Shell y Script

El shell nos provee un interfaz de usuario para acceder a los servicios del sistema operativo, estos pueden ser gráficos o en modo texto, y se encargan de ejecutar los distintos programas que se encuentran disponibles en el sistema.

Los shell en modo texto más comunes son Bourne Shell (/bin/sh), Bash, Korn Shell (ksh). Los Shell en modo gráfico son GNOME, KDE, etc. Una forma rápida de saber en qué shell nos encontramos es ejecutando el comando:

```
$ echo $SHELL
```

Todas las shell tienen asociados unos ficheros de arranque de inicialización, para que cada usuario del sistema personalice su entorno, en la shell bash, $HOME/.bashrc, el sistema tiene un fichero de personalización para todos los usuarios en /etc/bash.bashrc, al abandonar el sistema se ejecuta $HOME/.bash_logout

Una vez estamos en la consola (Shell) se muestra el prompt al usuario, ($) si es un usuario normal o (#) si es el usuario administrador, y ya podemos introducir comandos para su ejecución, mediante las siguientes fases:

- Interpretar el comando introducido.
- Evaluar los caracteres comodines ($,*,?)
- Gestionar la redirecciones de E/S, los pipes (|), los procesos en segundo plano (&)
- Gestión de señales
- Preparar la ejecución de los programas.

Podemos agrupar un conjunto de comandos en un fichero y darle permisos de ejecución creando un Shell script para poder automatizar tareas repetitivas en la administración de sistemas a realizar por el administrador.

Todos los comandos ejecutados en la Shell tienen tres ficheros predefinidos que son:

- Entrada estándar, asignada al teclado de la consola y usa el descriptor 0.
- Salida estándar, asignada a la pantalla del terminal y usa el descriptor 1.
- Salida de error estándar, normalmente asignada a la pantalla del terminal y usa el descriptor 2.

De esta forma la entrada de comando por defecto al sistema se realiza mediante el teclado, la salida es por pantalla y si hay errores estos también se muestran por pantalla.

En la Shell también existen los conceptos de:

- Redirección, que nos permite dirigir las salidas y entradas estándar de un comando o programa hacia un fichero, comando etc.

  Enviar la salida del comando a un fichero.

  ```
  $ comando > fichero
  ```

  Enviar la salida del comando al final del fichero.

  ```
  $ comando >> fichero
  ```

  Enviar como entrada del comando el contenido del fichero.

  ```
  $ comando < fichero
  ```

  En el intérprete de comandos para evitar que interprete una cadena de caracteres lo pondremos entre comillas simples '...' . Si queremos que el intérprete de

comandos no interprete una cadena de caracteres excepto $, ", \ y comillas simples, usaremos comillas dobles "....".

- Tuberías (pipes). la salida estándar de un programa puede usarse como entrada estándar de otro por medio de *pipes*. Varios programas pueden ser conectados entre sí mediante *pipes* para formar lo que se denomina un *pipeline*. El comando empleado es (|).

  # comando1 | comando2 | comando3

  Por ejemplo:

  # ls -al | wc -l

  Nos da el número de ficheros que hay en el directorio actual.
- Concurrencia, consiste en ejecutar varios programas a la vez, indicando que su ejecución de va a producir en segundo plano. Se representa mediante el símbolo **&**

  # comando &

  Ejecución secuencial de comandos en una sola línea:

  # comando1 ; comando2

Variables del sistema

VARIBLE	Descripción
CDPATH	Una lista de directorios separados por el signo ':' usada como ruta de acceso por el comando cd
HOME	El directorio principal de usuario
MAIL	Si este parámetro tiene un fichero definido y la variable MAILPATH no está definida, bash informa al usuario de la llegada de correo al fichero especificado.
MAILPATH	Una lista de ficheros separada por comas, en los cuales el intérprete de comandos comprueba periódicamente de la llegada de correo.
OPTARG	El valor del último argumento procesado por getopts.
OPTIND	El índice del último argumento procesado por getopts
PATH	Una lista de directorios, separados por comas, en los cuales el interprete de comandos busca por comandos
PS1	Prompt principal. El valor por defecto es '"\s-\v\$ '
PS2	El prompt secundario. El valor por defecto es '> '
BASH_VERSION	El numero de versión de bash.
COLUMNS	Usada por 'select' para determinar el ancho de la terminal cuando imprime listas de menús.
DIRSTACK	Una matriz que contiene los contenidos actuales del stack de directorios
EUID	El identificador numérico de usuario del usuario actual
FCEDIT	El editor usado por defecto por la opción -e del comando 'fc'
FUNCNAME	El nombre de la función que se está ejecutando actual
GROUPS	Una matriz que contiene la lista de los grupos a que pertenece el usuario actual
HISTCMD	El índice del comando actual en la historia de comandos
HISTCONTROL	Define si un comando es añadido a la historia de comandos

HISTFILE	El nombre del fichero en el cual se graba la historia de comandos de comandos. El valor por defecto es ~/.bash_history
HISTFILESIZE	El número máximo de líneas contenidas en la historia de comandos, por defecto 500
HISTIGNORE	Una lista separada por comas de los patrones usados para definir que comandos deben de grabarse en la historia de comandos
HISTSIZE	El máximo número de comandos a recordar en la historia de comandos, por defecto 500
HOSTNAME	El nombre de maquina actual
HOSTTYPE	Cadena describiendo el tipo de sistema que está ejecutando Bash
IGNOREEOF	Controla la acción a tomar cuando el interprete de comandos recibe un carácter EOF
INPUTRC	Nombre del fichero de inicialización de 'Readline', sobre escribiendo el valor por defecto /etc/inputrc.
LINES	Usada para determinar la anchura de la columna usada para imprimir listas
MACHTYPE	Cadena describiendo el tipo de sistema que está ejecutando Bash
MAILCHECK	Frecuencia de comprobación (en segundos) del correo electrónico en el fichero definido en las variables MAILPATH o MAIL
OLDPWD	Directorio previo definido por el comando 'cd'
OSTYPE	Cadena describiendo el sistema operativo que está ejecutando Bash
PPID	El numero de proceso del proceso padre del intérprete de comandos
PS3	El valor de esta variable se usa como 'prompt'
PWD	Directorio actual definido por el comando 'cd'
RANDOM	Cuando se llama esta variable un numero entero entre 0 32767 es generado
SECONDS	Numero de segundos desde que Bash fue arrancado
SHELLOPTS	Lista con opciones de Bash activadas
UID	El valor numérico real del usuario actual

Estas variables del sistema se pueden consultar con el comando echo, las variable de entorno definidas en el sistema puede consultarse con el comando:

```
$ env
```

Programación de scripts en Bash

Todos los scripts bash comienzan con la línea:

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

Para ejecutar los script podemos hacerlo de dos formas:

- Dando permiso de ejecución al script,
  ```
  $ chmod +x nombre_script
  ```

- Ejecución mediante shell,
  $ bash nombre_script

Variables en bash

La asignación de variables se realiza por:

variable = valor

El valor de la variable se puede ver con:

echo $variable

donde '$' nos hace referencia al valor de la variable.

La variable por defecto, su ámbito está restringido al script (o en el shell). Si la variable debe ser visible fuera del script será necesario "exportarla" además de asignarla. Podemos hacerlo de dos formas:

- Asignar y exportar después:
  var = valor
  export var
- Exportar en la asignación:
  export var = valor

En los scripts Bash tenemos algunas variables predeterminadas accesibles:

- $1-$N: Guarda los argumentos pasados como parámetros al script desde la línea de comandos.
- $0 : Guarda el nombre del script, sería el parámetro 0 de la línea de comandos.
- $* : Guarda todos los parámetros del 1 al N en esta variable.
- $: Guarda todos los parámetros, pero con comillas dobles (" ") en cada uno de ellos.
- $? ,"Status": guarda el valor devuelto por el último comando ejecutado. Útil para verificar condiciones de error, ya que Linux suele devolver 0 si la ejecución ha sido correcta, y un valor diferente como código de error.

Uso de comillas

Comillas dobles. En general los caracteres especiales no son interpretados cuando están entre comillas dobles. Sin embargo alguno de ellos sí son interpretados:

$ Esta permitido referenciar variables dentro de las comillas dobles.

\ Se pueden escapar caracteres.

` Se puede realizar sustitución de comandos, esto es, ejecutar un comando y sustituir la cadena por su salida.

Comillas simples. Dentro de las comillas simples todos los caracteres son interpretados literalmente, ninguno de los caracteres especiales conserva su significado dentro de ellas.

Comillas invertidas. Poner una cadena entre comillas invertidas supone ejecutar su contenido como un comando y sustituir su salida. Por ejemplo: var = `ls` guarda el listado del directorio en $var.

Comparaciones

Para las condiciones se suele utilizar la orden test expresión o directamente [expresión]. Podemos agrupar las condiciones disponibles en:

Comparación numérica: -eq, -ge, -gt, -le, -lt, -ne, correspondiendo a: igual que, más grande o igual que (ge), más grande que (gt), menor o igual que (le), menor que (lt), distinto que (ne).

Comparación de cadenas: =, !=, -n, -z, correspondiendo a cadenas de caracteres: iguales (=), diferentes (!=), con longitud mayor que 0 (n), longitud igual a cero o vacío (z).

Comparación de ficheros: -d, -f -r, -s, -w, -x -h, -e. El fichero es: un directorio (d), un fichero ordinario (f), es de lectura (r), es no vacío (s), es escribible (w), es ejecutable (x), es un enlace simbólico (h), si el fichero existe (e).

```
if [ -e fichero ]
then
        echo "\"fichero\" existe"
fi
```

Booleanos entre expresiones: !, -a, -o, condiciones de not , and y or.

Comando internos

Un comando interno en **bash** es un comando que se implementa y que ejecuta sin llamar a comandos externos. Por ejemplo, **echo** es un comando interno de **bash** y cuando se llama desde un script no se ejecuta el fichero /bin/echo.

Los comandos internos más utilizados:

echo Envía una cadena a la salida estándar, normalmente la consola o una tubería.

read Lee una cadena de la entrada estándar y la asigna a una variable:

```
echo -n "Introduzca un valor para var1: "
read var1
echo "var1 = $var1"
```

cd Cambia de directorio

pwd Devuelve el nombre del directorio actual, equivale a leer el valor de la variable $PWD.

pushd Apila un directorio en la pila de directorios.

popd Desapila y cambia a ese directorio

dirs muestra el contenido del directorio de la pila

let Realiza operaciones aritméticas:

```
let var1=$var2+6
```

export Hace que el valor de una variable esté disponible para todos los procesos hijos de la shell.

. source Estos dos comandos hacen que la shell cargue otro fichero. Es equivalente a #include en C/C++.

exit Finaliza la ejecución del script, recibe como argumento un entero que será el valor de retorno del script.

ps Ofrece un listado de los procesos que hay corriendo en la máquina, con diversa información sobre los mismos.

fg Devuelve un proceso que estaba ejecutándose en segundo plano al primer plano.

wait Detiene la ejecución hasta que los procesos que hay en segundo plano terminan.

kill Envía una señal a un proceso, normalmente para terminarlo.

Estructuras de control

Las estructuras de control propias de cualquier lenguaje de programación (for, while, ...), con la sintaxis propia de Bash.

La sintaxis básica de las estructuras de control es la siguiente:

Estructura if...then, se evalúa la expresión y si se obtiene un valor cierto, entonces se ejecutan los comando.

```
if [ expresion ]
then
        comando
fi
```

Estructura if..then...else, se evalúa la expresión, y si se obtiene un valor de cierto, entonces se ejecutan los comando1, en caso contrario se ejecutan los comando2:

```
if [ expresion ]
then
        comando1
else
        comando2
fi
```

Estructura if..then...else if...else, misma utilización que la anterior, con anidamientos de estructuras if.

```
if [ expresion ]
then
        comando
elif [ expresion2 ]
then
        comando
else
        comando
fi
```

Estructura case select, estructura de selección múltiple según valor de selección (en case)

```
case string1 in
str1)
        comando;;
str2)
        comando;;
*)
        comando;;
esac
```

Bucle for, sustitución de variable por cada elemento de la lista:

```
for var1 in list
do
        comando
done
```

Bucle while, mientras se cumpla la expresión:

```
while [ expresion ]
do
        comando
done
```

Bucle until, hasta que se cumpla la expresión:

```
until [ expresion ]
do
        comando
done
```

Declaración de funciones

La forma de declarar una función en bash es:

```
fname() {
        comando
}
```

Y con paso de parámetros:

```
fname2(arg1,arg2...argN) {
        comando
}
```

y la llamadas de la función con fname o fname2 p1 p2 p3 ... pN.

Práctica:

Crear un script donde debemos introducir como parámetro el nombre de un fichero, y sacar por pantalla si el archivo es de lectura, escritura o ejecutable por el usuario.