- \$ bash Hola_mundo
 - b.- \$ sh Hola_mundo
 - c.- \$./Hola_mundo

De esta forma el script fue ejecutado en un subshell (Variables de entorno exportadas)

Ahora seguiré con las funciones.

Jorval

Continuando con el estudio:

Funciones: La explicación de **Asterix** ¿qué es una función? me aclaró las dudas que me dejó el Manual, ahora sí que se entiende, gracias.

- 1.- Más eficientes que los scripts.
- 2.- Antes: Inconveniente de que deben ser cargadas en la memoria de Bash. Ahora: ya no es inconveniente por la gran memoria de los computadores.
- 3.- Dos formatos a: function nombrefn {...comando bash...} o bien b: nombrefn {...comando bash...} - No hay diferencia
- 4.- Borrar una función: unset -f nombrefn
- 5.- Se almacenan como variables de entorno.
- 6.- Para ver que funciones tenemos definidas en una sesión : declare -f o bien declare -F
- 7.- Si una función tiene el mismo nombre que un script, la función tiene preferencia.

Orden de preferencia de los símbolos del bash:

- a.- Alias
- b.- Palabra clave como function, if, for
- c.- Funciones
- d.- Comandos internos como cd, type
- e.- Scripts y programas ejecutables según PATH

- Este orden, a veces, conviene cambiarlo, se puede con los comandos: `command - builtin - enable`
- Si queremos conocer la procedencia de un comando empleamos: `type`
- Conocer la lista de las variables exportadas : `env`
- Para exportarlas : `export`



1.- ¿Por qué algunos dicen que al script hay que ponerle la extensión `.sh`? Yo no se la puse y me funcionó sin problemas.

2.- De la definición de función de Asterix no entendí "Si una función esta en un archivo aparte, debe importarse el script con el `."` para que cargue las ..."

3.- Respecto al terminal, hay veces que la pantalla se llena y solo vemos el final de la información ¿cómo me muevo pantalla hacia arriba para ver desde el comienzo? no me resulta con ninguna tecla, sólo colocando al comando en cuestión `| less`

Jorval

1. Por el poco tiempo que vengo usando GNU/Linux me di cuenta de que las extensiones no son tan importantes. (Agarrá una imagen `.jpg`, quitale la extensión y luego abrila)
2. Tendríamos que ver algún ejemplo... pero seguramente debe ser alguna manera de "importar" una función para poder usarla en el script
3. ¿El comando `| less` no está justamente para eso?

Fabricio

¡Muchachos, excelentes contribuciones las de ustedes!!!!

Es cierto lo que dice **Fabricio**, las extensiones parece ser que no influyen demasiado en GNU/Linux. Más que nada sirven para dar un orden (en relación al humano y no a la máquina) Colocar a los scripts la extensión `sh` serviría para ubicar con mayor facilidad los scripts desparramados por el disco rígido.

Tengo la misma duda en relación al `."`
¿Tendrá algo que ver con el método de ejecución `./`?"

Para moverme hacia arriba y hacia abajo en la terminal uso **Shift+AvPág** o **Shift+RePág**.

¿Quién se anima a seguir el resumen de los parámetros posicionales?

Alejandro

Los parámetros posicionales

- 1.- Reciben los argumentos de un script y los parámetros de una función.
- 2.- Sus nombres son 0, 1, 2, 3, etc. y para acceder a ellos les colocamos

delante \$ ejemplo: \$0, \$1, \$2, \$3,, etc.

3.- Creamos con gedit el script:

```
#!/bin/bash
# Ejemplo de script que recibe parámetros y los imprime en pantalla
echo "El script $0"
echo "Recibe los argumentos $1 $2 $3 $4"
```

4.- Guardamos el script anterior con el nombre "recibe" (sin las comillas) y le damos permiso de ejecución

```
$ chmod +x recibe
```

5.- Si hago como dice el Manual no me funciona:

```
$ recibe hola adios
bash: recibe: orden no encontrada
```

6.- Entonces lo pongo como lo hice anteriormente y funciona:

```
$ ./recibe hola adios
El script ./recibe
```

```
Recibe los argumentos hola adios
```

7.- Definimos una función en modo interactivo:

```
$ function recibe {
> echo "La función $0"
> echo "Recibe los argumentos $1 $2 $3 $4"
> }
```

8.- La ejecutamos:

```
$ recibe buenos días Bash
La función bash
Recibe los argumentos buenos días Bash
```

- Hay que poner **mucha atención** a la colocación de las comillas (") - abrir comillas y cerrar las comillas.
- Respecto a las extensiones, en realidad nos ayudan a los usuarios, pero linux busca en el interior del archivo para determinar si es un directorio, un archivo común, etc.
- He probado shift+AvPág y el otro, pero no me funcionan.

Ahora iremos por las variables locales y globales.

Jorval

Bueno, te me has adelantado por un pelo **Jorval** en los parámetros posicionales, así es que haré otro aporte (creo).

Para aclarar un poco la ejecución de un script añado lo siguiente:

Ejecución de un script

Para poder ejecutar un script o un programa en línea de comandos existen varias posibilidades:

- - 1. La ruta absoluta
 - 2. La ruta relativa
 - 3. Modificar la variable PATH

En primer lugar **convierte** el script en ejecutable

Ejemplo:

```
chmod +x /home/carlos-vialfa/mis_scripts/script.sh
chmod 0755 /home/carlos-vialfa/mis_scripts/script.sh
```

1. La ruta absoluta

Cualquiera que sea la ubicación donde te encuentres ingresa
`/home/carlos-vialfa/mis_scripts/script.sh`

2. La ruta relativa

En este caso hay que ir al directorio que contiene el ejecutable
`cd /home/carlos-vialfa/mis_scripts/`

Para ejecutar el script escribe
`./script.sh`

Si descendiste demasiado en el árbol de directorios debes utilizar `.` y `..`

- `.` - directorio actual
- `..` - directorio padre

Ejemplo:

El script se encuentra en `/home/carlos-vialfa/mis_scripts/`
Yo me encuentro en `/home/carlos-vialfa/bin/perl_scripts/`
Para ejecutar el script desde esta ubicación debo ingresar
`../../mis_scripts/script.sh`

3. Modificar la variable PATH

Para hacer esto vamos a agregar a la variable PATH la ruta que contiene el script
`export PATH=$PATH:/home/lami20j/mis_scripts`

Para ejecutar el script escribe:
`script.sh`

Por algún lado recuerdo haber leído, que los privilegios con que se ejecuten los script (superusuario, o usuario), dependen de la ruta donde se guarden; por ejemplo los script de usuarios comunes, basta con dejarlo en el directorio `/bin`, mientras que en el `/sbin`, están los ejecutables de root. En estos casos solo basta invocar el nombre del ejecutable, inclusive con TAB se autocompleta

He diseñado el siguiente script que me evita tener que digitar tanto en la consola, y que uso mucho en mi casa para conectar mi notebook con mi pc de escritorio en una red ad-hoc



Script N° 1 (Asterix)

```
#!/bin/bash

echo "apagando interfaz"
ifconfig wlanX down
echo "OK"
sleep 3
echo "configurando interfaz (modo, canal, key)"
iwconfig wlanX mode ad-hoc essid xxxxx channel xx key s:xxxxxxxxx
sleep 3
echo "configurando ip, máscara de subred"
ifconfig wlanx inet xxx.xxx.x.xx netmask xxx.xxx.xxx.x
sleep 3
echo "Levantando la interfaz"
ifconfig wlanx up
sleep 3
ifconfig wlanx
sleep 4
iwconfig
```

Y luego lo ejecuto en consola

```
#casa-ad-hoc.sh
```

Y me funciona de maravillas. Como tarea pronto le voy a agregar variables, para que se pueda configurar en cualquier pc (que se configure según parámetros que se soliciten, ej. dispositivo de red, modo de la red, canal, clave, etc.).

El script anterior lo guardé en /sbin, es decir ejecutables de root. Noten que las líneas de comandos no están con sudo, con los que en consola habría que anteponerlo antes de cada comando para que se ejecute la orden (salvo si abrimos un terminal con privilegios de root)

Asterix

Impresionante!!!

Vamos a revivir el:



GRAN PREMIO TUX SCRIPTING para Asterix!!!

Hacía rato que no le tocaba a nadie...

Pregunta: **Asterix** ¿aceptarías donar tu script -bajo licencia GPL, claro, jajaja- para que su mejoramiento por medio de variables y otros "chiches" pase a ser un proyecto grupal?



Me quedan **algunas dudas**:

Al exportar la variable PATH y asignarle un valor

```
export PATH=$PATH:/home/lami20j/mis_scripts
```

- ¿Eso no borra los valores previos de la variable PATH? ¿No deja de reconocer /bin /sbin, etc como directorios de la variable PATH? ¿Únicamente agrega un nuevo valor? Supongo que es eso, pero no entiendo bien la sintaxis, el "PATH=\$PATH:" del principio.
- En caso de ser así, si el directorio que estoy agregando tiene mis permisos de usuario, no los de root, cuando ejecuto un script que está ahí ¿Cómo funcionan los permisos?

Jorge, verifiqué el tema de Shift+AvPág, etc, y a mí me funciona ¿Qué versión de bash tienes tú? ¿En qué situaciones quieres utilizarlo?

Bueno, y ¿quién se anota para las variables globales y locales?

Alejandro

Alejandro: Mi versión de Bash es : 3.2.39(1)-release

Hice el comando \$ declare -f y solo puedo subir y bajar por la pantalla, pero no llegar al comienzo del resultado del comando.

Jorval

Alejandro no habría problemas en donar el script al grupo, y que se pueda mejorar. Con respecto a tu duda, es una muy buena duda, la verdad no la había pensado. En lo personal creo que solo agrega una nueva ruta, ya que el texto dice "Para hacer esto vamos a agregar a la variable PATH la ruta que contiene el script"

Variables Locales y Globales:

Por defecto los parámetros posicionales son locales al script o función y no se pueden acceder o modificar desde otra función. Por ejemplo en el siguiente listado vemos un script, que lo guardaremos en el fichero "saluda", que llama a una función para que salude al nombre que pasamos al script como argumento en \$1:

Ejemplo 1: Script que llama a una función para saludar:

```
function DiHola {  
echo "Hola $1"  
}
```

```
DiHola
```

Al ejecutarlo tenemos: `$ saluda Fernando Hola`

El argumento pasado en `$1` no a sido tomado por la función, eso es porque `$1` es local al script, y si queremos que lo reciba la función tendremos que pasárselo como muestra el siguiente listado.

Ejemplo 2: Script que llama a una función para saludar

```
function DiHola {  
echo "Hola $1"  
}
```

```
DiHola $1
```

A diferencia de los parámetros posicionales, el resto de las variables que definimos en un script o función son iguales, una vez definidas en el script son accesibles (y modificables) desde cualquier función. En el siguiente ejemplo vemos como un script llamado "dondeestoy", en el que la variables "donde" está definida por el script y modificada por la función.

Ejemplo 3 :Ejemplo de variable global

```
function EstasAqui {  
donde='Dentro de la funcion'  
}
```

```
donde='En el script'  
echo $donde  
EstasAqui  
echo $donde
```

Al ejecutarlo obtenemos: `$ dondeestoy En el script Dentro de la funcion`

Si queremos que una variable no posicional sea local debemos de ponerla el modificador "local", el cual sólo se puede usar dentro de las funciones (no en los scripts). En el ejemplo siguiente ahora la función se crea su propia variable local "donde" y no modifica la global del script. Ejemplo:

Ejemplo 4: Ejemplo de variable global

```
function EstasAqui {  
local donde='Dentro de la funcion'  
}
```

```
donde='En el script'  
echo $donde  
EstasAqui  
echo $donde
```

Al ejecutarlo obtenemos: \$ dondeestoy En el script En el script

Las variables también se pueden definir dentro de una función. Por ejemplo, en el siguiente caso se muestra como definimos la variable global "quiensoy" dentro de la función EstasAqui () y la usamos más tarde desde el script. Ejemplo:

Ejemplo 5: Ejemplo de variable global

```
function EstasAqui {  
  local donde='Dentro de la funcion'  
  quiensoy=Fernando  
}  
  
donde='En el script'  
echo $donde  
EstasAqui  
echo $donde  
echo "Soy $quiensoy"
```

Al ejecutar el script obtenemos: \$ dondeestoy En el script En el script Soy Fernando

Asterix

Googleando encontré lo siguiente:

Cerrar una shell

Para cerrar una shell usamos la orden exit. Esta orden termina la ejecución de la shell y vuelve a la shell padre. Si ejecutamos la orden exit sobre la shell inicial que abrimos al entrar al sistema (llamada login shell) entonces terminamos la sesión de trabajo.

Corríjanme si me equivoco, pero esto quiere decir que la shell "padre", ¿es la que ejecuta el arranque del sistema, configuración del mismo, inicio del entorno gráfico?. Después de todo en algunas distribuciones linux, el entorno gráfico se arranca desde una línea de comando (startx). De ser así, significa que cualquier shell que abramos desde el "entorno gráfico" (KDE, Gnome, etc) ¿son subshell?. ¿y que de las distintas terminales a los que se accede con Ctrl+Alt+ F1....F6, son shell o subshell?

Asterix

Puede ser... igualmente lo importante es que **exit** te lleva siempre "un nivel más arriba", es decir si estás dentro de una función y ejecutas **exit** es como que le digas que salga de esa "subshell" y vaya a la shell que la contiene. Por ejemplo:

1)

```
#!/bin/bash
function DiHola {
    echo "Hola";
    echo "Chau"
}

DiHola
```

2)

```
#!/bin/bash
function DiHola {
    echo "Hola";
    exit;
    echo "Chau"
}

DiHola
```

El resultado del primer ejemplo es

```
Hola
Chau
```

El resultado del segundo ejemplo es

```
Hola
```

El comando **exit** es similar al **break** en php, lo que hace es salir, en este caso, de la función.

Fabricio

(hiper-resumen) Parámetros posicionales. Variables locales y globales

Las **VARIABLES** definidas en un script o en una función, pueden ser:

- **GLOBALES**, es decir, accesibles y modificables desde cualquier lugar del script (incluso por las funciones). Y Pueden ser definidas tanto dentro como fuera de las funciones.
- **LOCALES**, es decir, sólo tienen validez adentro de la función en la que fueron definidas.

- Los **PARÁMETROS POSICIONALES** son locales por defecto. Si fueron definidos para el script no afectarán a las funciones.
- Las **VARIABLES NO POSICIONALES** (todas las demás) son globales por defecto. Para convertirlas en locales se usa **LOCAL** antes de la asignación de valor a la variable

```
local variable='valor_de_la_variable'
```

¿Para qué podría servir esto? Por ejemplo, para que la variable en cuestión no afecte otros procesos fuera de una función X.

Los **PARÁMETROS POSICIONALES** (\$0; \$1; \$2; etc.) son variables que reciben argumentos en modo interactivo para ser usados por un script o por una función.

Posdata: Sería interesante si a alguien se le ocurre una propuesta de **autoevaluación** para esta quinta semana ¿no?

Alejandro
