Comandos en linux

Que es un comando ?

En linux, un "comando" es un programa que tu puedes ejecutar. Se teclea el nombre del comando y se presiona Enter. Por ejemplo: teclea date y pulsa enter.

Comandos simples:

- Who
- Who am i

Comandos complejos:

- $ comando argumento1 argumento2 argumento3 ... argumentoN

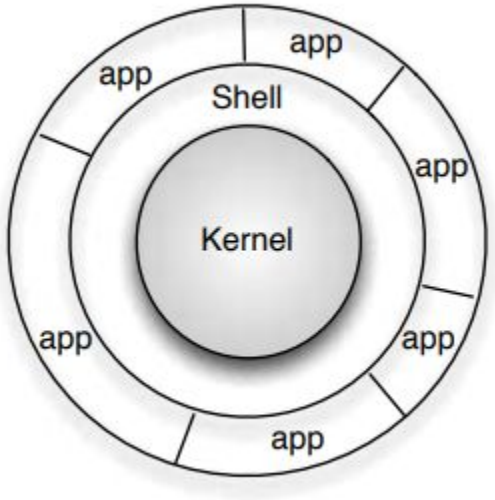
Comando compuesto:

- $ comando1 ; comando2 ; comando3 ; ... ; comandoN

Shell

El shell provee una interface a el sistema. Obtiene la informacion del usuario y ejecuta programas en base a esa informacion. Cuando el programa finaliza su ejecucion, despliega la salida de el programa. Por esta razon el shell es conocido como el interprete de comandos.

Una de las grandes ventajas de los sistemas *nix es que el shell es mas que un interprete de comandos, tambien es un lenguaje de programacion completo, con instrucciones de condicionales, asignacion, ciclos y funciones.



Tipos de shell

Desde la creacion de el sistema Unix original han existido diferentes tipos de shells con diferentes caracteristicas, el primer shell (el original creado por Dennis Ritchie y Ken Thompson) unicamente era un interprete de comandos, localizaba el comando y lo ejecutaba, cuando el comando terminaba el control regresaba a el shell, el cual estaba listo nuevamente para recibir algun otro comando o instruccion.

Durante los mas de 30 años que los sistemas *nix han existido se han creado una gran variedad de shells con diferentes caracteristicas, la mayoria de estos shells estan basados en 2 tipos de shell's:

- Bourne Shell (sh).- El shell mas popular de los sistemas *nix, este shell ha sido la base para crear diferentes variantes del mismo, algunas de estas variantes son:
    - Korn Shell (ksh)
    - Bash o Bourne Again Shell (bash)
    - POSIX shell (sh)
- C shell (csh).-

¿Cuál es la mejor? En la que tu te sientas mas cómodo ☺ Nosotros veremos bash ya que es la de defecto en Linux.

Primer script

```
#!/bin/bash
echo “Hola Mundo”
```

Al guardar recordad dar permisos de ejecución

```
chmod +x nombre_del_script.sh
```

Para ejecutarlo:

```
./nombre_del_script.sh
```

Entrada y salida

Cualquier comando que se ejecuta puede generar dos tipos de salida:

Resultado. Es decir, aquello para lo que está pensado la herramienta en cuestión.

Estado. El estado se refiere a si la ejecución ha sido correcta. En caso de que la ejecución no haya sido correcta devolverá un mensaje de error para informar al usuario de los detalles de la ejecución.

En Linux existen tres tipos de archivos por defecto:

stdin ó 0. Está conectado con el teclado.

stdout ó 1. Este archivo está conectado con la pantalla.

stderr ó 2. Cuando se produce un error, este se envía a este archivo, que también está conectado con la pantalla.

Redirigir entrada y salida te permite modificar tanto la entrada como la salida. Si normalmente introduces información desde el teclado, stdin, podemos hacer que entre desde otro archivo. Igualmente puedes proceder con la salida.

Redireccionar a archivos distintos de los de defecto

Comprobad la existencia de los ficheros por defecto:

```
ls  -l  /dev/std*
```

Probemos los siguientes comandos:

```
ls   >  salida.txt
```

```
ls  1> salida.txt
```

```
ls 2> salida.txt
```

¿Cuál es la diferencia?

Operadores de redirección

> redirecciona la salida hacia un archivo, sobrescribiendo la información anterior si la hubiese.

> >redirecciona la salida hacia un archivo, añadiendo la información a la existente si la hubiese

< redirecciona la entrada desde un archivo, sobrescribiendo la información anterior si la hubiese.

Probemos:

#!/bin/bash

Stdin utilización

Caso habitual stdin metiendo variables desde el teclado:

```
#!/bin/bash
echo "Introduce tu nombre:"
read nombre
echo "Introduce tu apellido:"
read apellido
```

Ejemplo redireccionando el stdin desde un fichero:

```
sort < lista_de_palabras.txt > lista_de_palabras_ordenada.txt
```

Variables

Ejemplo 1:

```
#!/bin/bash
MENSAJE=”Hola Mundo”
echo $MENSAJE
```

Ejemplo 2:

```
#!/bin/bash
echo "introduce un numero:"
read dato1
echo “introduce otro numero:"
read dato2
multipli=$[$dato1*$dato2]
echo "$dato1 multiplicado por $dato2 da como resultado $multipli“
```

**No se debe usar una palabra reservada del sistema como nombre de una variable.

Comentarios en Scripts

```
#!/bin/bash
 #esta linea es un comentario, no afecta en nada a la ejecución de este script
MENSAJE="Hola Mundo"
#otro comentario que no afecta
echo $MENSAJE
```

Es buena idea para documentar nuestros scripts el utilizar cabeceras como:

```
#!/bin/bash
#**************************************************************
 #******   Tema1:  ejercicio sobre comentarios     ************************
#**************************************************************
MENSAJE="Hola Mundo"
echo $MENSAJE
```

Condiciones: cadenas de texto

["$a" = "$b"] Igualdad
["$a" == "$b"] Igualdad
["$a" != "$b"] Desigualdad
["$a" \< "$b"] Orden alfabético ascendente.
["$b" \> "$b"] Orden alfabético descendente.
[-z "$a"] es verdadero si la variable esta vacía
[-n "$a"] es verdadero si la variable no esta vacia

Condiciones: números enteros

["$a" -eq "$b"] Igualdad
["$a" -ne "$b"] Desigualdad
["$a" -gt "$b"] Mayor que
["$a" -lt "$b"] Menor que
["$a" -ge "$b"] Mayor o igual que
["$a" -le "$b"] Menor o igual que

Condiciones: ficheros

[-e nombre fichero] Existencia
[-f nombre fichero] es fichero ordinario
[-s nombre fichero] es fichero vacío (0 bytes)
[-d nombre fichero] es directorio
[-b nombre fichero] es dispositivo de bloques
[-c nombre fichero] es dispositivo de caracteres
[-L nombre fichero] es enlace simbólico
[-r nombre fichero] tiene permiso de lectura
[-x nombre fichero] tiene permiso de ejecución
[-w nombre fichero] tiene permiso de escritura
[-O nombre fichero] somos propietarios del fichero (UID)
[-G nombre fichero] somos propietarios del fichero (GID)
[-N nombre fichero] el fichero se modifico desde la ultima vez que fue leído
[-h nombre fichero] es enlace duro

Recordar que para negar se usa el carácter !
[! -e fichero]
[! -f fichero]

Estructuras condicionales

Condición sencilla:

if <condición>;
then <comandos>
fi

También se puede incluir acción en caso de que no se cumpla la condicion:

if <condición>;
then <comandos>
else <comandos>
fi

Estructuras condicionales II

Con bloques:

if <condición 1>;
then
<bloque 1>
elif <condición 2>;
then
<bloque 2>
elif <condición 3>;
then
<bloque 3>
fi

Estructuras condicionales III

Ejemplo:

```
#!/bin/bash
echo “elija su opción”
echo "1 - Verdadero"
echo "2 - Falso"
read OP
if [ "$OP" = "1" ];
then
echo "Esto se vera si la condición es VERDAD"
else
echo "Esto se vera si la condición es FALSA"
fi
echo "FIN"
```

Condicional: Case

```
read x
 case $x in
   1)
     echo "uno"
   ;;
   2)
     echo "dos"
   ;;
   *)
     echo "no sé qué numero es"
   ;;
 esac
```

Bucles: For

```
for a in $x
do

……..

done
```

Ejemplo:

Tengo 10 archivos (manzanas) en una determinada carpeta, llamados respectivamente desde el número 1 hasta el 10 (sus nombres serían entonces 1, 2, 3, 4... y así hasta llegar al 10 respectivamente. Cada archivo contendra un digito (0 ó 1).

Leer cada fichero y determinar cuantos contienen el numero 1 (Cuantas manzanas tienen gusano).

Solución ejemplo

```
#!/bin/bash
# Manzana Parser: El script importa la variable guardada en cada archivo
#                 y determina si "tiene o no gusano"

## Verificando que los parámetros sean válidos
if [ $# -ne 1 ]
then
        echo "Haz el favor de sólo especificar un directorio"
        exit 1;
elif [ ! -d $1 ]
then
        echo "El archivo $1 especificado no es un directorio, abortando."
        exit 1;
fi

##Cambiando al directorio especificado

lastdir=$PWD
cd $1
```

Solución ejemplo ….continuación

```
## Inicializando aplicación

contador=0

for archivo in `ls $1`
do
      if [ -f $archivo ]
      then
            let contador=contador+1
      fi
done

for manzana in $contador
do
      if [ -f $archivo ]
      then
            source $archivo
            if [ $gusano = 0 ]
            then
            echo "La manzana $archivo no tiene gusano, guardando en $HOME/refrigerador"
            mv $archivo $HOME/refrigerador
            else
            echo "La manzana $archivo tiene gusano, eliminando la manzana"
            rm $archivo
            fi
      fi
done

## regresando al directorio anterior|

cd $lastdir

exit 0
```

Bucles: while-do-done

```
while [ "condición lógica" ]
do
    acción 1
    acción 2
    acción n
done
```

Ejemplo:

```
limite=5
i=0;

while [ $limite -gt $i ]
do
    echo -e "Acción $i ejecutada.\n"
    let i=$i+1
done
```

Bucles: until-do-done

```
until [ "condición lógica" ]
do
    acción 1
    acción 2
    acción n
done
```

Ejemplo:

```
limite=5
i=10;

until [ $limite -gt $i ]
do
    echo Acción $i ejecutada
    let i=$i-1
done
```

Scripts ejemplo

1- Un script al que le pasamos un cierto numero de parametros y tiene que sacar por pantalla:

”Has introducido: ___ parametros”

2-Añadir al script un bucle que imprima los parametros que se han pasado