EJECUCION SCRIPTS.

-tipos de shell sh, bash, ksh

-Primera linea del script

-Permisos antes de ejecutar un script.

-Sin permisos con "source"

- diferencia entre . ./script (no crea subshell) ./script y /bin/bash ./script
Establecer a otro valor la variable PATH en un script, denominado modifica_PATH.sh
ejecutarlo como . ./modifica_PATH.sh y como ./modifica_PATH.sh y ver el valor de la variable PATH

-comentarios con #

PERFILES (profile)
http://www.hospedajeydominios.com/mambo/documentacion-manual_linux-pagina-130.html
Existen tres ficheros en el directorio de un usuario que tienen un significado especial para el shell Bash. Estos ficheros permiten al usuario configurar el entorno de su cuenta automaticamente cuando entra en el sistema, cuando arranca un subshell o ejecutar comandos cuando sale del sistema.

Los nombres de estos ficheros son .bash_profile, .bashrc y .bash_logout. Si ninguno de estos ficheros existe en el directorio del usuario, /etc/profile es utilizado por el sistema como fichero de configuracion de bash.

.bash_profile es el el mas importante de los tres. Es leido y los comandos incluidos en el, ejecutados, cada vez que el usuario entra en el sistema. Cualquier cambio hecho en este fichero no tendra efecto hasta que salgamos y entremos en el sistema de nuevo. Una alternativa para no tener que salir del sistema es ejecutar el comando source .bash_source.

Bash permite dos sinonimos para este fichero, .bash_login (derivado del C shell) y .profile (derivado del Bourne y Korn shell). Si .bash_profile no existe, el sistema buscara primero .bash_login y luego .profile. Solamente uno de estos ficheros es leido, en el caso que existan simultaneamente.

.bashrc es leido cuando el usuario arranca un subshell, escribiendo por ejemplo bash en la linea de comandos. Esto nos permite ejecutar diferentes comandos para la entrada al sistema o para la ejecucion de un subshell. Si el usuario necesita los mismos comandos tanto a la entrada como en subshells, podemos incluir la siguiente linea en .bash_profile:

.bash_logout es el fichero leido por Bash, cuando salimos del sistema. Podemos definir, por ejemplo que se borren los ficheros temporales creados en nuestra ultima sesion o registrar el tiempo que hemos estado utilizando el sistema. Si .bash_logout no existe, ningun comando sera ejcutado a nuestra salida.

VARIABLES
-Por defecto, alfanumericas y globales
-variables locales con palabra reservada "local"
- establecer una variable variable=valor
    no dejar ningun espacio entre el igual y el nombre-valor de la variable.
- unset para limpiar una variable.
- mostrar una variable, recomendado entre llaves echo ${variable}
- Concatenacion con "+"
- Diferencia entre $pwd $(pwd) $(3+4) $((3+4))
    $(pwd) = `pwd` con tilde invertida.
    $((3+4)) es equivalente a let, con let no se utiliza el $, ejemplo let SUMA=NUMERO+5
    $expr 4 + 5 $expr 4 * 5
    Basic Calculator (bc) x=`echo 23*34 |bc`

-NOMBRE_FICHERO="red"$(date +%d%m%y)".conf";cp /etc/network/interfaces $NOMBRE_FICHERO
-lectura de variables read -p "Mensaje de pregunta" -n 3 <- tres caracteres.

OPERACIONES ARITMETICAS.

SUMA	+
RESTA	-
MULTIPLICACION	*
DIVISION	/
MODULO	%

EJECUCION VARIOS COMANDOS

- con ; comando1;comando2

$orden1 && orden2 -> ejecuta la orden que le precede y, si devuelve 0(éxito), entonces (y sólo entonces) ejecuta la segunda orden

$orden1 || orden 2 -> El operador || (OR lógico) ejecuta la primera orden y, si ésta devuelve una salida no nula (estado falso), entonces (y sólo entonces) ejecuta la segunda orden.

FUNCIONES

Definición:

```
function nombre_función {
        líneas de la función
}
```

variables locales (solo en bash y ksh)

```
function saludo {
        local NOMBRE="jose"
        echo "Hola $NOMBRE"
}
```

ESTRUCTURAS CONDICIONALES IF

```
if [ expresión1 ]; then
        realizar si expresión1 es verdadera
elif [ expresión2 ]; then
        realizar si expresión1 es falsa, pero es verdadera expresión2
elif [ expresión3 ]; then
        realizar si exp1 y exp2 son falsas, pero es verdadera expresión3
else
        realizar si todas las expresiones anteriores son falsas
fi
```

-comparacion de cadenas.
-comparacion alfanumerica
-test (entre corchetes) $if test -f fich; $if [-f fich] ;

Test Ficheros

[-f fichero] : cierto si el fichero existe
[-x fichero] : cierto si el fichero existe y es ejecutable
[-r fichero] : cierto si el fichero existe y se puede leer
[-w fichero] : cierto si el fichero existe y se puede escribir
[-d directorio] : cierto si el directorio existe
[-s fichero] : cierto si el fichero existe y no esta vacío

Operaciones sobre cadenas:

[cadena] : cierto si no es la cadena vacía
[-z cadena] : cierto si la longitud de la cadena es cero
[-n cadena] : cierto si la longitud de la cadena no es cero
[cadena1 = cadena2] : cierto si las cadenas son idénticas

[cadena1 != cadena2]: cierto si las cadenas son diferentes

Operaciones sobre números (enteros):
[num1 -eq num2]: cierto si los dos números son iguales
[num1 -ne num2]: cierto si los dos números son distintos
[num1 -gt num2]: cierto si num1 mayor que num2
[num1 -ge num2]: cierto si num1 mayor o igual que num2
[num1 -lt num2]: cierto si num1 menor que num2
[num1 -le num2]: cierto si num1 menor o igual que num2

Combinación de expresiones lógicas:
! (NOT), -a (AND) y -o (OR).
Ejemplo
```
if [ ! -f $f1 -a -f $f2 ]
```

ESTRUCTURAS CONDICIONALES CASE
```
case VARIABLE in
        valor1)
        Se ejecuta si VARIABLE tiene el valor1
        ;;
        valor2)
        Se ejecuta si VARIABLE tiene el valor2
        ;;
        *)
        Se ejecuta si VARIABLE no tiene ninguno de los valores anteriores
        ;;
        esac
```

Los ";;" hacen de break.

ESTRUCTURAS ITERATIVAS FOR
```
for variable in conjunto; do
        estas líneas se repiten una vez por cada elemento del conjunto,
        y variable va tomando los valores del conjunto
done
```

Ejemplo1
```
#!/bin/bash
for dia in lunes martes miércoles jueves viernes sabado domingo; do
        echo el dia de la semana procesado es $dia
done
```

Ejemplo2
```
#!/bin/bash
for programa in $( find ~ iname “*sh” 2> /dev/null ); do
        echo “Uno de mis scripts :” $programa
done
```

-steps Ejemplo:
```
for I in $( seq 1 3 20 ); do
        echo “Número vale :” $I
done
```

-break, break n, continue, continue n y exit n.

ESTRUCTURAS ITERATIVAS WHILE UNTIL

```
while [ expresión ]; do
        estas líneas se repiten MIENTRAS la expresión sea verdadera
done
```

Ejemplo:
```
while [ $NUMERO -ne 0 ]; do
        instrucciones
done
```

```
until [ expresión ]; do
        estas líneas se repiten HASTA que la expresión sea verdadera
done
```
Ejemplo:
```
until [ $NUMERO -eq 0 ]; do
        instrucciones
done
```

ESTRUCTURA SELECT
```
#!/bin/bash
select OPCION in Chiste Refrán Proverbio Salir; do
        if [ $OPCION = “Chiste” ]; then # quedaría más mono con case claro
        echo “Van dos por la calle y se cae el de en medio”
        elif [ $OPCION = “Refrán” ]; then
        echo “Quien cría cuervos tendrá muchos”
        elif [ $OPCION = “Proverbio” ]; then
        echo “Ten cerca a tus amigos, y mucho mas cerca a tus enemigos”
        else
        # Ha escogido Salir
        break
        fi
done
```

PARAMETROS

$1 Devuelve el 1º parámetro pasado al script o función al ser llamado.
$2 Devuelve el 2º parámetro.
$3 Devuelve el 3º parámetro. (Podemos usar hasta $9).
${10} también es válido.
$* Devuelve todos los parámetros separados por espacio.
$# Devuelve el número de parámetros que se han pasado.
$0 Devuelve el parámetro 0, es decir, el nombre del script o de la función.

VALORES DEVUELTOS:
$?

DEPURACION DE SCRIPTS

set -x Aparece la linea que se está ejecutando en la propia ejecución
set -v Parecido

CAPTURA DE SEÑALES
- trap "echo 'señal captada, continuo la ejecucion del guion' " 1 2 15
Captura las señales CTRL-X (1), CTRL-C (2) o KILL (15).

Ampliación:
http://www.linuxtopia.org/online_books/advanced_bash_scripting_guide/index.html