There are no translations available.
El objetivo de este tutorial es acercarnos a la programación de scripts de una manera sencilla..

Tutorial Shell Scripts

El día a día de un administrador de Sistemas, ya sea de una importantísima red o de un modesto servidor, esta lleno de tareas repetitivas y generalmente tediosas. Para facilitar nuestra tarea existen los scripts. Pequeños programas cuya finalidad es precisamente automatizar y encadenar tareas relacionadas con los sistemas.

El objetivo de este tutorial es acercarnos a la programación de scripts en LINUX (UNIX) de una manera sencilla,.. Primero con nociones de teoría y segundo con algunos ejemplos. Los ejemplos pretenden ser útiles no solo por que ayuden al lector a entender lo expuesto en la teoría sino porque son scripts sacados de un entorno de producción real que pueden ser usados por cualquier administrador de sistemas que tenga escenarios y necesidades parecidas.También se incluye al principio una pequeña explicación sobre los scripts en un entorno Windows.

El tutorial se divide en los siguientes puntos:

1. Introducción (Tutorial Shell Scripts I)
2. Shell en Windows
3. SCRIPTS EN LINUX: Shell de LINUX. Historia y Conceptos básicos
4. SCRIPTS EN LINUX: Estructuras de control (Tutorial Shell Scripts II)
5. SCRIPTS EN LINUX: Ejemplos de Scripts I
   - Script par realizar un "ping" a todas las máquinas de nuestro sistema
6. SCRIPTS EN LINUX: Un paso adelante : SSH sin contraseña, RSYNC y AWK. (Tutorial Shell Scripts III)
7. SCRIPTS EN LINUX: Ejemplos de Scripts II.
   - Script para automatizar la réplica de una Base de Datos MySQL
   - Script para la creación de usuarios en varias máquinas remotas

En esta primera entrega veremos los tres primeros.

1. Introducción

Antes de empezar a meternos de lleno en el mundo de la programación de Shell-Script, haremos una pequeña introducción, explicando los conceptos mas sencillos y realizando un breve resumen acerca de la historia de las shells, los diferentes tipos... También explicaremos el Crontab para la automatización de tareas, ¿estas preparado?...pues comenzamos..

1.1 ¿Qué es una shell?

S*hell* es el intérprete de comandos, es decir, como los ordenadores no entienden nuestro lenguaje (sólo saben de ceros y unos), necesitaremos un programa intermedio, capaz de hacer que cuando nosotros tecleemos alguna orden, nuestro ordenador sea capaz de entenderlo. Es decir proporciona comunicación directa entre el usuario y el sistema operativo.

1.2 ¿Qué es un Shell Script?

Normalmente, usamos el término Shell Script para referirnos a programas escritos para la shell de **UNIX**/**LINUX**, mientras que cuando usamos la línea de comandos de **MS-DOS** (COMMAND.COM) o el cmd.exe de **Windows**, nos referimos como Batch files (archivos por lotes) y los guardaremos con extensión .bat.

La programación en shell-script es muy útil para resolver tareas repetitivas, típicas de los Administradores. Son ficheros de texto que contienen comandos y son directamente ejecutables por el sistema.

2. Shell en Windows

La shell de comandos de los sistemas operativos Windows utiliza el intérprete de comandos Cmd.exe, que carga aplicaciones y dirige el flujo de información entre ellas. Entra en Inicio/ejecutar y escribe cmd para iniciar una nueva instancia del intérprete de comandos.

Puedes utilizar el shell para automatizar tareas rutinarias, tales como las copias de seguridad o la administración de los perfiles de usuarios.

Siempre se ha he tenido la idea de que la shell de Windows es bastante limitada en comparación con UNIX (la ausencia de comandos como grep o awk...) Nosotros veremos un sencillo ejemplo para Windows, pero nos centraremos a lo largo del tutorial en la programación de Shell Scripts en UNIX/LINUX.

Para comenzar, podemos probar a escribir el comando, **color 57,** para cambiar el color del símbolo del sistema al color blanco y el fondo a púrpura. Una vez que hemos probado que la shell, "nos hace caso", podemos probar el
resto de comandos
ejecutables en la shell de Windows.

A continuación muestro en una tabla los principales comandos:

COMANDO	DESCRIPCION
:ETIQ	Identifica una posición de salto
%NUM	Introduce parámetros en el fichero
CALL	Llama a otro fichero Batch
CLS	Borra la pantalla
ECHO	Visualiza en pantalla una secuencia de caracteres
FOR	Repite un número determinado de veces un mismo proceso
GOTO	Salta y ejecuta una nueva línea indicada por una etiqueta
IF	Se utiliza para saltos condicionales
PAUSE	Detiene momentáneamente la ejecución de un fichero
REM	Introduce un comentario
SHIFT	;@ Evita que una línea aparezca por pantalla

2.1 Argumentos

Si preparamos un script para windows o unix podemos hacerlo de tal manera que sea capaz de ejecutarse recibiendo argumentos en el momento de su llamada. Por ejemplo podemos crear un script de copia de seguridad que reciba como parámetros la carpeta origen y la carpeta destino. Los argumentos que recibe un batch, son recibidos de la siguiente forma:

nombre_script argumento1 argumento2 argumento3

siendo:

%0 = nombre del archivo
%1 = argumento1
%2 = argumento2
%3 = argumento3

Podemos borrar el contenido de un parámetro con el comando SHIFT.

2.2 Variables de entorno

Para ver todas las variables de entorno registradas en tu sistema teclea **set** en tu consola.

El resultado puede ser parecido al siguiente:

```
Microsoft Windows XP [Versión 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.



C:Documents and Settingsjavi>set
ALLUSERSPROFILE=C:Documents and SettingsAll Users.WINDOWS
APPDATA=C:Documents and SettingsjaviDatos de programa
CLASSPATH=.;C:Archivos de programaJavaj2re1.4.2libextQTJava.zip
CommonProgramFiles=C:Archivos de programaArchivos comunes
COMPUTERNAME=-JAVI-PORTATIL
ComSpec=C:WINDOWSsystem32cmd.exe
FP_NO_HOST_CHECK=NO
HOMEDRIVE=C:
HOMEPATH=Documents and Settingsjavi
LOGONSERVER=JAVI-PORTATIL
NUMBER_OF_PROCESSORS=1
OS=Windows_NT
Path=C:WINDOWSsystem32;C:WINDOWS;C:WINDOWSSystem32Wbem;C:Archivos de prog
ramaQuickTimeQTSystem;C:Archivos de programaArchivos comunesAdobeAGL
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH
PROCESSOR_ARCHITECTURE=x86
PROCESSOR_IDENTIFIER=x86 Family 6 Model 9 Stepping 5, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=0905
ProgramFiles=C:Archivos de programa
PROMPT=$P$G
QTJAVA=C:Archivos de programaJavaj2re1.4.2libextQTJava.zip
SESSIONNAME=Console
SystemDrive=C:
SystemRoot=C:WINDOWS
TEMP=C:DOCUME~1javiCONFIG~1Temp
TMP=C:DOCUME~1javiCONFIG~1Temp
USERDOMAIN=JAVI-PORTATIL
USERNAME=javi
USERPROFILE=C:Documents and Settingsjavi
windir=C:WINDOWS
```

Por ejemplo, la variable **COMSPEC** nos indica la ubicación del interprete de comandos DOS, que suele ser command.com (**C:WINDOWSsystem32cmd.exe**).
En **PATH** se guardan las rutas donde DOS debe buscar comandos o programas a ejecutar y en **PROMPT** es el símbolo de sistema que indica al usuario que puede introducir texto. En nuestro caso se muestra la ruta donde se encuentra y el signo de mayor que.

2.3 Ejemplo

Para empezar haremos un script muy muy sencillo. Pongámonos en la situación de que somos administradores y queremos que se haga una copia de seguridad de ciertas carpeta. Pues esta tarea es tan sencilla como crear un archivo ejemplo.bat con el siguiente contenido.

```
;@echo off REM Ejemplo1
XCOPY %SystemDrive%carpetaOrigen D:copiaSeguridad /E /I
exit
```

Notas

- Si lo que quisieramos copiar fuera un archivo, en vez de usar XCOPY usaríamos **COPY**.
- La variable de entorno **SystemDrive**, nos devuelve la letra del disco duro. No sería necesario usar esta variable si conocemos la ruta, pero he considerado didáctico introducirlo en este ejemplo.
- La opción /**E** es para especificar que deseo copiar directorios y subdirectorios, incluyendo los que estan vacíos.
- La opción /**I** es para especificar que queremos crear un subdirectorio si no existe, ya que si no lo pones, nos preguntaría.
- Para mas información acerca de las opciones , teclear **XCOPY /?**.

2.4 Tareas programadas

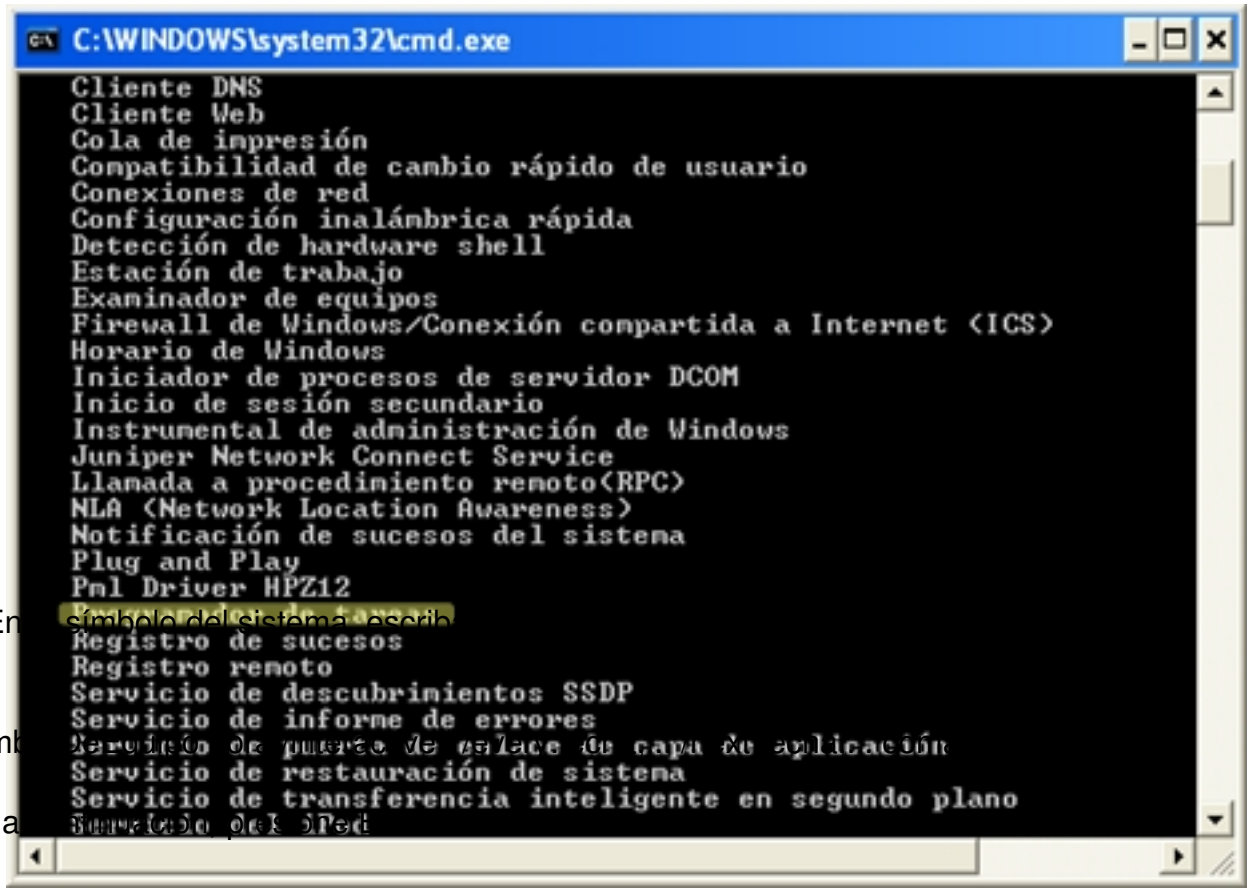
Ahora si queremos que este proceso se repita cada cierto tiempo, tenemos que entender el funcionamiento del comando **at**, que utiliza la siguiente sintaxis:

at
ombreDeEquipo hora /interactive | /every:fecha,... /next:fecha,... comando

Tendríamos que seguir los siguientes pasos para crear nuestra tarea programada:

1. Ir a **Inicio**/**Ejecutar** y escriba cmd. En la consola que se abre, escriba **net start,** para ver el listado con los servicios que se estan ejecutando actualmente.

2. Si el Programador de tareas no aparece en nuestra lista, tecleé net start "task scheduler".



3. En símbolo del sistema escrib

at

omb

y, a

Por ejemplo, si quisiesemos que se ejecutase en nuestro equipo a una hora:

```
C:Documents and Settingspepito>at 11:20 c:/ejemplo.bat
Se ha agregado un nuevo trabajo con identificador = 1
```

Si queremos ver las tareas pendientes:

```
C:Documents and Settingspepito>at
Estado ID Día Hora Línea de comando
-------------------------------------------------------------------------
1                 Hoy 11:20 c:/ejemplo.bat
```

Si quisieramos eliminar esa tarea:

at
ombreDeEquipo id /delete | /delete/yes

En nuestro caso, si tenemos nuestra tarea con el identificador 1.

at 1 /delete

Para mas información sobre el comando at, visitar http://support.microsoft.com/kb/313565/es

También podemos visitar esta extensa colección de ejemplos.

3. Shell de LINUX. Historia y Conceptos básicos

La Shell permite al usuario interactuar con el Kernel a través de la interpretación de los comandos que el usuario ingresa en la línea de comandos (ó a través de los "scripts", archivos que ejecutan un conjunto de comandos).

Si quieres saber qué Shells tienes instalados en tu distribución GNU/Linux sólo tienes que teclear el comando:

cat /etc/shells ó **cat /etc/shell** (dependiendo de la distribución que tengas instalada).

En mi caso, la salida de este comando es la siguiente:

```
# /etc/shells: valid login shells
/bin/ash
/bin/bash
/bin/csh
/bin/sh
/usr/bin/es
/usr/bin/ksh
/bin/ksh
/usr/bin/rc
/usr/bin/tcsh
/bin/tcsh
/usr/bin/zsh
/bin/sash
/bin/zsh
/usr/bin/esh
/bin/rbash
/usr/bin/screen
```

Para saber que tipo de Shell estamos usando, tecleamos el siguiente comando en una consola de Linux:

```
root@ubuntu:~# echo $SHELL
/bin/bash
```

Para cambiar el tipo de Shell que se esta usando usaremo el comando:

```
root@ubuntu:~# sh
```

Es importante también saber que un script puede ejecutarse en primer o en segundo plano:

-

- **Primer plano (foreground)**: se lanza introduciendo el nombre del script, pero hasta que el script no termine no se devolverá el control de la shell al usuario.

- **Segundo plano (background)**:se lanza igual pero añadiendo a continuación &. El control de la shell pasa inmediatamente al usuario(normalmente usado para procesos de larga duración)

3.1 Historia de las Shells

La primera Shell fue programada por Steven Bourne (llamada por este motivo Bourne-Shell). Según las versiones se le llama **sh** o **bsh**. Es una shell muy limitada, y usa una sintáxis de comandos usada en los primeros sistemas UNIX.

Cronológicamente, la siguiente shell fue la c-shell o **csh**, desarrollada por el unix BSD de Berkeley. Y cómo su nombre indica, usa comandos muy parecidos al lenguaje de programación C. También existe una shell llamada
tcsh
, que es una especie de c-shell mejorada, siendo capaz de chequear ficheros, completar nombres de variables, aliases. No suele venir incluida en las instalaciones estándar.

En 1986, David Korn en los laboratorios AT&T programó la korn- shell (**ksh**) que nació de juntar lo mejor de la bourne shell y la c-shell.

En la mayoría de los sistemas Linux, viene por defecto la shell **bash** (Bourne-Again-shell , en referencia al inventor de la primera Shell). Esta Shell posee toda la funcionalidad del sh con características avanzadas de C Shell, por eso cualquier script escrito para una shell sh correrá perfectamente. La shell bash fué programada por desarrolladores del proyecto GNU . La ventaja principal es su potencia y que es gratuita. Por este motivo nosotros usaremos esta shell .

3.2 Conceptos Básicos

Para comenzar a introducirnos en el mundo de los scripts, recomendamos antes la lectura de estos artículos:

- Editor VI , ya que necesitaremos manejarnos con este editor para escribir nuestros scripts...
- La estructura del sistema de archivos en Linux , imprescindible si no se conoce el sistema de archivos de Linux.
- Introducción a la Shell de Linux , dónde se hace un breve repaso a los comandos más básicos de Linux

Antes de comenzar con ejemplos de Scripts, vamos a explicar una serie de conceptos, necesarios a la hora de implementar nuestros shell-scripts:

3.2.1 Variables

Para asignar un valor a una variable:

```
valor=7
```

Para visualizar el contenido de la variable

```
root@ubuntu:~# echo $valor
7
```

Para borrar el contenido

```
root@ubuntu:~# set valor
```

```
root@ubuntu:~# echo $valor
```

A continuación se muestra una tabla con una serie de variables especiales que nos serán de

utilidad a la hora de escribir nuestros scripts:

VARIABLE	DESCRIPCION
$0	Nombre del Shell-Script que se está ejecutando.
$n	Parámetro o argumento pasado al Shell-Script en la posición n, n=1,2,...

$PS1	Prompt
$#	Número de argumentos.
$*	Lista de todos los argumentos.
$?	Salida del último proceso ejecutado.
$$	Número de identificación del proceso (PID)
$!	Número del último proceso invocado por la shell

3.2.2 Variables de entorno

Existen dos áreas de memoria en las shells para almacenar variables, el Área local de datos y el Entorno. Podemos visualizar su contenido de ambas áreas, al igual que Windows, con el comando **set**.

Por defecto, cuando asignamos un valor a una variable, es local, es decir, es conocida por esa shell, pero si se abre otra shell a partir de la que estamos, estas nuevas 'subshells' desconocen el valor de las variables que hemos asignado anteriormente.

En cambio, las variables del entorno están disponibles para las subshells. Es decir los valores de estas variables son conocidos por los procesos hijos de la shell.

Para hacer que una variable se almacene en el área de Entorno, se utiliza el siguiente comando:

```
root@ubuntu:~# export nombre=javi
```

Para ver la lista de variables del entorno usaremos el comando **env**.

Debemos saber que una variable exportada **NO** es lo mismo que una *variable global*, sino una copia, ya que podrá ser modificada pero volverá a tener su anterior valor cuando salga de la subshell.

Cómo ya sabemos, existe un conjunto de variables de entorno predefinidas, que a continuación listamos en esta tabla:

COMANDO	DESCRIPCION
PATH	Camino de búsqueda de órdenes
HOME	Directorio de trabajo del usuario
USER	Usuario que estableció la sesión
PWD	Ruta completa del directorio de trabajo actual
LOGNAME	Nombre del usuario que ejecuta la shell
TERM	Tipo de terminal.
SHELL	Shell que está ejecutándose
PS1, PS2, PS3, PS4	Prompts

3.2.3 Entrecomillado

Debido a que la shell bash puede usar nombres simbólicos que representan valores, como el path del directorio personal, necesitaremos conocer la diferencia entre los distintos tipos de entrecomillado, el simple, el doble y el uso de las sencuencias de escape ().

Entrecomillado simple

Nos sirve para hacer que la shell tome lo encerrado entre él como una expresión literal, ya

sean variables, comodines u otras expresiones entrecomilladas.
Es decir el contenido no es interpretado por la shell.

Para entenderlo mejor, imaginemos que deseamos crear una carpeta de nombre * (asterisco). Si lo intentamos, es decir si tecleamos **mkdir ***, nos dará error. Sin embargo si escribimos **mkdir '*** ',
el intérprete de comandos tomará el carácter comodín (*) como un literal y no como un comodín, y podremos crear la carpeta con ese nombre.

Entrecomillado doble

Permiten la sustitución de parámetros, la sustitución de comandos y la evaluación de expresiones aritméticas, pero ignoran los caracteres tubería, sustituciones de tilde, expansión de caracteres comodines, aliases y la división de palabras vía delimitadores.

Las comillas simples dentro de las comillas dobles no tienen efecto. Se puede incluir comillas dobles dentro de una cadena con comillas dobles anteponiendo .

Es decir, se produce sustitución de variable (el signo del dolar se interpreta) por ejemplo, podremos ejecutar **ls "$BASH**" y nos devolverá correctamente el tipo de shell (/bin/bash), pero si hacemos **ls "*"**, el comodin será tomado como un literal.

Sencuencias de escape ()

Una barra inclinada inversa no entrecomillada, es decir **,** preserva el valor literal del siguiente carácter que lo acompaña.

Para entenderlo mejor, veamos un ejemplo:

```
root@ubuntu:~# echo $BASH
/bin/bash
```

Si utilizamos la encuencias de escape:

```
root@ubuntu:~# echo $BASH
$BASH
```

Veamos otros ejemplos:

root@ubuntu:~# **usuario=javi**

root@ubuntu:~# **echo $usuario**

javi

root@ubuntu:~# **echo "$usuario"**

javi

root@ubuntu:~# **echo '$usuario'**

$usuario

root@ubuntu:~# **echo $usuario**

$usuario

root@ubuntu:~# **echo "'$usuario'"**

'javi'

A continuación se muestra una tabla para que fácilmente veamos la diferencia entre un tipo de entrecomillado y otro:

EXPRESIÓN	VALOR
$usuario	javi
"$usuario"	javi
'$usuario'	$usuario
$usuario	$usuario

" '$usuario' "	'javi'

3.2.4 Ejecución

Existe dos formas de ejecutar los scripts:

-
- Anteponiendo sh, source o bien "." al nombre del script.

root@ubuntu:~# **sh ejemplo.sh**

esto es un ejemplo

root@ubuntu:~# **source ejemplo.sh**

esto es un ejemplo

root@ubuntu:~# **./ejemplo.sh**

esto es un ejemplo

-

-

- Dando permiso de ejecución y a continuación, invocándolo con su nombre anteponiendo

la ruta donde se encuentra el script-:

root@ubuntu:~# **chmod +x ejemplo.sh**

root@ubuntu:~# **./ejemplo.sh**

-

-

Si nos encontramos en el mismo directorio que el script-:

root@ubuntu:~# **./ejemplo.sh**

esto es un ejemplo

-

-

Sino pondremos la ruta, por ejemplo:

root@ubuntu:/# **/root/ejemplos/ejemplo.sh**

esto es un ejemplo

En caso de que el directorio donde se encuentra el script este en el *PATH*, se podr ía ejecutar introduciendo simplemente el nombre.

3.2.5 Parámetros

Un shell-script soporta argumentos o parámetros de entrada. Desde el interior del script se referencian mediante las variables especiales $i con i=1, 2, ..., NumeroArgumentos. Es decir, si lanzamos el siguiente script-:

root@ubuntu:~# **ejemploScript uno dos gato**

Tendremos que el primer parámetro es uno, el segundo dos y el tercero gato, referenciándolos en el script como $1, $2 y $3 respectivamente. Recordemos como ya vimos anteriormente que la variable $0 hace referencia al nombre del script.

También podemos escribirlos en la shell con el comando "set":

```
root@ubuntu:~# set parametro1 parametro2 parametro3
root@ubuntu:~# echo $2
parametro2
```

Con el comando "shift", podemos desplazar, es decir borrar varias posiciones:

```
root@ubuntu:~# set parametro1 parametro2 parametro3
root@ubuntu:~# shift 2
root@ubuntu:~# echo $1
parametro3
```

Con "shift 2" hacemos que desaparezca el valor de $1 y $2, y que ahora $1 valga lo que estaba en $3.

3.2.6 Automatización de tareas con Crontab

Crontab es una herramienta cuya función es la automatización de tareas. Por ejemplo podemos apagar un equipo a una hora determinada o realizar backups de manera automática.

Para ver qué tareas tenemos en el crontab, deberemos ejecutar:

```
root@ubuntu:~# crontab -l
```

Veamos un ejemplo que ejecutaría el comando who todos los martes a las o-nce y media de la noche, guardando la salida en fichero.txt::

30 23 * * 2 /usr/bin/who >> /home/fichero.txt

El formato de las entradas de crontab es el siguiente:

[minutos] [hora] [día] [mes] [dia_de_semana] [comando]

Teniendo en cuenta que:

- Minutos: (0-59): Es el minuto exacto en el que quieres que se ejecute la tarea

- Hora: (0-23) La hora exacta en formato de 24 horas

- Día: (1-31) Valor numérico del día del mes

- Mes: (1-12) Valor numérico del mes

- Día de la semana: (0-6), siendo 1=Lunes, 2=Martes,... 6=sábado y 0=Domingo: Valor numérico del día de la semana

- Usuario: usuario que ejecuta el comando, sino se pone, se usa root por defecto

- Comando: comando a lanzar

Para especificar todos los valores posibles de una variable se utiliza el asterisco (*).

Para agregar, quitar o modificar tareas en el crontab usaremos:

```
root@ubuntu:~# crontab -e
```

que abrirá el editor (el que hayamos definido en la variable de entorno $EDITOR) y cargará el archivo **crontab** correspondiente al usuario que está logueado.

Además con los siguientes comandos también podremos:

- crontab -r: elimina el fichero crontab del usuario.

- crontab -u usuario: aplica una de las opciones anteriores para un usuario determinado.

- crontab fich: instala el fichero fich como crontab del usuario.

Ahora veamos un ejemplo de cómo ejecutar nuestro script cada 20 minutos y que la salida se guarde en un log:

20 * * * * /home/script.sh >> mylog.log

Para continuar viendo este tutorial vaya al siguiente enlace:

Tutorial Shell Scripts II

.