jue jun 10 00:23:34 CEST 2004

Bases de la programación para el Bash.

Como de introducción al Shell scripting sobre Bash.

< Por Xento Figal http://xinfo.sourceforge.net >

INTRODUCCION.

En este documento no se pretende enseñar a nadie a programar ni mucho menos, el objetivo es distinto quiero dar a entender una serie de conceptos sencillos sobre la programación en bash (algunos dirán que no es un lenguaje de programación, pero en Internet se encuentran softwares complejos, gestión de bancos de bases de datos, programas para crear diagramas, y un sin fin de aplicaciones, que cuanto menos, son fáciles de escribir), estos conceptos son muy sencillos, básicamente se explicaran dos tipos de aplicaciones, una directamente en la linea de comandos (shell) y otra en scripts ejecutables por la shell.

Vamos que sirve para los que aun no se han enterado de lo que se puede hacer con el shell :)
La manera de escribirlo intenta ser fiel a la primera versión (año 2002, unos 5kb creo recordar), en esta ultima versión ya no es tan fácil acogerse al original, aquí ya empezamos a tratar temas mas complejos, pero mantengo la idea de que tenga una continuidad de lectura, a modo de noticia en uno de los numerosos blogs que pueblan Internet, de esta manera se pretende que los conceptos se asimilen de manera progresiva, no voy a explicar el uso de condiciones antes de explicar que es una variable, sus tipos y la manera de usarlas, de la misma manera no me quiero quedar con el ejemplo útil para la comprensión pero no para la ejecución, de esta manera he incluido ejemplos prácticos en casi todas las secciones.

Por supuesto que el índice esta para algo, puedes consultar alguna duda y las secciones están pensadas para poder ser visitadas de manera única, pero la idea principal es para aquellos que nunca han tocado un script en bash, que se hagan una idea y puedan crear sus propios scripts, además se incluyen aspectos mas avanzados, como el uso de sockets o bash en otros sistemas.

inicio

LO PRIMERO QUE DEBEMOS SABER.

El lenguaje bash es como cualquier otro, se puede hacer lo extensible que queramos, pero principalmente nos centraremos en aplicaciones muy concretas:

-El prompt se puede modificar al gusto del usuario.

-Crear scripts que nos faciliten el uso diario de nuestra maquina linux.

-Ser una magnifica manera de acercarse a la programación y entender los fundamentos de esta.

inicio

NUESTRA LINEA DE COMANDOS

La propia linea de nuestra shell es ya en si un script en bash. (Basicamente, un script para la shell empieza por #!/bin/bash o la ruta donde se encuentre el ejecutable) Se pueden ejecutar scripts directamente usando sh nombre_del_script.sh o el clásico ./)

Una linea de comandos estandart en cualquier distro suele ser esta:
[usuario@host /]#

Bien esto esta definido en el archivo /etc/bashrc generalmente en la linea ps$=1

Esta es la linea normal para un prompt "normal": PS1="[\u@\h \W]\\$ " lo
que nos muestra
[usuario@host /]

Esto se puede cambiar a nuestro antojo puesto que podemos cambiar la linea del prompt como mas nos guste, en mi caso le añadí un reloj, le cambie los colores y añadi la entrada completa de los directorios.

Esta es la linea :

PS1="\[\033[0;36m\][\$(date+%H%M)]\[\033[0m\]\[\033[0;33m\][\u@\H\W]\\:\[\033[0m\] "

Aqui usamos códigos de color, [0;36m] y [0;33m] y las opciones propias del shell, \u \H \W mas una función ($date) y argumentos, la hora %H y los minutos %M.

El uso de de lineas \ (barra invertida) nos sirve para separar unas cosas de otras, o en el caso de opciones internas (\n \H etc...) para determinarlas y para volver al color original usamos \033.

Si esto no lo tienes nada claro no te preocupes echarle un vistazo a los códigos de color y opciones para el bash en este documento.

Observese que los espacios entre caracteres son definibles por el usuario.

Tambien se pueden hacer cambios no permanentes, simplemente metiendo argumentos en la linea de la shell esto esta explicado en otro articulo, la utilidad de esto esta fuera de toda duda.

Pero sobre todo, cabe destacar la utilidad de incluir scripts en bash en la shell, y o bien hacerlos fijos o no, para incluir scripts lo único que debemos hacer es añadirlo en el bashrc, o bien hacer un script que nos modifique la shell al ejecutarlo, y que luego podamos terminar el proceso y volver a la shell que tengamos definida.

inicio

CUADROS DE COLORES

Negro 0;30	Gris oscuro 1;30
Azul 0;34	Azul claro 1;34
Verde 0;32	Verde claro 1;32
Cyan 0;36	Cyan claro 1;36
Rojo 0;31	Rojo claro 1;31

Purpura 0;35	Purpura claro 1;35
Marron 0;33	Amarillo 1;33
Gris claro 0;37	Blanco 1;37

inicio

PRINCIPALES OPCIONES GENERICAS

\a caracter de campana ASCII (07)

\d la fecha en formato odia p.ej., mar may 26)

\e caracter de escape ASCII (033)

\h el nombre del host hasta el primer .

\H el nombre del la maquina completo (FQDN)

\n caracter de nueva linea

\r retorno de carro

\s el nombre del shell, el nombre base de $0 (el fragmento que sigue a la ultima barra)

\t la hora actual en formato 24-horas HH:MM:SS

\T la hora actual en formato 12-horas HH:MM:SS

\@ la hora actual en formato 12-horas AM/PM

\u el nombre de usuario del usuario actual

\v la version de bash (p.ej., 2.0)

\V la version del paquete del bash, version + patch-level (p.ej., 2.00.0)

\w el directorio actual de trabajo

\W el nombre base del directorio actual de trabajo

\! el numero del comando actual en el historico

el numero de comando del comando actual

\$ si el UID efectivo es 0, un #; en otro caso, $

\nnn el caracter correspondiente al numero en octal nnn

\\ una contrabarra

\[inicio de una secuencia de caracteres no imprimibles que pueden usarse para incrustar una secuencia de control del terminal en el prompt.

\] fin de una secuencia de caracteres no imprimibles

inicio

DOCUMENTACION BASICA

Hay unos cuantos documentos muy buenos sobre la programación en bash:

[http://www.tldp.org/LDP/abs/html/]
Ultima version del advanced bash scripting, sencillamente todo sale de este magnifico "libro" con una cantidad que no es normal de ejemplos, si buscas la madre de todos los how-to, comos, etc, etc sobre los scripts para el bash esta es la pagina.

[http://org.laol.net/lamug/beforever/bashtut.htm]
En ingles pero muy completo, una revisión antigua del anterior documento, una cantidad increíble de ejemplos, usando awk, perl, y todo lo que me dejo por poner (que es mucho), opciones y demás digamos que imprescindible.

[http://www.shelldorado.com] Excelente pagina dodne encontrar miles(no escribo en sentido figurado) de ejemplos, documentacion, articulos y muchisimo material.

[http://es.tldp.org/COMO-INSFLUG/COMOs/Bash-Prog-Intro-COMO/]
Uno de los únicos how-to completos en ESPAÑOL (me ha servido para las tablas de color, el bucle for-C, y las tablas de opciones genéricas).

[http://cs.bilgi.edu.tr/pages/courses/year_1/comp_151/examples/bash/]
Ejemplos muy prácticos y explicados.

[http://gyulai.freeyellow.com/en/linux/bash.html]
Muy bueno en ingles.

[http://clri6f.gsi.de/gnu/bash-2.01/examples/scripts.noah/]
Scripts explicados muy pero muy bien.

[http://www.linups.org/modules/documentacion/gdialog/]
Buen how-to de dialog,m cdialog y gdialog en ESPAÑOL.

[http://www.linuxfocus.org/Castellano/November2002/article267.shtml]
Una explicación con sencillos pero claros ejemplos, con capturas de pantalla y lo que nos tiene acostumbrados linuxfocus, sobre Xdialog. y en ESPAÑOL

[http://www.terra.es/personal/garzones/shell-scripts-micro-como.html]
Un micro-como en ESPAÑOL sobre la introducción del bash-scripting, tiene su sitio por que cada uno explicamos las cosas de una manera, esta español y tal vez hay cosas que aquí puedas entender y en otros sitios no (este incluido).

[http://ulysses.gulic.org/]
Paginas del grupo de usuarios de linux de gran canarias, con un software interesante, un programa de contabilidad domestica echo en bash con dialog y posibilidad de usar Xdialog en ESPAÑOL.

[http://igor2.mcomp.hu/gpl/tetris.sh/]
Una version del popular juego tetris para el bash, XD vale la pena verlo en acción.

inicio

EL EJEMPLO MAS SENCILLO

Bueno como bien se explica antes, las utilidades de la shell son infinitas, a continuación voy a explicar los scripts mas fáciles para el shell.

Ejemplo 1:

-Problema:
En un menu tipo blackbox, wmaker, fluxbox, ice,etc,,, queremos incluir una orden, esta orden es una llamada al terminal con argumentos, pero que estos argumentos definen un proceso o programa que termina y no requiere intervención del usuario, mas sencillo que esto pongo un ejemplo:

llamamos a un terminal para que nos muestre información sobre particiones, conexiones memoria, etc,,, como es lógico nos estamos

refiriendo al /proc/blah pero vamos que podríamos hacerlo llamando a cualquier archivo, bueno la orden en si:
xterm -e cat /proc/partitions

Cuando ejecutemos esto en un menu de los anteriores, resultara que cuando termine de mostrar la información el terminal se cierra, esto un problema directamente de la bash, bueno no es un problema, mas bien es necesario y así esta definido, imaginaros que debemos interrumpir siempre que hagamos un ls , en fin imaginaros ...

El problema se presenta en que si añadimos la orden | less al final del argumento, se queda igual, esto pasa por que un comando cuando termina, tiende a "automatarse".
Es importante este concepto, puesto que no tenerlo en cuenta nos puede dar dolores de cabeza en los errores de algunos scripts.

-Solucion:
Script para bash que nos ejecute la orden y ya esta, así de sencillo.

Ejemplo de las lineas del menu: [exec] (info particiones) {Eterm -e /scripts-bash/particiones}

Aqui llamamos a ejecutar una orden: Eterm a Eterm le decimos que ejecute algo(-e), ese algo es un script para la bash que se encuentra en /bash-script/ y se llama particiones, bueno ya se que esto es obio, pero por si acaso XD, bien claro esto, vamos a ver el script en cuestión:

```
#!/bin/bash
cat /proc/partitions
```

Esto es sencillo definimos el interprete en la primera linea, después el comando que queremos ejecutar.

```
#!/bin/bash
echo Informacion de las particiones
exec cat /proc/partitions | less
done
```

Como se puede ver, la manera de ejecutar algo en un script para el bash es extremadamente sencilla, usamos la primera linea para decirle donde esta el interprete después con echo mostramos información en pantalla como un printf, después usamos exec para ejecutar un comando, y acabamos con done para indicar que no tenemos nada mas que hacer.

Otro ejemplo útil, montar una partición de un disco duro extraible sin tener que meter todos los argumentos, y claro nos evitamos de escribir directamente el fstab, puesto que el disco unos dias esta y otros no.

```
#!bin/bash
echo montando partición
exec mount -t ext3 dev/hdc5 /mnt/otrolinux
done
```

Sin esto no lo sabias o no lo entiendes bien... me parece que poco podemos hacer, debería leer la documentación sobre linux que explica el uso básico del sistema, [http://www.linuxparatodos.com] es un buen sitio para comenzar a comprender que es linux y que puede hacer por ti (o tu por el).

inicio

USO DE VARIABLES

El script como podéis ver es de lo mas sencillo del mundo, pero también es útil al máximo, puesto que nos permite ver el resultado en pantalla y no cierra el terminal hasta que nosotros se lo decimos,usamos la orden less que mira tu por donde si funciona si lo aplicamos a

un script para el bash, como se puede imaginar las posibilidades son infinitas, y si usamos variables mucho mas, una manera simple de definir una variable:

```
#!/bin/bash
variable1=hola mundo
$variable1
```

El resultado es obio :
hola mundo

Aqui permitimos una entrada y la usamos como variable:

```
#!/bin/bash
echo Archivo a buscar:
read$ARCHIVO
find / '!' -type d -name "$ARCHIVO" -print0
```

Pues aquí lo tenéis, hemos introducido una variable en un comando, que a su vez contiene opciones, este script es bueno para buscar archivos en modo general, además se podría introducir una serie de variables, definidas por el usuario, que podemos colocar como argumentos definibles del comando, por ejemplo:

```
#!/bin/bash
echo Introduzca nombre del archivo y directorio, separados por un espacio:
read ARCHIVO DIRECTORIO
find /$DIRECTORIO'!' -type d -name "$ARCHIVO" -print0
```

Bueno como esto se ha "complicado" un poco mas, lo voy a explicar paso a paso:

-Con echo introducimos un texto que aparcera en pantalla.

-Con read dejamos que el usuario introduzca datos, y estos son las variables que usaremos para definir los parámetros en el comando.

-Despues introducimos las variables en el comando situándolas donde están los parámetros de este.

El usuario no debería introducir el operador / para llamar al directorio, esto puede ser útil o no depende, pero esto lo podemos cambiar a antojo, simplemente cambiando la variable , espero que quede claro.

-Fijate que en al orden original del comando, es decir directamente sobre la shell, las " son remplazadas por ' y estas no se representan cuando introducimos la variable.

Este es otro de esos detalles importantes si un script no hace lo que tu tenias pensado

Por supuesto, las variables también pueden ser de tipo numérico y se pueden ejecutar operaciones con los números lógicamente.

```
#!/bin/bash
numeroA=2
numeroB=5
frase="es correcto"

suma=$[$numeroa+$numerob]

multiplica=$[$numeroa*$numerob]

divide=$[$numeroa/$numerob]

resta=$[$numeroa-$numerob]

echo "2 + 5 = $suma $frase"
echo "2 * 5 = $multiplica $frase"
echo "2 / 5 = $divide $frase"
echo "2 - 5 = $resta $frase"
```

Ademas de estos ejemplos, también existe la variable de retorno, esta es la resultante de la ejecución del comando anterior, solo puede ser 1 o 0 , dependiendo de si el proceso finalizo con exito o no, esto es útil para determinar condiciones sin tener que calentarnos mas la cabeza, por ejemplo:

```
#!/bin/bash
ls directorio_real
echo "el valor de la variable es: $?"

echo " "
```

Si el directorio es real nos devolverá 0

Para dar opciones a un script hay que usar un modo distinto de pasarle argumentos, estas son las variables propiamente dichas.

```
#!/bin/bash
echo "\$0 es $0"
echo "\$1 es $1"
echo "$# parámetros"
```

Si ejecutamos este script nos daremos cuenta de que nos devuelve esto:

```
$0 es (nombredelscript)
$1 es
0 parámetros
```

Como se puede ver el aplicar una barra \ nos permite "escapar" en este caso "escapamos" de que en el script lea el parámetro $0 y $1 como texto normal y no como variables.
Como no hemos introducido nada las variables permanecen vacías, si hubiéramos ejecutado el script con argumentos nos devolvería otra cosa:

```
$./nombredelscript hola
```

Esto nos mostraría en pantalla:

```
$0 es nombredelscript
$1 es hola
1 parámetro
```

Creo que se entiende bien el uso de este tipo de variables.

Usando variables locales en funciones (ver sección funciones):

```
#!/bin/bash
variable=hola
function hola {
local variable=hola
echo $variable
}
```

Usando la salida de un comando como variable:

```
#!/bin/bash
var1=$(free | grep "buffers")
echo "$var1"
```

Este script nos mostraría el equivalente a ejecutar free | grep buffers en una consola.

inicio

CUADRO DE VARIABLES PREDEFINIDAS

$$ El numero identificador del proceso del shell.
$? Resultado de la ejecución del comando anterior.
$0 Nombre del script que se esta ejecutando
$1-$9 Primer a noveno argumento con los que se invoca.
$* Todos los argumentos como única palabra separador _.
$@ Array con todos los argumentos pero con posición.
$# Numero de argumentos recibidos.

inicio

USANDO IF FI

Logicamente nos haría falta usar condicionantes para poder dotar a los scripts de cierta "interactividad y toma de decisiones", para eso usaremos if fi.

Un programa para montar particiones usando diálogos, parámetros definidos por el usuario, colores y poco mas solo se usa if fi para ilustrarlo además read para que el usuario introduzca variables y se le ponen colores para hacerlo mas atractivo y se ilustra el -e, que nos sirve para ejecutar en la misma shell, recordáis el asunto del prompt.

Ejemplo básico de if fi:

```
#!/bin/bash
variable=verdadero
if [ "$variable" = "verdadero" ]; then
echo "la variable es correcta"
else
echo "la variable no es correcta"
fi
```

En este caso nos valemos de el operador = para indicar que si la variable1 es igual a la palabra verdadero se ejecute una acción, pensar que esa variable puede ser introducida por el usuario, como veremos en el siguiente script , mas adelante tenéis una lista de operadores.

Aqui tenéis un script que hace algo útil; montar particiones usando diálogos, parámetros definidos por el usuario, colores y poco mas solo se usa if fi para ilustrar, read para que el usuario introduzca variables y se le ponen colores para hacerlo mas atractivo y se ilustra el -e, que nos sirve para ejecutar en la misma shell, recordáis ; el asunto del prompt.

```
#!/bin/bash
echo -e "\033[4 32m MONTA PARTICIONES POR XENTO \033[0m"
echo -e Â¿Desea ayuda sobre el programa? "\033[34msi\033[0m" o
"\033[34mno\033[0m".
read A
if[ "$A" = "si" ]; then
echo Este es un script para facilitar el uso del comando mount
echo se le pedirán datos sobre los parámetros usados por el comando.
echo Si elije la opción mostrar parámetros de montaje entrar en el man del mismo.
echo -Primero: debería introducir el nombre de la partición.
echo -Segundo: debería introducir el directorio de montaje.
echo -Tercero: debería introducir el tipo de partición.
echo -Cuarto: debería definir los parámetros de montaje.
else
echo
```

```
fi
echo -e Â¿Desea ver la tabla de particiones? "\033[34msi\033[0m" o
"\033[34mno\033[0m".
read V0
if[ "$V0" = "si" ]; then
cat /proc/partitions
else
echo
fi
echo Introduzca el nombre de la partición:
read V1
echo Introduzca el directorio donde quiere montar la partición con la ruta completa
read V2
echo -e Â¿Desea ver los tipos soportados por su kernel?
"\033[34msi\033[0m" o "\033[34mno\033[0m".
read V00
if[ "$V00" = "si" ]; then
cat /proc/filesystems
else
echo
fi
echo Introduzca el tipo de partición:
read V3
echo -e Â¿Desea ver los parámetros de montaje "\033[31m(se muestra man)\033[0m"? "\033[34msi\033[0m" o "\033[34mno\033 0m".
read V01
if[ "$V01" = "si" ]; then
man mount
else
echo
fi
echo Introduzca los parámetros de montaje:
read V4
echo -e Â¿Son estos los datos correctos: mount -$V4 $V3 /dev/$V1 $V2? "\033[34msi\033[0m" o "\033[34mno\033[0m".
read R
if[ "$R" = "si" ]; then
exec mount -$V4 $V3 /dev/$V1 $V2
else
echo -e "\033[7 31m INTRODUZCA LOS DATOS DE NUEVO \033[0m"
echo
echo
exec ./nombredelscript
fi
echo Particion montada correctamente
```

En este script hemos usado if y fi para definir los parámetros del usuario,y usando variables, definimos que cosas se deben ejecutar, el script es realmente sencillo pero ilustra bien el uso de if fi.

inicio

TEST

Para usar if fi hemos usado un nuevo elemento, los corchetes que evalúan las condiciones, esto esta basado directamente en test, test nos sirve para evaluar condiciones.

Opciones básicas de test

-lt Menor que
-eq Igual que
-gt Mayor que
-le Menor o igual que

-ge Mayor o igual que
-ne No coinciden

-a
Operador lógico and
exp1 -a exp2 .
-o
Operador lógico or
exp1 -o exp2 .

Uso de test:

test 8 -lt 9

Usando variables en test:

variable1=5
variable2=3
test $variable1 -gt $variable2

Otra opción muy útil de uso de test:

[3 -lt 5]

Para que entender mejor test mirar este ejemplo:

#!/bin/bash
variable1=5
variable2=3
[$variable1 -lt $variable2]
echo $?

Aqui hemos evaluado dos variables para ver si la variable1 era menor que la variable2 y para ver el resultado hemos acudido a la variable de retorno, que nos mostrara 0 o 1.

inicio

USANDO CASE

Para evitar escribir tantos if fi (aunque no tiene nada de malo :-) podemos agrupar una serie de variables usando case, aquí esta el mismo ejemplo pero con case.

Case nos sera muy útil por ejemplo en este caso, lo que nos permite case es comprobar variables, el uso de case en este script es que el usuario al introducir una opción esta pasando un argumento que si coincide con una variable definida ejecutara una parte del programa.

Fijate en que se muestran 6 lineas de texto con una diferencia respecto a las anteriores las " " y el operador numérico entre [1....n] son las usadas para asociarlas a los números.
1)
...n)
Lo que sigue a estos números son las instrucciones que debería ejecutar el programa.

Dejamos un *) por si el usuario no introduce una opción valida (variable no definida).

Y después seguimos con la ejecución del script.

#!/bin/bash
echo -e "--"
echo -e "33[4 32m MONTA PARTICIONES POR XENTO 33[0m"

```
echo -e "------------------------------------------------"
echo -e "33[34m[1]33[0m" "Iniciar el programa"
echo -e "33[34m[2]33[0m" "Mostrar ayuda"
echo -e "33[34m[3]33[0m" "Mostrar tabla de particiones"
echo -e "33[34m[4]33[0m" "Mostrar tipos soportados por el kernel"
echo -e "33[34m[5]33[0m" "Mostrar man del comando mount"
echo -e "33[34m[6]33[0m" "Salir del programa"
echo "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"
echo "Introduzca un numero"
read tuseleccion
case $tuseleccion in
1) echo ;;
2) echo "Este es un script para facilitar el uso del comando mount se le
pedirán datos sobre los parámetros usados por el comando mount "
echo "-Primero: debería introducir el nombre de la partición "
echo "-Segundo: debería introducir el directorio de montaje"
echo "-Tercero: debería introducir el tipo de partición "
echo "-Cuarto: debería definir los parámetros de montaje" ;;
3) cat /proc/partitions ;;
4) cat /proc/filesystems ;;
5) man mount ;;
6) exit 0 ;;
*) echo "Por favor introduzca un numero de opción valido 1,2,3,4,5";
echo -e "Se continua con al ejecución normal del programa 33[7 31mcontrol + c para salir33[0m";;
esac
echo -e "33[4 32m Introduzca el nombre de la partición: 33[0m"

read V1
echo -e "33[4 32m Introduzca el directorio donde quiere montar la partición con la ruta completa 33[0m"
read V2
echo -e "33[4 32m Introduzca el tipo de partición: 33[0m"
read V3
echo -e "33[4 32m Introduzca los parámetros de montaje: 33[0m"
read V4
echo -e "33[4 31m Â¿Son estos los datos correctos: mount -$V4 $V3 /dev/$V1 $V2? 33[0m" "33[34msi33[0m" o "33[34mno33 0m".
read R
if [ "$R" = "si" ]; then
exec mount -$V4 $V3 /dev/$V1 $V2
echo Particion montada correctamente
else
echo -e "33[7 31m INTRODUZCA LOS DATOS DE NUEVO 33[0m"
echo
echo
exec ./montaparticiones
fi
```

Bueno esto nos da mas interactividad y nos ahorra lineas de código, es otra manera mas de como se pueden hacer las cosas

Hay que destacar llegados a este punto, que la interactividad del shell es impresionante.

inicio

USANDO SELECT

Por supuesto que el bash ofrece muchas mas opciones, una manera distinta es el uso de select.

```
#!/bin/bash
variables="montar salir"
select opt in $OPCIONES; do
if [ "$opt" = "montar" ]; then
echo "Introduzca el nombre de la partición"
read v1
echo "Introduzca directorio de montaje"
read v2
echo "introduzca tipo de archivos"
read v3
exec mount -t $v3 $v1 $v2
echo "partición montada"
exit
elif [ "$opt" = "salir" ]; then
exit
else
clear
echo "opción incorrecta"
fi
done
```

Como se puede ver este método es muy parecido a usar un bucle, pero puede ser útil, eso si nos imposibilita a voz de pronto mostrar las acciones asociadas a cada variable, por lo que se debería mostrar información en pantalla antes sobre estas.

inicio

BUCLES

Los bucles nos sirven básicamente para definir lo que queremos se haga si se cumple una condición, existen varios tipos de bucles, cada uno tiene mas o menos definida una acción.

El bucle for permite iterar sobre una serie de `palabras' contenidas dentro de una cadena.

El bucle while ejecuta un trozo de código si la expresión de control es verdadera, y solo se para cuando es falsa (o se encuentra una interrupción explicita dentro del código en ejecución).

El bucle until es casi idÃ©ntico al bucle loop, excepto en que el cÃ³digo se ejecuta mientras la expresiÃ³n de control se evalÃºe como falsa.

FOR

```
#!/bin/bash

for i in v1 v2 v3 v4; do
```

echo "Estas son las variables $i"

Este script nos mostraría en pantalla esto:

Estas son las variables v1 Estas son las variables v2 Estas son las variables v3 Estas son las variables v4

La explicación creo que queda clara.

WHILE

```
#!/bin/bash
MINUM=8
while [ 1 ]; do
echo "Introduzca un numero: "
read USER_NUM
if [ $USER_NUM -lt $MINUM ]; then
 echo "El numero introducido es menor que el mio"
echo " "
elif [ $USER_NUM -gt $MINUM ]; then
echo "El numero introducido es mayor que el mio"
echo " "
elif [ $USER_NUM -eq $MINUM ]; then echo "Acertaste: Mi numero era $MINUM"
break fi
done
```

Con este ejemplo lo que hemos echo es sencillo, evaluar una variable usando un bucle.Nos hemos valido de algunos operadores de test y de if fi.

BREAK

Ademas nótese que hemos terminado el bucle con break, esta es una magnifica manera de terminar nuestros bucles.

UNTIL

```
#!/bin/bashCONTADOR=20 until
[ $CONTADOR -lt 10 ]; do echo CONTADOR $CONTADOR
let CONTADOR-=1
done
```

Este ejemplo ilustra muy bien el uso de until, en este caso el bucle se reproduce con una condición que varia (let) cuando el bucle se ejecuta por lo que al final se cumple la condición y termina el bucle.

OTRO TIPO DE BUCLE

-Usando sintaxis tipo C.

```
((a = 1)) while (( a <= LIMIT )) do
echo -n "$a " ((a += 1)) # let "a+=1"
done
echo
```

-Otra manera de usar la sintaxis estilo C

```
#!/bin/bash
for i in `seq 1 10`;
do echo $i done
```

En la [18] de todos los how-to sobre bash scripting, hay un montón de ejemplos de como hacer bucles de maneras distintas.

inicio

FUNCIONES

Las funciones nos sirven para tener código dentro del código, es decir podremos ejecutar pequeños o no tan pequeño trozos de código como si se tratasen de scripts distintos, esto es útil sobre todo para definir variables, sobre todo si esas variables no poseen un valor fijo.

```
function hola {
echo "hola mama"
}
```

Esta función nos devolvería en pantalla: hola mama, para acceder a esta función solo hay que llamarla como si un comando mas se tratase.
Escribi un pequeÃ±o [20] sobre el prompt, en el explico que es y como usar una función, pero como este es otro documento, pues pondremos mas ejemplos.

```
#!/bin/bash
function memf {
exec free | grep Mem | awk '{print $4}' }

function memt {
exec free | grep Mem | awk '{print $3}' }

function cpu {
exec top -n1 | grep "CPU states" | awk '{print $3+$5"%"}' }

function date {
exec date | awk '{print $1"-"$3"-"$2" "$6 }' }

echo "Informacion del sistema"
memf
memtcpu
date
exit
```

Este script nos mostraría lo que las funciones contienen.

inicio

USO DE CARACTERES ESPECIALES (COMODINES VARIOS)

Los caracteres especiales, son considerados por el shell como parámetros especiales, de hay que nombres de variables, con estos no sean una buena idea.

```
Var*?=hola Mala eleccion
var1-0=hola Mala elección
var1_0=hola Buena opción
[ "$number" < 5 ] Mala elección
[ "$numero" -lt 5 ] Buena opción
```

-La expansión de un comando

La salida producida por un comando, substituye al propio comando.(comando) 'comando' producen la salida, no se ejecutaría el comando. 'exec emacs'

Mostraria en pantalla el texto exec emacs, si por el contrario simplemente no lo encerramos se ejecutaría emacs.

-La expansión aritmética.

El shell evalúa una expresión y muestra su resultado: ((1 + 1)) mostraría en pantalla 2

-La expansión de sufijos y prefijos.

Ejemplo útil y mas claro que perderme en explicaciones.

Echo hola{mundo,mama,pepe}adiós
Esto nos mostraría en pantalla:

hola mundo adiós
hola mama adiós
hola pepe adiós

-La expansión de nombres de fichero.

En cada palabra que aparezcan caracteres especiales * ? | el shell tratara esa palabra como un patrón, P.ejm:

ls * .rpm Esto devolvería todos los archivos con extensión .rpm

-Otras expansiones.

~ esto substituye todas las palabras por argumentos de un directorio de usuario.

~xento/ esto nos mandaría a al home del usuario xento. ~/ nos mandaría al directorio home del usuario que ejecute el script.

Var=hola {var} Esto nos mostraría el valor de la variable var, es decir nos mostraría en pantalla hola.

inicio

SCRIPTS BASH PARA X-WINDOW

Una manera de crear scripts que parecen software, y es mas con multitud de opciones es Xdialog.

Esta es una herramienta que substituye a dialog o cdialod y que usa gtk para crear ventanas de una manera muy simple, básicamente la sintaxis es la misma que para el bash. documentación y descarga, mirar en esta web[23] para ver las posibilidades de Xdialog

Ejemplo de una ventana:
xdialog --title "Este es el titulo" --msgbox "Este es el texto de la ventana" 0 0

Esto nos mostraría una pantalla en gtk con el texto:

Este es el texto de la ventana

Xdialog es un buen substituto de dialog o cdialog, y pose numerosas opciones típicas en cualquier lenguaje, crear entradas de texto, lista de botones, botones de confirmación, diálogos yes-no, opción de abrir archivo, texto editable, etc...

Un Ejemplo útil.

<--Mostramos la salida de un comando y la pasamos a un fichero-->
ps -aux | expand >> /tmp/textboxjobs.tmp.$$

<--después mostramos ese archivo en una caja-->
xdialog --title "Actuales procesos " --textbox "/tmp/textboxjobs.tmp.$$" 0 0

```
<--pedimos al usuario que introduzca una variable-->
V1=`xdialog --inputbox "introduzca el nombre de la tarea a terminar " 8 40 2>&1 1>/dev/tty`

<--la variable es 'V1', y usamos la salida a tty para poder guardarla-->
<--Usamos la variable para meterla en un comando directo al bash-->

exec killall -9 $V1

<--Caja de confirmación>

dialog --title " $V1 " --msgbox "La tarea $V1 a sido finalizada con exito" 0 0
```

Xdialog posee una buena cantidad de ejemplos, de todas las funciones que podemos hacer con el en la documentación que acompaña al paquete.

ZENITY AL DESNUDO

Zenity es un programa que nos permite usar ciertos argumentos para crear ventanas usando GTK.

El porque he decido extenderme mas con este que con otro, tiene una sencilla explicación; últimamente he creado algunas cosas con el, y puedo dar un ejemplo de estructura de trabajo con estos ejemplos, y así dejar mas claro el uso de estos programas (xdialog y kdialog) en general.

Posibilidades de zenity

Mostrar texto en una ventana (--text).

Podemos definir la altura y anchura de la ventana.

Podemos definir el icono (--error, --warning).

Tambien tenemos la posibilidad de indicar cual sera el icono de la ventana (--window-icon)

Ademas zenity nos permite usar listas de elementos y checkboxes con posibilidad de hacerlas editables, y configurar las cabeceras.

Calendario y selección de dias, meses, años y formato del calendario.

Dialogos tipo yes/no (--question).

Barras de progreso (--progress) con opciones distintas, auto cerrarse, indicador del tanto por ciento, texto del dialogo y enunciado.

Mostrar texto en una ventana (--filename) hacerlo editable o/y ocultarlo.

Tambien nos permite la selección de archivos (--file-selection).

Y usar una entrada de texto (--entry) con posibilidad de introducir un texto predeterminado, o mostrar el texto que se escribe como asteriscos.

Formas de trabajo con zenity.

Ejemplo básico:

```
#!/bin/bash
zenity --title "titulo de la ventana" --info --text "hola mundo"
```

La opción --info, puede ser otra (--question, --warning, --error)

Si usamos la opción --question el resultado nos devolverá 0 o 1 que se almacena en la variable $?, ver cuadro de variables predefinidas.

Ejemplo usando variables de retorno y diálogos tipo yes/no

```
#!/bin/bash
zenity --title "titulo de la ventana" --question --text "¿Aceptar?"
if [ "$?" = "0" ]; then
zenity --info --text "Has aceptado"
else
zenity --info --text "No has aceptado"
fi
```

Trabajando con fechas

```
#!/bin/bash
zenity --calendar
```

Este sencillo script nos muestra un calendario completo, donde podemos ir hacia delante o hacia atras en el.

Si pulsamos aceptar nos devuelve la fecha de la selección, esto es interesante por su posibilidad de aplicación

```
#!/bin/bash
var=$(zenity --calendar --text "Selecciona una fecha")
zenity --question --text "Has seleccionado $var"
```

En este ejemplo queda ilustrado como "aprovechar" esta opción para almacenarla como variable, simplemente uso un comando como variable.

En la creación de listas, se pueden usar los elementos de las listas de la siguiente manera, por ejemplo:

```
#!/bin/bash
v1=$(zenity --list --column "Tipos de opciones" "opcion1" "opcion2")
if [ "$v1" = "opcion1" ]; then
zenity --info --text "Opcion1"
elif [ "$v1" = "opcion2" ]; then
zenity --info --text "Opcion2"
else
fi
```

Como en cualquier script, zenity admite que le pasemos variables como opciones.

```
#!/bin/bash
var1="Introduzca un año"
var2=$(zenity --entry --hide-text --text "$var1")
var3=$(zenity --entry --text "Introduzca un mes")
var4=$(zenity --entry --text "Introduzca un odia")
zenity --calendar --year "$var2" --day "$var4" --month "$var3"
```

Un aspecto muy útil es el uso de las "herramientas" que zenity (o kdialog y xdialog) ponen a nuestra disposición para trabajar con archivos.

```
#!/bin/bash
zenity --text-info --filename "/etc/fstab"
```

Desde luego para poder ver mejor el uso de zenity es imprescindible leer bien la ayuda del comando así como su pagina del manual (man zenity, incluye algunos ejemplos útiles), también puede ver:

zenity --help

zenity --usage

En el navegador de ayuda de gnome (yelp) se puede consultar un manual con scripts de ejemplos y capturas de cada ejemplo, muy completo.

Puede acceder dicho manual usando zenity --about.

KDIALOG

Kdialog

Kdialog es una aplicación bastante completa que interactúa con kde.

Trabaja con Qt y nos permite crear scripts con verdadera potencia, siempre y cuando trabajemos con kde.

Kdialog es una aplicación potente, posee una serie de aspectos que lo hacen distinto que xdialog o que zenity.

He aquí la salida de kdialog --help-all.

Opciones genéricas:
--help Muestra ayuda sobre las opciones.
--help-qt Muestra opciones específicas de Qt.
--help-kde Muestra opciones específicas de KDE.
--help-all Muestra todas las opciones.
--author Muestra información del autor.
-v, --version Muestra información de la versión.
--license Muestra información de la licencia.
-- Fin de las opciones.

Opciones de Qt:
--display <displayname> Usa la pantalla 'displayname' del servidor X.
--session <sessionId> Restaura la aplicación para el Id. 'sessionId' dado.
--cmap Provoca que la aplicación instale un mapa de color privado en una pantalla de 8 bits.
--ncols <count> Limita el número de colores reservados en el cubo de color en una pantalla de 8 bits, si la aplicación usa la especificación QApplication::ManyColor.
--nograb Impide que Qt capture el ratón o el teclado.
--dograb Ejecutarlo con un depurador puede provocar un -nograb implícito; use -dograb para evitarlo.
--sync Cambia a modo síncrono para depurar.
--fn, --font <fontname> Define la fuente para la aplicación.
--bg, --background <color> Especifica el color predeterminado del fondo y una paleta de colores (sombreados claros y oscuros son calculados).
--fg, --foreground <color> Especifica el color predeterminado de primer plano.
--btn, --button <color> Especifica el color predeterminado de los botones.
--name <name> Especifica el nombre de la aplicación.
--title <title> Especifica el título de la aplicación.
--visual TrueColor Fuerza a la aplicación a usar una paleta de colores TrueColor en una pantalla de 8 bits.
--inputstyle <inputstyle> Especifica un estilo de entrada XIM (X Input Method).
Los valores posibles son: 'onthespot', 'overthespot', 'offthespot' y 'root'.
--im <XIM server> Especifica el servidor XIM.
--noxim Desactiva XIM.
--reverse replica el diseño completo de objetos visuales.

Opciones de KDE:
--caption <caption> Usa 'caption' como nombre en la barra de título.
--icon <icon> Usa 'icon' como icono de la aplicación.
--miniicon <icon> Usa 'icon' como icono en la barra de título.
--config <filename> Utilizar archivo alternativo de configuración.
--dcopserver <server> Usa el servidor DCOP especificado por 'server'.
--nocrashhandler Desactiva el manejador de fallos, para obtener volcados de memoria.
--waitforwm Espera a un administrador de ventanas compatible con WM_NET.
--style <style> Especifica el estilo del GUI de la aplicación.

--geometry <geometry> Especifica la geometría cliente del objeto visual (widget) principal

Opciones:
--yesno <text> Cuadro de texto de preguntas con botones de si/no
--yesnocancel <text> Cuadro de texto de mensajes con botones de si/no/cancelar
--warningyesno <text> Cuadro de aviso de mensajes con botones de si/no
--warningcontinuecancel <text> Cuadro de aviso de mensajes con botones de continuar/cancelar
--warningyesnocancel <text> Cuadro de aviso de mensajes con botones de si/no/cancelar
--sorry <text> Cuadro de mensaje de 'Lo sentimos'
--error <text> Cuadro de mensaje de 'error'
--msgbox <text> Cuadro de diálogo de mensaje
--inputbox <text> <init> Cuadro de diálogo de entrada
--password <text> Diálogo de contraseña
--textbox <file> [width] [height] Cuadro de diálogo de texto
--combobox <text> [tag item] [tag item] ... Diálogo de lista desplegable
--menu <text> [tag item] [tag item] ... Diálogo de menú
--checklist <text> [tag item status] ... Diálogo de lista de comprobación
--radiolist <text> [tag item status] ... Diálogo de lista de selecciones excluyentes
--getopenfilename [startDir] [filter] El diálogo de archivos abre un archivo existente
--getsavefilename [startDir] [filter] Diálogo de archivos para guardar un archivo
--getexistingdirectory [startDir] Diálogo de archivo para seleccionar un directorio existente
--getopenurl [startDir] [filter] El diálogo de archivos abre una URL existente
--getsaveurl [startDir] [filter] Diálogo de archivos para guardar una URL
--title <text> Título de diálogo
--separate-output Devolver elementos de lista en líneas separadas (para opción de comprobación)
--print-winid Muestra el winId de cada diálogo
--embed <winid> Hace el diálogo transitorio para una aplicación X especificada por winid
--dontagain <file:entry> Archivo de configuración y nombre de la opción para guardar el estado "no-mostrar/volver-a-preguntar"

Argumentos:
arg Argumentos - dependiente de la opción principal

Como se puede apreciar, kdialog nos proporciona un entrono mas "amigable" si usamos kde, de todas maneras también se puede usar sin este, pero lanzar todo lo que requiere kde para una simple aplicación con cuadros de dialogo,bajo mi punto de vista no es productivo.

Otra tema seria si buscamos alguna función mas espeficica.

Podemos tratar las variables igual que en zenity o xdialog.

kdialog nos permite unos cuadros de dialogo con otro tipo de funciones.

P.ejm:

```
#!/bin/bash
kdialog --warningcontinuecancel "Desea continuar"
kdialog --msgbox "$?"
```

Si aceptáramos nos devolvería un 0 en la variable de retorno ($?) si no directamente cerraría el programa, no salta a la siguiente instrucción.

P.ejm

```
#!/bin/bash
var1=$(kdialog --getopenurl / html)
mozilla $var1
```

En este ejemplo, usamos una ventana de selección de archivo partiendo del directorio raiz (/) y filtrando a los archivos que tengan la extensión html.

Despues el archivo lo visualizamos con mozilla.

En el capitulo de zenity explico mejor como poder usar las opciones de este tipo de programas usar variables, las listas, los checkboxes y

demás.

Como apunte, decir que si alguien se ve tentado a crear cosillas bajo kde, existen herramientas tales como kdevelop, pero si buscáis algo mas sencillo pero con posibilidades de crear esqueletos gráficos completos, tenéis el kommaneder es una aplicación que requiere quanta para funcionar, y trabaja con xml, es muy fácil de aprender a usarla y puedes crear cosas mas serias.

inicio

TUBERIAS Y REDIRECCION.

En estos ejemplos como en otros, hemos usado tuberías (|) y hemos redireccionado la salida a un archivo.

-**Tuberias**:
Las tuberías permiten utilizar la salida de un programa como la entrada de otro.

P.ejem:
netstat -an | grep -v "unix"

En este ejemplo usamos la salida del comando netstat para mandarla a grep que eliminara todas las lineas que este la palabra unix.

-**Redireccion**:

La redireccion puede parecer lo mismo pero lógicamente no lo es, la redireccion en lo que nos ocupa, nos permite usar la salida de un comando a un fichero o usar el buffer resultante de otra maneras.

Ejemplos:
comando > archivo esto hace que la salida de un comando se escriba en un archivo, ojo que borrara todo lo que contiene el fichero.

ls -al > lista_de_archivos.txt Esto mandaría la salida al fichero.

ls -al &> todo.txt Esto mandaría todo a un fichero: todo.txt

ls -al >> fichero esto añade la salida al final de fichero, sin borrar los datos anteriores, y si no existe, lo crea.

comando < archivo Esto hace que el comando use como argumentos archivo, muy útil para crear interfaces mas complejas.

En el [25] del insflug puede ver mas sobre redireccion.

inicio

Comandos útiles

En este punto del documento, nos vamos a centrar solo en dar a conocer unos comandos que nos pueden ayudar muchísimo a trabajar, solo se van a nombrar, citar su uso, explicar que hacen y en algunos casos se expondrán algunos ejemplos, es recomendable que se tengan unas nociones básicas sobre http://www.linuxfocus.org/Castellano/July1998/article53.html expresiones regulares, si bien no es imprescindible, es recomendable.

No se pretende dar una guia de uso de cada uno de estos, solamente se citan para que el lector, si desea profundizar en su uso busque

información al respecto.

-AWK
Awk es un lenguaje en si mismo, nos permite trabajar de con texto de una manera especial.

Basicamente (awk es extenso) el uso de “dia a dia” que le doy:

awk '/patrón { print }' archivo

Me busca un patrón en un fichero e imprime en pantalla.

Uname -a | awk '{print $14 $4}'

Esto nos devolvería: GNU/Linux 2.6.5-1.mdk

Si ejecutamos uname -a : Linux xento-local 2.6.5-1.mdk #1 Mon Apr 26 23:18:57 CEST 2004 i686 unknown unknown GNU/Linux

Con lo cual hemos mostrado solo los campos 13 y 4, que corresponden a GNU/Linux y 2.6.5-1.mkd

Awk es bastante extenso, http://www.linux.org.uy/uylug/cursos/awk/awk.html tenéis un manual en español bastante completo para entender las peculiaridades de este programa.

-SED
Es un completo editor de texto, eso si trabajando con comandos, lo que es de muchísima utilidad para trabajar con texto dentro nuestros scripts.

El uso de sed es extenso, pero también es muy versátil.

A continuación algunos comodines básicos de sed que suelo usar con asiduidad.

echo “hola mundo” | sed '/mundo/q' Muestra el texto hasta que encuentra mundo.

echo “hola mundo” | sed '/mundo/d' Elimina las lineas que contienen mundo.

sed -n '20,30p' Muestra las lineas entre 20 y 30

sed '$d' *Borra la ultima linea.*

sed '1,10d' Borra desde la linea 1 a la 20

Sed tiene muchísimas mas posibilidades, en este articulo de bulma http://bulma.net/body.phtml?nIdNoticia=1281 hay muchísima información sobre sed, podéis ver mas o menos un centenar de usos en http://www.student.northpark.edu/pemente/sed/sed1line.txt

-GREP,EGREP,FGREP

La utilidad grep nos permite buscar en cadenas de texto conforme a distintos parámetros, aparte de grep tenemos fgrep,egrep.

Estos son comandos mas o menos amplios y un poco complejos,
grep cdrom /etc/fstab

Nos mostrara las lineas del archivo /etc/fstab que contienen la palabra cdrom.

grep -v cdrom /etc/fstab

Nos mostrara las lineas que no contienen la palabra cdrom.

Estos comandos son extensos y poseen cierta complejidad, podéis ver un tutorial bastante completo en

documentación útil sobre estas:

http://docs.sun.com/db/doc/801-7484/6i1pp3qp3?l=es&a=view

http://www.ciberdroide.com/misc/novato/curso/regexp.html esta pagina es de las mas cuidadas que he visto, y sobre todo explica de una manera clarisima como usar expresiones regulares.

http://www.zaralinux.org/docs/texto/texto_cap5.php buen articulo sobre fgrep, egrep y grep.

-CUT
Cut nos sirve para cortar (cut del ingles) cadenas de texto, conforme a caracteres.

echo “hola:mundo” | cut -b2

o

Esta opción nos muestra el segundo carácter de la cadena.

echo "hola:mundo" | cut -d: -f2

mundo

Usando -d definimos el campo a partir del cual cortamos (separamos) la cadena de texto y con -f indicamos el campo en cuestión.

echo “hola:mundo” | cut -d: -f1

hola

El mismo ejemplo usando el primer campo

Cut tiene mas opciones, recomiendo la lectura de las paginas del manual, o pruebe cut --help

-PASTE
Nos sirve para combinar ficheros.

archivo 1 archivo 2

main () { printf
(“ hola mundo “);}

paste archivo1 archivo2 > paste-ar1-ar2

fichero paste-ar1-ar2

main () { printf
(“ hola mundo “);}

Este es el uso básico, ver man del comando para poder sacar todo el jugo a paste.

-HEAD
Nos muestra X primeras lineas de un fichero.

Head -20 /etc/fstab

Nos mostraría las primeras 20 lineas de /etc/fstab

-TAIL
Es muy parecido a head, solo que en vez de mostrar las primeras lineas muestra las ultimas X lineas.

Tail -10 /etc/fstab

Nos mostraría las 10 ultimas lineas de /etc/fstab

-SORT
Sort básicamente, (posee bastantes opciones) ordena lineas de texto.

La sintaxis de sort es bastante extensa,

-**NL**

Coloca los números de las lineas en un fichero.

nl fichero

1 linea

2 otra linea

-**TEE**

Nos permite redericcionar la salida a múltiples ficheros.

Tee “hola” archivo archivo2 archivo 3

-**JOIN**

Join nos sirve para comparar cadenas de texto en archivos.

Join archivo1 archivo2

Salida de join --help

-a NUMFICH muestra una línea por cada línea no emparejable del fichero NUMFICH, donde NUMFICH es 1 o 2, correspondiendo a FICHERO1 o FICHERO2.

-e VACÍO reemplaza los campos inexistentes por VACÍO

-i, --ignore-case no atiende a las diferencias entre mayúsculas y minúsculas

-j CAMPO equivalente a '-1 CAMPO -2 CAMPO'

-o FORMATO utiliza FORMATO para mostrar las líneas de salida

-t CARÁCTER Usa CARÁCTER como delimitador de campos, en la entrada y en la salida

-v NUMFICH Como -a NUMFICH, pero no muestra las líneas emparejadas

-1 CAMPO usa este campo del fichero 1

-2 CAMPO usa este campo del fichero 2

-**TR**

Nos permite borrar y comprimir dos cadenas de texto, y substituir cadenas por otras.

P.ejem:

#!/bin/bash

echo “HolA” | tr “a-b” “A-Z”

HOLA

Este script cambiaría las letras en minúsculas por las mismas letras en mayúsculas.

Tr posee mas opciones de uso, puede borrar caracteres, puede operar solo sobre algunos caracteres o puede cambiarlos.

-**BC**

Es un programa que nos permite cálculos numéricos avanzados, es muy pequeño y posee un lenguaje propio
es bastante parecido a C, pero muy simplificado.

-http://usuarios.lycos.es/acisif/bc/bc.html hay un manual que esta muy bien explicado el uso del programa, mucho mejor de lo que yo podría hacerlo.

-http://www.die.net/doc/linux/man/man1/bc.1.html paginas del man en ingles.

http://bulma.net/body.phtml?nIdNoticia=2045 una articulo de la mano de los chicos de bulma.

-UNIQ

Uniq nos permite comparar lineas, contarlas y eliminarlas, (ver man del comando para poder ver mejor todo el uso de uniq).

uniq -c fichero

-DIFF

Compara dos archivos, indicándonos las líneas que son distintas.

Si los ficheros son iguales, no mostrara nada.

Por ejemplo, tenemos los siguientes ficheros:

hola mama (archivo1.txt)

hola mundo (archivo2.txt)

diff archivo1.txt archivo2.txt

1c1 <hola mama --- > hola mundo .

Si no muestra nada , los ficheros son iguales.

-CMP

Compara el contenido de dos archivos, muestra la primera diferencia encontrada, si no son diferentes no muestra nada.

$cmp fichero fichero2

differ: char 2, line 1

Es decir, la primera diferencia está en el carácter 1, línea 1 .

-WC

Es un contador de líneas, palabras y caracteres, su uso es muy sencillo y útil.

-*l*: cuenta líneas.

-*c*: cuenta caracteres.

-*w*: cuenta palabras.

Puede trabajar con cadenas y/o archivos.

$wc -l archivo.txt

2

$wc -c archivo.txt

8

$wc -w archivo.txt

2

Siendo el archivo:

hola

hola

Hasta aquí con los comandos útiles, espero que cuando se planteen problemas del tipo, ¿como hago esto? Con la descripción de las funciones de estos comandos tenga por lo menos una idea sobre donde empezar.

http://www.tldp.org/LDP/abs/html/textproc.html aquí tenéis una referencia muy buena de programas que trabajan con texto, con ejemplos,muy, muy útil, pertenece al advance bash scripting.

inicio

SOCKETS EN BASH

Hemos visto que con bash se pueden hacer muchas cosas, pero a mi entender hay algo que es necesario en cualquier lenguaje: crear sockets, esto es imprescindible cuando se pretende programar a cierto nivel, aquí te propongo ideas al respecto de como usar ciertas herramientas para poder usar tus scripts para bash.

Esta parte del documento es un poco mas avanzada, y por tanto requiere unos conocimientos mayores, vamos a usar Netcat, xinetd y netpipes que se puede decir que se diseño para manejar sockets con otros programas, y por lo tanto usable con scripts para bash.

-Creando servicios basado en xinetd.

Bash no puede manejar sockets como si de C o perl, java, se tratase.

En su lugar si se pueden crear servicios basados en xinetd y que este servicio sea un script para el bash.

En primer lugar NO es el objetivo de este documento explicar que es xinetd ni que hace ni como se usa, tenéis un magnifico articulo en linuxfocus y en español sobre xinetd.

Tambien disponéis de alguna ayuda usando las paginas de redhat sobre el tema en español.

Y por ultimo usar man xinetd.conf para mas detalles.

En el directorio /etc/xinetd.d/ se encuentran los scripts, que usan los demonios que corren sobre xinetd, pues nosotros lo que haremos a continuación, es crear un nuevo servicio y un demonio usando un script en bash, evidentemente no es la mejor idea, pero ilustra la funcionalidad e bash en este aspecto.

Ejemplo de un servicio nuevo.

```
#default: on
#description: Una prueba de creación de servicios usando un script en bash
service arhf
{
port = 9999
socket_type = stream
wait = no
user = root
server = /bin/arhf-login.sh
log_on_success += USERID
log_on_failure += USERID
disable = no
only_from = 192.168.1.4
}
```

Este archivo se guardaria con el nombre del servicio, en este caso arhf.

Repito, para poder ver las opciones de este tipo de archivos, ver la documentación antes reseñada.

Aqui lo que nos interesa es que con un sencillo archivo, definimos un script en bash (/bin/arhf-login.sh) para actué de demonio al recibir conexiones por el puerto 9999,

EL puerto debería estar declarado en /etc/services.

Extracto de la linea que hace referencia en /etc/services:
arhf 9999/tcp #arhf

Y por fin nuestro servicio:

Evidentemente podemos correr desde un "hola mundo" a un script que maneje otros programas, esto ya lo dejo a la imaginación de cada uno, de todas maneras este es un ejemplo útil:

```
#!/bin/bash
var1=$(hostname)

echo "Bienvenido a $var1 corriendo sistema ARHF 0.1"

echo "Introduzca un nombre de usuario"

read v1

var2=$(cat /etc/arhf/passwd | awk '{print $1}'| cut -d: -f1 | grep $v1 )

if [ "$v1" == "$var2" ]; then

echo "Nombre de usuario correcto, introduzca la contraseña "

read key2

key1=$(arhf-auth $key2)

key=$(cat /etc/arhf/shadow | awk '{print $1}'| cut -d: -f3)

if [ "$key" = "$key1" ]; then
exec arhf
else
echo "Desconectado = Contraseña incorrecta
exit 0
fi
else
echo "Desconectando nombre de usuario incorrecto"
exit 0
fi
```

Este script se llamaría arhf-login.sh y seria el que se situaría en /bin/ o donde definamos.

Como se puede ver el script es sencillo, con una función externa llamada arhf-auth encripto la contraseña se la mando comparar con el archivo shadow, y si concuerdan ejecutamos el programa arhf.

Este script tiene una aplicación real, el programa arhf es una pequeña base de datos creada con tcl y archivos de texto plano, como veréis no es que sea precisamente seguro, pero, para ilustrar ejemplo es suficiente.

P.D Una vez creado el nuevo servicio debemos reiniciar xinetd.

-Usando Netcat para manejar sockets.

Esto que hemos echo, simplemente es crear un servicio nuevo basado en xinetd, pero con la particularidad de que el servicio en si es un script para el bash.

Existen diferentes utilidades que nos pueden ayudar a tratar con sockets, una de ellas y quizás la mas usada y la que mejor se adapta a

nuestras necesidad es Netcat, en estas lineas no se pretende un manual de netcat para eso ya existen muchas otras paginas.

Netcat es un programa que nos permite trabajar sobre el protocolo TCP/IP.

La ventaja que tenemos sobre telnet p.ejm, es que podemos interactuar con el programa automatizando tareas, por que no trabaja esperando que interactúes con el.

La forma mas simple de crear una comunicación entre dos máquinas es esta:

```
-----Servidor------
#!/bin/bash
netcat -l -p localhost 9999

---cliente--------
#!/bin/bash
netcat localhost 9999
```

Como podéis ver aquí realmente solo hacemos uso de netcat y poco mas.

Pero, hay otras maneras de usarlo en scripts.

Ejemplo: Un script para gestionar el e-mail, basado en netcat.

```
#!/bin/bash
#ncmailer 0.1 by xento Figal (c) 2004 GNU/GPL http://gnu.org
echo "------------------------------------------------------------"
echo " NCMailer 0.1 By Xento Figal http://xinfo.sourceforge.net "
echo "------------------------------------------------------------"
echo "[1]""Ver correo"
echo "[2]""Mandar correo"
echo "[3]""Eliminar correo"
echo "[4]""Obtener correo"
echo "[5]""Salir del programa"
echo "Introduzca un numero"
read var
case $var in
1)
echo "Introduzca el nombre del servidor pop3"
read v3
echo "Introduzca el puerto del servidor"
read v4
echo "Introduzca el nombre del usuario"
read v1
echo "Introduzca la contraseña"
read v2
echo "USER $v1
PASS $v2
STAT
LIST
QUIT" | nc $v3 $v4
;;
2)
echo "Introduzca el nombre del servidor smtp"
read v1
echo "Introduzca el puerto del servidor"
read v2
echo "Introduca la dirección de origen"
read v3
echo "Introduzca la dirección de destino"
read v4
echo "Introduzca el cuerpo del mensaje sin puntos"
read v5
```

```
function mandar {
cat <<EOF | netcat $v1 $v2
RSET
HELO
MAIL FROM: <$v3>
RCPT TO: <$v4>
DATA
$v5
.
QUIT
EOF
}

echo "¿Mandar el e-mail? si / no"
read v6
if [ "$v6" == "si" ]; then
mandar
echo "El mail se ha mandado"
elif [ "$v6" == "no" ]; then
echo "El mail no se ha mandado"
else
echo "Opcion incorrecta, introduzca si o no sin mayúsculas ni espacios"
exit 0
fi
;;
3)
echo "Introduzca el nombre del servidor pop3"
read v3
echo "Introduzca el puerto del servidor"
read v4
echo "Introduzca el nombre del usuario"
read v1
echo "Introduzca la contraseña"
read v2
echo "USER $v1
PASS $v2
STAT
LIST
QUIT" | nc $v3 $v4
echo "Introduzca el numero del mensaje que quiere eliminar"
read v5
echo "¿Borrar el mensaje numero $v5? si / no"
read v6
if [ "$v6" == "si" ]; then
echo "Borrando el mensaje numero $v5"
elif [ "$v6" == "no" ]; then
echo "El mensaje $v5 no se ha borrado"
else
echo "Opcion incorrecta, introduzca si o no sin mayúsculas ni espacios"
exit 0
fi
echo "USER $v1
PASS $v2
STAT
DELE $v5
QUIT" | nc $v3 $v4
echo "El mensaje numero $v5 ha sido eliminado"
;;
4)
echo "Introduzca el nombre del servidor pop3"
read v3
echo "Introduzca el puerto del servidor"
read v4
echo "Introduzca el nombre del usuario"
```

```
read v1
echo "Introduzca la contraseña"
read v2
echo "USER $v1
PASS $v2
STAT
LIST
QUIT" | nc $v3 $v4
echo "Introduzca el numero del mensaje que quiere descargar"
read v5
echo "USER $v1
PASS $v2
STAT
LIST
RETR $v5
QUIT" | nc $v3 $v4 > mensaje-numero$v5-de-$v3
echo "Mensaje guardado como mensaje-numero$v5-de-$v3"
;;
5)exit ;;
*) echo "Por favor introduzca un numero de opción valido 1,2,3,4,5";
esac
exit
```

Aqui hemos usado variables para definir algunos aspectos de los comandos pop mandados por netcat al servidor, como podréis ver el fallo mas grande(entre otros por supuesto) reside en que cada vez que efectuamos una operación cerramos la conexión y nos toca volverá reiniciarla para efectuar mas operaciones, pero creo que es un cliente de correo bastante ligero no? :))

Esto es por parte del cliente, ahora vamos a ver por parte del servidor.

```
#!/bin/bash
echo "Bienvenido" | netcat -l -p 9999
exit
```

Esto nos mostrara en mensaje Bienvenido y cerrara la conexión.

```
#!/bin/bash
netcat -l -p 9999 < archivo.txt
```

Este script usara archivo.txt para mostrarlo en pantalla cada vez que alguien se conecte por ese puerto.

Por otro lado, una manera muy fácil de crear un demonio es esta:

Ejecutamos netcat de esta manera:

```
netcat -l -p 9999 -e script.sh
```

Al hacer así las cosas, lo que el indicamos a netcat es que permanezca a la escucha en el puerto 9999 y que cuando reciba una conexión, ejecute script.sh.

NOTA: Dado que permitimos la conexión, el archivo debe tener permisos de ejecución y lectura.

Si hacemos esta combinación:

```
netcat -l -p 9999 -e ncmailer.sh
```

La conexión entrante, dispondrá de un cliente de correo.

Por lo que solo queda hacer un script que haga lo que nosotros queremos.

De todas maneras en http://www.csd.uch.gr/~mstamat/hacks/foohttpd.html tenéis un servidor web echo con netcat y bash, otro en

http://lug.umbc.edu/~mabzug1/bash-httpd.html y numerosa documentación.
.

Ademas de netcat podemos aprovechar otras herramientas,en el paquete de netpipes, encontramos una variedad interesante, la explicación del uso de estas herramientas no es tan extensa como la que se dio con netcat, netcat es quizás la herramienta mas versátil para este tipo de scripts, pero recuerdo que el documento no es un manual de este tipo de herramientas, por lo que sera necesario que el lector aprenda a usarlas para poder extraer todo el rendimiento de estas.

El paquete netpipes contienen diversas utilidades, nos vamos a centrar en dos:

-Hose: Nos permite conectarnos a un puerto abriendo un socket.

Este paquete nos sera de utilidad para crear clientes.

-faucet: Nos permite abrir un socket en la maquina local.

Este otro nos servirá para crear servidores.

Ejemplo de faucet permaneciendo a la escucha en el puerto 9999.

faucet 9999 --out --verbose --once echo "hola cliente"

Cuando un cliente se conecte al puerto 9999 vera el texto hola cliente y se cerrara la conexión. La diferencia es que podemos usar de una manera mas intuitiva para crear scripts.

http://www.starlinux.net/articulos/bash3/starbot tenéis un cliente de irc construido con netpipes.

http://web.purplefrog.com/~thoth/netpipes/netpipes.html aquí encontraras tanto el paquete como el manual y ejemplos de uso.

Autentificacion basada en bash:

Para manejar las contraseñas en los scripts tipo servidor, es mas seguro autentificarlas, aquí tenéis un programa que es muy útil para esos casos.

Ya se que puedo pecar de insistente, pero el uso de estas herramientas a conciencia y en profundidad se aleja muchísimo del ámbito de este tutorial.

inicio

BASH EN TIERRAS HOSTILES (El titulo y la idea de este apartado se lo debo a Subbohfer: es.comp.os.linux)

A veces nos hace falta correr scripts en bash, o usar directamente utilidades de Linux en otros entornos.

En estos casos, podemos recurrir a ciertas herramientas, no puedo poner una descripción muy grande puesto que no las he probado nunca, pero tal vez a alguien le puedan ser de utilidad.

Aqui tenéis un bash 2.03, precompilado para windows.

Esto otro, son una serie de .exe, según ellos son 3 cosas:

- Libraries that provide a UNIX operating system by implementing the UNIX Application Programming Interface (API).
- Include files and development tools such as cc, yacc, lex, and make.
- Korn Shell and over 250 utilities such as ls, sed, cp, stty etc.

Son varios .exe pero según vi (no lo he probado) puedes bajar la korn en un solo .exe, solo tiene un pero; esta bajo licencia restrictiva, te dejan usarlo como "educational" o bien por un periodo de 90 dias de prueba, la otra solución es pagar la licencia.

Tambien esta esto , es*"An implementation of the CSH for Windows"*
Hay una versión gratuita para evaluación y demás. pero no huele muy bien, mas bien capada seria la mejor definición.

Estas , son las utilidades Gnu para windows incluida ZSH, con comandos varios (cat, grep, find, etc...) , pero supongo que serán unos cuantos .exe

Aqui hay una serie de utilidades, cat, bc, shell-utils, less, textutils, y unas cuantas mas, no es necesario cgywin.

En esta linea también hay otra cosilla, que si bien parte de cgywin dicen que no es necesario para hacerlas correr, son 18 utilidades, incluidas gzip, bash etc..., lo de la licencia no esta muy claro, no es trial, no es share, no supe averiguar que tipo de licencia es.

Evidentemente para poder hacer funcionar las cosas, y en esta linea esta claro que cgywin es la opción mas recomendable, pero si tenéis que usar scripts para una red, cgywin trae el problema de instalarlo en todas las máquinas, este problema esta en un hilo de discusión actualmente "parado" en libertonia.

inicio

DEPURANDO BASH

Existen dos formas de hacer una depuración (debug) de nuestros scripts.

Añadiendo un echo a todas las las variables. (esta no es "real")

Estas otras si son modos de depuración real.

sh -x scriptconerror

Esto ejecutará el script y mostrará todas la sentencias que se ejecutan con las variables y comodines ya expandidos.

El shell también tiene un modo para comprobar errores de sintaxis sin ejecutar el programa. Se usa así:

sh -n tu_script

Si no retorna nada entonces tu programa no tiene errores de sintaxis.

inicio

APENDICE 1

Comando de MS-DOS	**Comando en linux**	**descripciones**
ASSIGN	ln	Crear enlaces
ATTRIB	chmod	Cambiar permisos

Comando de MS-DOS	Comando en linux	descripciones
CD	cd	Cambiar directorio
CHDIR	cd	Cambia directorio
CLS	clear	Limpia la pantalla
COMP	diff, comm, cmp	Compara archivos
COPY	cp	Copiar archivos
Ctl-C	Ctl-C	Señal break
Ctl-Z	Ctl-D	Final de archivo
DEL	rm	Borrar archivo
DELTREE	rm -rf	Borrado recursivo
DIR	ls -l	Ver directorio
ERASE	rm	Borrar archivos
EXIT	exit	Salir de un proceso
FC	comm, cmp	Comprar archivos
FIND	grep	Buscar cadenas de texto
MD	mkdir	Crear directorio
MKDIR	mkdir	Crear directorio
MORE	more	Ver pantalla a pantalla
MOVE	mv	mover
PATH	$PATH	Ruta
REN	mv	Renombrar o mover
RENAME	mv	Renombrar o mover
RD	rmdir	Borrar directorio
RMDIR	rmdir	Borrar directorio
TIME	date	Fecha
TYPE	cat	Salida hacia stdout
XCOPY	cp	Copia de archivos

inicio

Argumentos de ejecución del bash.

Dentro de un script podemos añadir argumentos con -o argumento.

Para eliminar esos argumentos +o argumento.

Opcion.	Nombre	descripciones
-C	noclobber	No permite sobrescribir archivos por redireccion.

Opcion.	Nombre	descripciones
`-D`	(none)	Muestra las dobles ( ) precedidas de $, pero no ejecuta en el script.
`-a`	allexport	Exporta todas las variables definidas.
`-b`	notify	Notifica cuando hay trabajos por termina en background.
`-c ...`	(none)	Lee comandos desde **...**
`-f`	noglob	Expansion de archivos desactivada.
`-i`	interactive	*El script se ejecuta en modo interactive.*
`-p`	privileged	El script se ejecuta como "suid"
`-r`	restricted	El script se ejecuta en modo *restricted*.
`-u`	nounset	Se usa para que si un fallo de salida de variable lo fuerza y sale de este.
`-v`	verbose	Modo verbose
`-x`	xtrace	Modo verbose con expansión de comandos
`-e`	errexit	Aborta el script al primer error.
`-n`	noexec	Lee los comandos del script pero no los ejecuta.
`-s`	stdin	Lee comandos desde `stdin`
`-t`	(none)	Sale antes del primer comando.
`-`	(none)	Fin de las opciones con flags.

inicio

REFLEXIONES IMPORTANTES.

Como se puede comprobar, los scripts para el bash forman parte de nuestro sistema, (rc por ejemplo) por lo que una comprensión mínima de como funcionan es obligado para poder conocer que hace, como y cuando lo hace nuestra maquina.Por otro lado, existen programas que son directamente scripts en bash y que permiten hacer muchas cosas.Los scripts para el bash nos van a solucionar la vida y automatizar muchas tareas.

Tambien nos serán muy útiles y en algunas ocasiones imprescindibles para realizar muchas cosas.Como añadido hay que citar que aprender a escribir scripts para el shell, nos puede ser muy útil para crear una puerta de entrada a la programación (mis primeros pasos en linux fueron con scripts para el bash), puesto que posee una sencilla manera de hacer las cosas, nadie me puede negar que es una de las mejores maneras de introducirse en el mundo del código.

Se que muchos lectores/as se quedaran un poco "vacíos" con algunas secciones especialmente el uso de programas externos, bash en tierras hostiles y los sockets en bash,pero una vez mas debo recordar que el objetivo del documento era dotar de ideas para empezar a escribir script para bash, y no crear una guia avanzada de programación, de todas maneras estoy animándome a escribir un how-to centrado nada mas en el uso de netcat y netpipes y la creación de script potentes, cualquier ayuda sera bienvenida.

inicio

NOTAS DE LA VERSION.

-Version 2.0 del 1.6
-Version en html , html.tar.gz , pdf.tar.gz en:
http://xinfo.sourceforge.net/documentos/bash-scripting/

-Actualizaciones e hilos http://xinfo.sourceforge.net
