Tema: 11 Introducción a scripting y tareas programadas.

1. Introducción.

Un archivo de comandos es un archivo de procesamiento por lotes. Se trata de archivos de texto sin formato, guardados con la extensión que contienen un conjunto de comandos del sistema operativo con el que trabajamos. Cuando se ejecuta este archivo, (Desde la consola de usuario) los comandos contenidos son ejecutados en grupo, de forma secuencial, permitiendo automatizar diversas tareas. Cualquier comando del sistema operativo se puede utilizar.

Ésta es la forma de automatizar procesos (copiar, pegar, renombrar y enviar datos). De este modo, evitamos procesos rutinarios y monótonos, acelerando los mismos. Tiene la funcionalidad de conectarse con otras interfaces por línea de comandos.

Estos archivos permiten el paso de parámetros, y su programación con sentencias básicas de programación. Con la programación de estos archivos se pueden desarrollar programas muy completos.

Antiguamente en Windows, estos archivos eran poco usados y se llaman archivos batch o bat. En Linux, un sistema operativo destinado a sistemas siempre se ha utilizado ampliamente. En Linux se llaman script o bash. Microsoft se percató de su debilidad como sistema operativo de sistemas, y desde 2006 lanzo una potente consola de administración llamada powershell. Hoy día esta consola también está instalada es las versiones de Desktop de Microsotf.

2. Powershell.

Por seguridad la ejecución de scripts se encuentra deshabilitada, de esta forma evitamos que, por accidente, ejecutemos código malicioso en nuestro sistema. Para permitir un ajuste fino de la ejecución de scripts, el sistema provee varios modos de ejecución:

- Directiva AllSigned: esta directiva requiere que todos los scripts y archivos de configuración estén firmados por un editor de confianza, incluidos los scripts que se escriban en el equipo local. Además, requiere confirmación cuando se ejecuta u script. El editor de confianza debe ser añadido como tal a través del editor de certificados. Para obtener más información podemos teclear en la consola Get-Help About_Signing.
- Directiva RemoteSigned: esta directiva requiere la firma digital de un editor de confianza en scripts y archivos de configuración descargados de Internet, pero no requiere firmas digitales en scripts ejecutados desde el equipo local. Si el script viene de un editor de confianza, no pide confirmación para ejecutarlo.
- La directiva Unrestricted: esta directiva permite ejecutar los scripts sin firma. Tan solo preguntará cuando ejecutemos un script descargado a través de Internet.

De todas las opciones las recomendadas para poder trabajar son AllSigned y RemoteSigned. ¿Cuál tiene nuestro equipo?

PS C:\> Get-ExecutionPolicy

AllSigned

El sistema sólo deja ejecutar los scripts que vengan firmados por editores de confianza, cosa que no somos. Podemos cambiar la directiva a una un poco menos restrictiva:

PS C:\> Set-ExecutionPolicy RemoteSigned

Windows PowerShell utiliza un sistema de nombres con la estructura "verbo- sustantivo": el nombre de cada cmdlet consta de un verbo estándar y un sustantivo concreto unidos por un guion. Los verbos de Windows PowerShell expresan acciones concretas en Windows PowerShell. Los sustantivos son muy parecidos a los de cualquier idioma, ya que describen tipos de objetos concretos que son importantes para la administración del sistema.

Los sustantivos están menos limitados, pero deben describir siempre a qué se aplica un comando. Windows PowerShell incluye comandos como Get-Process, Stop-Process, Get-Service y Stop-Service.

A menudo se reconoce la función de un comando con sólo leer su nombre, y suele ser evidente el nombre que debe utilizarse para un comando nuevo. Por ejemplo, un comando que apaga el equipo podría ser Stop-Computer. Un comando que enumera todos los equipos de una red podría ser Get-Computer. El comando que obtiene la fecha del sistema es Get-Date.

Puede obtener una lista de todos los comandos que incluyen un verbo concreto con el parámetro -Verb de Get-Command. Por ejemplo, para ver todos los cmdlets que utilizan el verbo Get, escriba

PS> Get-Command -Verb Get

La notación utilizada para las canalizaciones es parecida a la que se utiliza en otros shells por lo que, a primera vista, puede no ser evidente que Windows PowerShell introduce novedades. El carácter utilizado es el pipe (tubería): |

```
PS> Get-Process | Get-Member | Out-Host -Paging
```

Windows PowerShell trabaja con objetos. Permite crear variables, que son básicamente objetos con nombre, para almacenar los resultados y poder utilizarlos más adelante. Si está acostumbrado a trabajar con variables en otros shells, recuerde que las variables de Windows PowerShell son objetos, no texto.

Las variables se especifican siempre con el carácter inicial $ y pueden contener cualquier carácter alfanumérico o el carácter de subrayado en sus nombres.

```
$loc = Get-Location
```

2.1. Comparadores

Los más importantes son:

- -eq igual
- -ne no igual
- -gt mayor que
- -ge mayor o igual que
- -lt menor que
- -le menor o igual que
- -like comparación de caracteres comodín
- -notlike lo contrario que el anterior.
- -match que coincida el segundo operando exactamente
- -notmatch lo contrario de “match”

2.2. Estructuras de control

El idioma de Windows PowerShell es simplificado, aunque es suficiente para realizar el trabajo.

2.2.1. La construcción if

Ésta es la construcción de toma de decisiones principal de Windows PowerShell:

```
If ($this -eq $that) {
        # commands
} elseif ($those -ne $them) {
        # commands
} elseif ($we -gt $they) {
        # commands
} else {
        # commands
}
```

La palabra clave "If" es una parte obligatoria de esta construcción. Una expresión entre paréntesis siguiente que se debe evaluar Verdadero o falso. Windows PowerShell siempre se interpretarán cero como False y cualquier valor distinto de cero como True.

Windows PowerShell también reconoce las variables integradas $True y $False que representa los valores booleanos. Si la expresión entre paréntesis funciona en True, se ejecutarán los comandos en el siguiente conjunto de llaves. Si la expresión es False, no ejecutan los comandos. Y eso es todo lo que necesita para un válido si construir.

Puede ir un poco más lejos al proporcionar una o varias secciones de "Elself" y finalmente un bloque Else, que se ejecutará si ninguno de los bloques anteriores se ejecuta.

El carácter # es un carácter de comentario, Windows PowerShell omite cualquier cosa detrás de ese carácter y hasta el final de la línea.

2.2.2. La construcción while

Se trata de una construcción de bucle en Windows PowersShell. Se ha diseñado para repetir un bloque de comandos mientras alguna condición es verdadera o hasta que una condición sea verdadera. Esto da lugar a dos bucles ya conocidos:

```
Do {
        # commands
} While ($this -eq $that)
```

y

```
While (Test-Path $path) {
        # commands
   }
```

Observe el segundo ejemplo: no utiliza un operador de comparación como -eq. Eso es porque se pasa el cmdlet Test-Path que devolverá verdadero o falso según se cumpla o no la condición.

2.2.3.La construcción ForEach

El propósito de ForEach es coger un array (o una colección, que en Windows PowerShell es el mismo que una matriz) y enumerar sus objetos para poder trabajar con ellos de uno en uno:

```
$services = Get-Service
ForEach ($service in $services) {
        $service.Stop()
}
```

3. Script en Linux.

Los archivos *script* suelen ser identificados por el sistema a través uno de los siguientes encabezamientos en el contenido del archivo, conocido como SheBang.

```
#!/bin/bash ;
#!/bin/ksh ;
#!/bin/csh ;
```

3.1. if-else

Permite seleccionar entre opciones, dependiendo del valor de una o más variables

```
if [$num -eq 1] then
        echo "la variable num contiene el valor 1"
else
        echo "la variable num no es 1"
fi
```

3.2. for

Repite tantas veces como elementos hay en la cadena de elementos

```
for num in {1..10}
do
        echo "contador: $num"
done
```

3.3. while

```
#!/bin/bash
CONTADOR=0
 while [  $CONTADOR -lt 10 ]; do
       echo El contador es $CONTADOR
       let CONTADOR=CONTADOR+1
 done
```

Ejemplo de script:

```
# comentario programa de suma
echo “soy Gnu/Cal”
echo “Tu me dices lo que quieres calcular y yo te doy el resultado”
echo “Introduce el primer valor”
read valor1
echo “ Introduce el operador. Puedes escoger entre: + - * /”
read operador
```

```
echo " Introduce el segundo valor"
read valor2
echo "El resultado es:"
sleep 2
expr $valor1 $operador $valor2
sleep 1
echo "gracias por su tiempo!"
```

Las tres primeras líneas son, por así decirlo, de presentación. En la cuarta línea hay un read, llamado valor1. Es una variable, que va a introducir desde el teclado el usuario. Después, es necesario saber que operación se quiere realizar (suma, resta, multiplicación o división). Para ello hay un nuevo read, llamado "operador" para que el usuario pueda escoger. Después, tenemos el segundo valor de la operación llamado "valor2". El comando "sleep" lo único que hace es esperar un poco de tiempo para que de la sensación de que el programa esta pensando poco antes de dar el resultado. La operación matemática propiamente dicha se realiza en la línea 11.

Una buena costumbre cuando definamos variables en Bash es utilizar letras mayúsculas, esto no es necesario, pero nos ayudara a tener un script más fácil de entender. La variable se define y se le asigna un valor.

```
MENSAJE="Hola Mundo"
echo $MENSAJE
```

En los scripts también se pueden utilizar parámetros, los aquí se muestran su valor con símbolo $. Así tenemos:
$0: es el nombre del archivo que se ejecuta.
$1 a $9: es el nombre de parámetro correspondiente.
$*: Son todos los parámetros.

4. Filtros.

Un filtro es un programa que recibe una entrada, la procesa y emite una salida.

Son una parte fundamental del sistema ya que relacionando filtros simples se puede conseguir prácticamente cualquier cosa, a nivel de administración de archivos y scripts.

Algunos ejemplos de filtros en Linux/UNIX son:

- para seleccionar una parte de la entrada: grep, tail, head y cut.
- para ordenar la entrada: sort.
- para alterar la entrada en forma ordenada: tr.
- para comparar dos entradas: comm y diff.
- para contar elementos de la entrada: wc.

Una expresión regular es un patrón que define a un conjunto de cadenas de caracteres. Las expresiones regulares se construyen de forma análoga a las expresiones aritméticas. Existe la posibilidad de combinar expresiones simples; para ello, debemos emplear distintos operadores.

Los caracteres ^ y $ son metacaracteres que representan una cadena vacía al principio y al final de la línea, respectivamente. Los símbolos \< y \> representan una cadena vacía al principio y al final de una palabra.

Una expresión regular que representa un carácter sencillo puede ser continuada con uno o varios caracteres de repetición:

?	El elemento precedente es opcional y debe coincidir al menos una vez (0 o 1)
*	El elemento precedente debe coincidir cero o más veces (0 a n)
{n}	El elemento precedente debe coincidir exactamente n veces
+	El elemento precedente debe coincidir una o más veces. (1 a n)
{m,}	El elemento precedente es opcional y debe coincidir al menos m veces
{,m}	El elemento precedente es opcional y debe coincidir a lo sumo m veces.
{n,m}	El elemento precedente debe coincidir al menos n veces pero no más de m veces.

Dos expresiones regulares pueden unirse con el operador |. La expresión resultante representa cualquier cadena que responda a uno de los dos patrones.

Patrón	Qué representa
pablo	La cadena pablo
^pablo	La cadena pablo al comienzo de una línea.
pablo$	La cadena pablo al final de una línea.
^pablo$	La cadena pablo formando una única línea.
niñ[oa]	La cadena niño o niña
ni[^aeiou]o	La tercera letra no es una vocal minúscula.
ga.o	La tercera letra es cualquier carácter.
^....$	Cualquier línea que contenga 4 caracteres.
^\.	Cualquier línea que comienza por punto.
^[^.]	Cualquier línea que no comienza por punto.
niños*	niño, niños, niñoss, niñosss, etc
“niño”	niño entre comillas dobles.
“*niño”*	niño con o sin comillas dobles.
[a-z][a-z]*	una o más letras minúsculas.
[a-z]+	una o más letras minúsculas (sólo válido en algunas aplicaciones).
[^0-9A-Z]	cualquier carácter que no sea ni número ni letra mayúscula.
[a-zA-Z]	cualquier letra sea mayúscula o minúscula.
[Ax5]	cualquier carácter que sea A, x o 5.
niño|niña|nada	una de las tres palabras.
(s|arb)usto	la palabra susto o arbusto.
\<ga	cualquier palabra que empiece por ga.
ño\>	cualquier palabra que termine por ño
\<niño\>	la palabra niño
o\{2,\}	dos o más oes en una misma fila.

Siempre que empleemos expresiones regulares con algún filtro como grep, deben ser encerradas entre comillas dobles para que el intérprete de órdenes no los considere. Si dentro de la expresión regular tenemos el metacarácter $, deberemos emplear comillas simples en lugar de las comillas dobles.

4.1. grep

El comando grep nos permite buscar, dentro de los archivos, las líneas que concuerdan con un patrón.

Si no especificamos ningún nombre de archivo, tomará la entrada estándar, con lo que podemos encadenarlo con otros filtros. Si no se dice otra cosa, grep muestra las líneas que concuerden.

La sintaxis de este comando es la siguiente:

grep [opciones] patrón [fichero ...]

Por defecto, grep imprime las líneas encontradas en la salida estándar. Es decir, que podemos verlo directamente la pantalla, o redireccionar la salida estándar a un archivo.

Algunas de las opciones utilizadas en grep son las siguientes:

-c: en lugar de imprimir las líneas que coinciden, muestra el número de líneas que coinciden.

-e PATRON: nos permite especificar varios patrones de búsqueda o proteger aquellos patrones de búsqueda que comienzan con el signo -.

-r: busca recursivamente dentro de todos los subdirectorios del directorio actual.

-v: nos muestra las líneas que no coinciden con el patrón buscado.

-i: ignora la distinción entre mayúsculas y minúsculas.
-n: numera las líneas en la salida.
-E: nos permite usar expresiones regulares.
-o: le indica a grep que nos muestre sólo la parte de la línea que coincide con el patrón.
-f ARCHIVO: extrae los patrones del archivo que especifiquemos. Los patrones del archivo deben ir uno por línea.
-H: nos imprime el nombre del archivo con cada coincidencia.

Veamos algunos ejemplos:

- Buscar todas las palabras que comiencen por a en un archivo:

  $ grep "a*" archivo

Otra forma de buscar, sería:

  $ cat archivo | grep "a*"

- Mostrar por pantalla, las líneas que contienen comentarios en el archivo /boot/grub/menu.lst:

  $ grep "#" /boot/grub/menu.lst

- Mostrar las líneas del archivo /boot/grub/menu.lst que no son comentarios:

  $ grep -v "#" /boot/grub/menu.lst

- Contar el número de interfaces de red que tenemos definidos en el fichero /etc/network/interfaces:

  $ grep -c "iface" /etc/network/interfaces

- Mostrar las líneas de un fichero que contienen la palabra BADAJOZ o HUELVA:

  $ grep -e "BADAJOZ" -e "HUELVA" archivo

- Mostrar las líneas de un fichero que contienen la palabra BADAJOZ o HUELVA, numerando las líneas de salida:

  $ grep -n -e "BADAJOZ" -e "HUELVA" archivo

- Mostrar los ficheros que contienen la palabra TOLEDO en el directorio actual y todos sus subdirectorios:

  $ grep -r "TOLEDO" *

  Veamos algunos ejemplos con expresiones regulares:

- Obtener la dirección MAC de la interfaz de red eth0 de nuestra máquina:

  $ ifconfig eth0 | grep -oiE '([0-9A-F]{2}:){5}[0-9A-F]{2}'

  Se extrae la dirección MAC de la interfaz eth0 de nuestra máquina haciendo un: ifconfig eth0, y después se utiliza el filtro. La expresión regular, se puede dividir en dos partes:

  [0-9A-F]{2}:){5} para buscar 5 conjuntos de 2 caracteres seguidos de dos puntos,
  [0-9A-F]{2} seguido por un conjunto de dos caracteres.

  Como las direcciones MAC se representan en hexadecimal, los caracteres que buscamos son los números del 0 al 9 y las letras desde la A a la F.

- Extraer la lista de direcciones de correo electrónico de un archivo:

  grep -Eio '[a-z0-9._-]+@[a-z0-9.-]+[a-z]{2,4}' fichero.txt
  Analicemos la expresión regular:
  [a-z0-9._-]+@[a-z0-9.-]+[a-z]{2,4}
  Al igual que antes, la vamos dividiendo en partes:
  [a-z0-9._-]+ Una combinación de letras, números, y/o los símbolos '.' '_' y '-' de uno o más caracteres @ seguido de una arroba
  [a-z0-9.-]+ Seguido de una cadena de letras, números y/o los símbolos '.' y '-'
  [a-z]{2,4} seguido de una cadena de entre dos y cuatro caracteres.

- Obtener la dirección IP de la interfaz de red eth1 de nuestra máquina:

```
$ ifconfig eth1 | grep -oiE '([0-9]{1,3}\.){3}[0-9]{1,3}' | grep -v 255
```

En el ejemplo anterior, hemos tomado la información que nos ofrece ifconfig. Hemos filtrado dicha información con el comando grep, obteniendo todas las direcciones IP que aparecen:

```
grep -oiE '([0-9]{1,3}\.){3}[0-9]{1,3}'
```

Por último, hemos filtrado la salida del comando anterior, para eliminar la dirección de broadcast junto con la máscara de red para quedarnos sólo con la dirección IP de la máquina:

```
grep -v 255
```

La línea anterior no mostraría las líneas que no contengan el valor 255, es decir, las direcciones de broadcast y máscara de red. Analicemos ahora el comando grep:

```
grep -oiE '([0-9]{1,3}\.){3}[0-9]{1,3}'
```

En cuanto a la expresión regular:

```
'([0-9]{1,3}\.){3}[0-9]{1,3}'
```

([0-9]{1,3}\.){3} Representa 3 bloques de entre uno y tres dígitos separados por puntos. Observemos que como el punto es un metacarácter, tengo que usar el carácter de escape \ para que no sea interpretado como un metacarácter, sino como un carácter normal.

[0-9]{1,3} Representa el último bloque de la dirección IP, que está formado por un número de entre 1 y 3 dígitos.

NOTA: egrep y fgrep están actualmente en desuso y en su lugar se utiliza el equivalente grep –E y grep –F, respectivamente.

4.2. sort

Nos permite ordenar los registros o líneas de uno o más archivos. La ordenación se puede hacer por el primer carácter, por el primer campo de la línea o por un campo distinto al primero en el caso de ficheros estructurados.

Podemos ordenar el contenido de un fichero de la siguiente manera:

```
sort fichero
```

Se realizaría la ordenación y el resultado se mostraría por pantalla. Así que, si lo que queremos es obtener el resultado de la ordenación en un fichero, haríamos:

```
sort fichero > ficheroordenado
```

Si lo que queremos es ordenar varios ficheros y añadir el resultado a otro, podemos indicar varios ficheros en la línea de entrada:

```
sort fichero1 fichero2 > fichero3
```

Y si lo que queremos es ordenar un fichero y dejar el resultado de la ordenación en el mismo fichero, podemos hacerlo con el parámetro -o (output):

```
sort -o f1 f1
```

Veamos una lista de los parámetros que pueden resultarnos más útiles a la hora de usar este comando:

- -f: este parámetro nos sirve para indicar que las mayúsculas y las minúsculas se van a tratar de forma diferente y que por tanto se va a seguir un ordenamiento alfabético.
- -n: este parámetro nos sirve para ordenar los campos numéricos por su valor numérico.
- -r: nos permite realizar una ordenación inversa, es decir, de mayor a menor.
- -k número: de este modo especificaremos por qué columna o campo vamos a realizar la ordenación en las versiones más recientes de Linux.
- --field-separator = separador: normalmente, se usa como delimitador de campos el espacio en blanco. Podemos utilizar el parámetro --field-separator para indicar que vamos a usar otro delimitador de campo cualquiera. Ej: --field-separator=. La

opción abreviada de --field-separator es -t.

- -u: nos permite suprimir todas las líneas repetidas después de realizar la ordenación.

Y algunos ejemplos con dichos parámetros:

Obtener un listado de los ficheros del directorio actual, ordenado por tamaño de archivo:

$ ls -l | sort –k5

Obtener un listado de los ficheros del directorio actual, ordenado de mayor a menor por tamaño de archivo:

$ ls -l | sort -r -k5

Obtener un listado de los ficheros del directorio actual, ordenado por nombre del archivo:

$ ls -l | sort –k9

Ordenar un fichero eliminando las líneas repetidas:

$ sort -u fichero

Ordenar un fichero en el que los campos están separados por comas, por el campo número 3:

$ sort -t, -k3

El siguiente es un ejemplo en el que se ordena usando la sintaxis actual para ordenar por columnas: imaginemos que queremos ver un listado de usuarios del fichero /etc/passwd ordenado por uid:

$ cat /etc/passwd | sort -t":" -k3n

Con -k3 le indicamos a sort que queremos ordene por la columna 3. Y, al añadir la opción –n se le indica que ordene por orden numérico.

Un ejemplo para eliminar las líneas repetidas de un archivo y dejar el contenido en el mismo archivo:

$ sort -o fichero -u fichero

4.3. tr

Es un filtro que nos permite cambiar una determinada información de un archivo por otra.

Cambia cada uno de los caracteres especificados en el conjunto inicial por los caracteres especificados en el conjunto final. El fichero de origen o fichero destino lo especificamos con los caracteres de redirección: < ó >.

Veamos unos ejemplos:

tr ':' ' ' < /etc/passwd > ficheropasswd
tr '[a-z]' '[A-Z]' <> listaalumnosmayusculas tr ' ' '\n' <> lineasusuarios
tr -s " " <> prueba2

Parámetros útiles:

- -s: sustituye un conjunto de caracteres repetidos por uno sólo. Es muy útil cuando hay secuencias de caracteres que queremos borrar:

  tr -s " " <> ficherodestino

- -c: hace que se traduzcan todos los caracteres que no se encuentren especificados en el primer parámetro. En el siguiente ejemplo se traduce por una ? todo lo que no sean letras o números.

  tr -c '[a-z][A-Z][0-9]' ? <> ficherodestino

- -d: borra los caracteres que especifiquemos.

  tr -d '[a-z][0-9]' ? <> ficherodestino

- Otro ejemplo práctico para conseguir una lista, convirtiendo los caracteres de nueva línea en espacios:

  tr '\n' ' ' < /etc/pkgsync/musthave > listaenunasolalinea

4.4. uniq

uniq es uno de los filtros que nos sirve para filtrar o eliminar las líneas repetidas con los que trabajamos bastante (equivale a sort –u).

Los parámetros más importantes son:

- -u: para visualizar líneas no repetidas no tenemos que indicar ningún parámetro, aunque podemos pasarle este parámetro.
- -d: para visualizar las líneas repetidas.
- -c: para contar líneas repetidas.

4.5. find

Utilizamos este comando para buscar archivos dentro de una jerarquía de directorios. Pero, lo mejor de todo es que no sólo podemos buscar, sino que, además, podemos ejecutar acciones sobre los elementos localizados por el comando find. Por otro lado, podemos realizar la búsqueda mediante varios criterios.

La sintaxis de este comando es:

find [ruta...] [expresión]

Veamos un ejemplo sencillo: buscar los archivos de imágenes con extensión .jpg en el directorio del usuario pepe:

$ find /home/pepe -name "*.jpg"

Otro ejemplo: listar los directorios que hay en el directorio actual:

$ find ./ -maxdepth 1 -type d

Ahora se quiere listar los ficheros que se han modificado hoy en el directorio actual:

$ find ./ -mtime 0 -type f

Borrar todos los subdirectorios del directorio /var/backup que tengan una antigüedad mayor de 20 días:

$ find /var/backup -mtime +20 -type d -exec rm -f {} \;

Otro ejemplo: queremos borrar todos los directorios del sistema que contengan la palabra sane:

find / -name "*sane*" -type d -exec rm -rf {} \; 2>/dev/null

Si lo que queremos es borrar todos los ficheros del sistema que contengan la palabra sane, no tenemos más que cambiar el tipo en el comando anterior:

find / -name "*sane*" -type f -exec rm -fr {} \; 2>/dev/null

Otro ejemplo: Imaginemos que queremos recopilar todos los ficheros mp3 que tenemos repartidos en diferentes directorios y moverlos a un único directorio:

find / -name "*.mp3" -exec mv {} /compartido/musica/ \;

Imaginemos también que los usuarios de nuestro sistema descargan ficheros mp3 que almacenan en sus cuentas y terminan excediendo su cuota. Tan sólo tenemos que ejecutar el siguiente comando y borraremos todos los mp3 que haya en el home:

find /home -name "*.mp3" -exec rm {} \;

4.6. head

cmp [opciones]fichero1 fichero2

Por defecto, imprime las diez primeras líneas de cada fichero dado. Si no se le indica ningún fichero lee de la entrada estándar. Su sintaxis es la siguiente:

head [opciones] [fichero]

Algunas de las opciones más usuales son:

- -n número (sin –n haría lo mismo): imprime las primeras [numero] filas del fichero.
- -c número: imprime los primeros [numero] bytes del fichero.

Por ejemplo, el siguiente comando imprimiría las primeras diez líneas del fichero indicado:

head –n 10 archivo.txt

4.7. tail

Este comando sirve para visualizar la parte final de un documento (en inglés tail significa cola). Se puede usar en muchos casos, por ejemplo, supongamos que tenemos

un log de errores del servidor web Apache que ocupa varios GB. Mostrarlo usando el comando cat error_log no sería una buena idea, y menos si lo único que queremos es ver un error reciente que está en las últimas líneas del fichero. En este caso esta instrucción nos vendría muy bien para que nos mostrara por pantalla las últimas 50 líneas del archivo error_log.

La sintaxis es:
tail [opciones] [fichero]
Algunas de las opciones más usuales de este comando son:

- -n número (sin –n haría lo mismo): imprime las últimas [numero] filas del fichero.
- -c numero: imprime los últimos [numero] bytes del fichero.
- -b numero: imprime los últimos [numero] bloques de 512 bytes del fichero.
- -r: muestra las líneas en sentido inverso a como se encuentren en el fichero.

tail –n 10 fichero.txt

Por defecto, muestra las últimas 10 líneas del fichero.

4.8. cut

Permite trabajar con determinadas columnas o campos de un fichero indicadas mediante parámetros al comando. La sintaxis del comando cut es:

cut [opciones] [fichero]
Y las opciones más comunes:

- -b lista: seleccionamos sólo esos bytes por línea.
- -c lista: selecciona sólo esos caracteres por línea.
- -d: usa un carácter en vez del tabulador para delimitar un campo, debe entrecomillarse si es un carácter especial.
- -f lista: seleccionamos sólo esos campos/columnas. También muestra cualquier línea que no contenga un carácter separador.
- -s: no muestra las líneas que no contienen delimitadores.

Y el posible rango de valores a indicar en cada “lista” es:

- N: sólo ese byte, carácter o columna. O varios separados por comas.
- N-: desde N hasta el final de línea.
- N-M: desde N hasta M (incluido).
- -M: desde el primero hasta M (incluido).

Algunos ejemplos de uso de este comando serían:

```
cut -c 1-3 /etc/passwd
cut -c 1,3 /etc/passwd
ls -l /etc|cut -c 2-10
ls -l /etc|cut -f 1
ls -l /etc|cut -f 1 -d " "
cut -f 5,3 -d ":" /etc/passwd
```

4.9. wc

Se trata de un contador de líneas, palabras y caracteres. La sintaxis del comando wc es la siguiente:

wc [-lwc] [archivo(s)]
Opciones:

- -l: visualiza sólo el número de líneas.
- -w: visualiza sólo el número de palabras.

- -c: visualiza sólo el número de caracteres.

4.10. df (disk free).

El comando df nos informa sobre la cantidad de espacio en disco que utiliza el sistema de archivos. Más precisamente, nos detalla el espacio total, ocupado y libre de nuestro sistema. Si lo queremos más amigable, se recomienda el parámetro –h (de human), que saca el tamaño en el formato más próximo al humano.

```
df -h
```

4.11. du (disk usage)

El comando du, de manera similar al anterior, nos expresa el uso en disco que realiza un conjunto de archivos y de manera recursiva en los directorios existentes.

Las opciones importantes son:

-h que, al igual que con el comando df, nos presenta el espacio ocupado en unidades fácilmente interpretables

-s nos ofrece un resumen de la carpeta explorada, obviando, de esta manera, entrar en detalles de cada archivo y cada subcarpeta existente.

Combinado con comandos anteriores, podemos por ejemplo listar las 3 carpetas que más ocupan, en el directorio raiz.

```
du -s /* 2>/dev/null | sort -nr | head -n3
```

5. Herramientas para manipulación de datos.

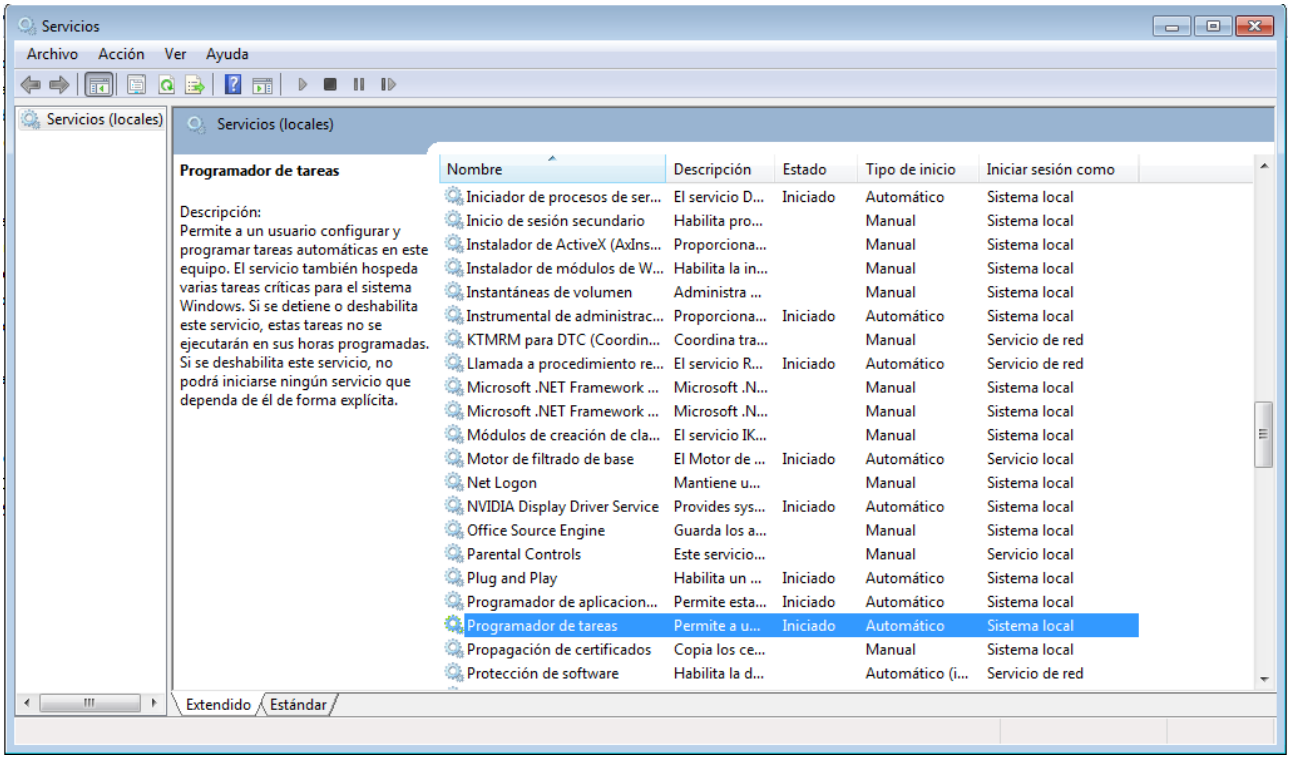
Existen varias herramientas potentes y relativamente difíciles de usar para la manipulación de datos. Un ejemplo de estas seria awk y sed.

6. Tareas programadas en Windows.

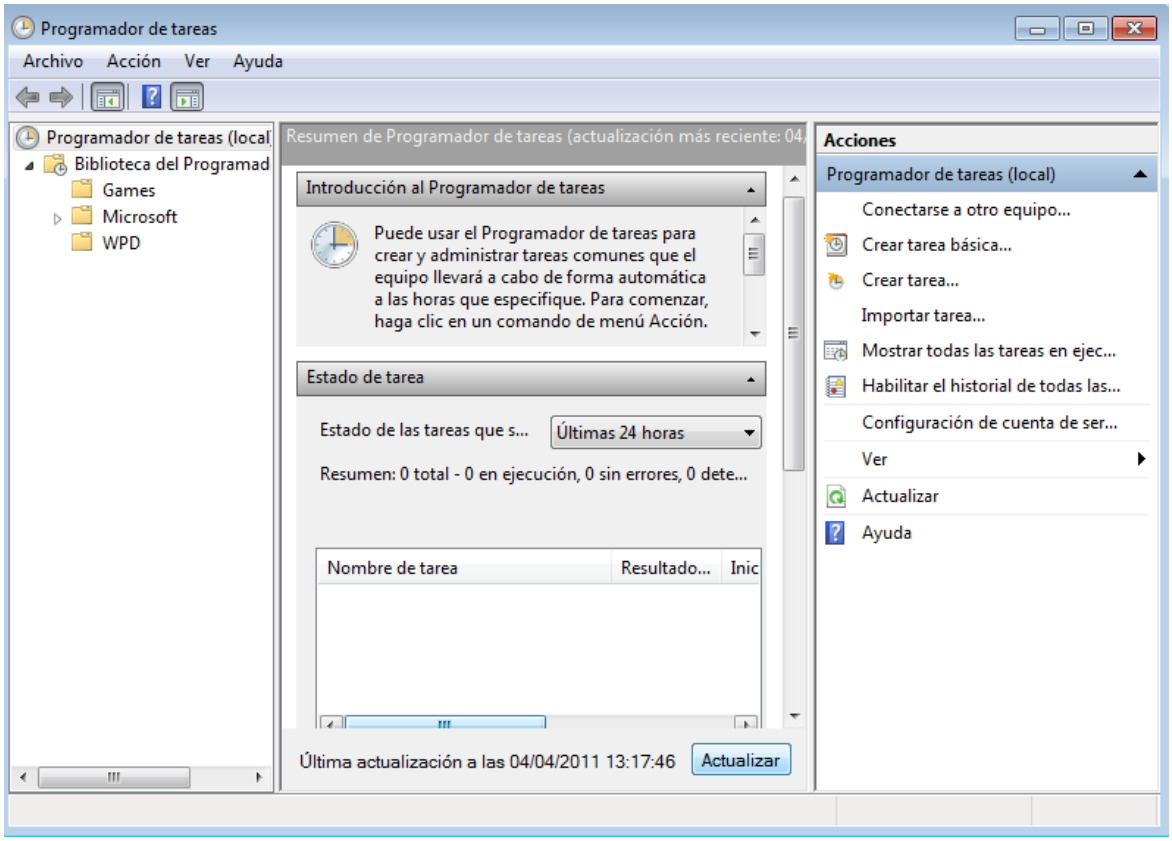
Windows cuenta con un programador de tareas que nos permite:

- Programar una tarea para que se ejecute diaria, semanal o mensualmente, o determinadas horas (por ejemplo, al iniciarse el sistema).
- Cambiar la programación de una tarea.
- Detener una tarea programada.
- Personalizar la forma en que se ejecutará una tarea a la hora programada.

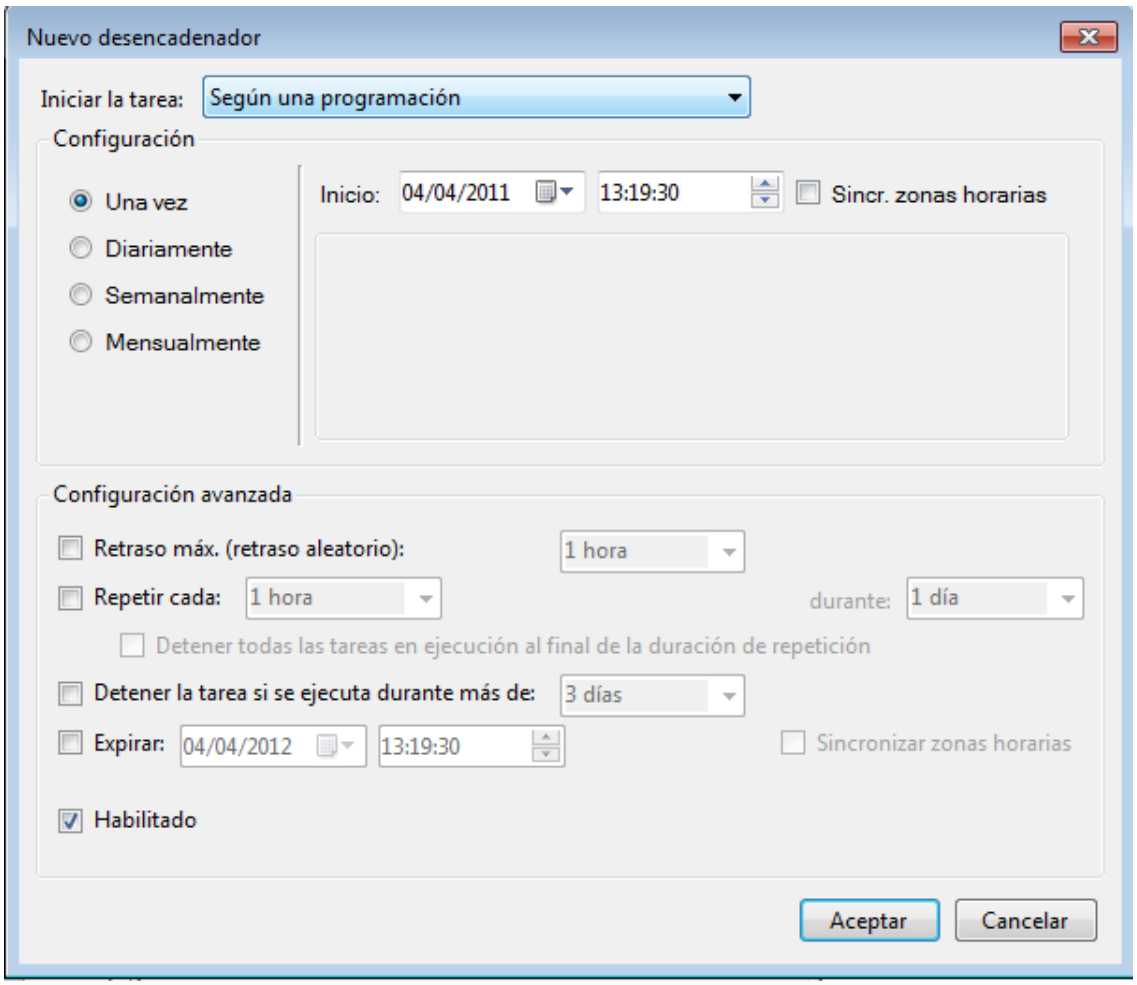
Para ello, se tiene que poner en marcha el servicio correspondiente, el servicio de programador de tareas:



Desde Windows 7, se accede por panel de control ⇒ Sistema y seguridad ⇒ Herramientas administrativas y, a continuación, haga doble clic en Programador de tareas.



Si pulsamos en crear tareas.



También se pueden programar tareas desde línea de comandos. La utilidad se llama schtasks (de scheduler – planificador- y tasks – tareas).

Los parámetros que incluyen son:

/Create	Crea una nueva tarea programada.
/Delete	Borra las tareas programadas.
/Query	Muestra todas las tareas programadas.
/Change	Cambia las propiedades de la tarea programada.
/Run	Ejecuta la tarea programada inmediatamente.
/End	Detiene la tarea programada que se está ejecutando actualmente
/?	Muestra esta ayuda/uso.

7. Tareas programas en Linux.

Existen en Linux, varios programas que realizarían la programación de tareas. El más común para gnome es gnome-schedule, aunque existen otros como kalarm o kcron. Para ello sería necesario instalar la herramienta.

En Linux existen comandos para realizar la programación de tareas. En particular existen 3 que suelen ser complementarias:

- cron: para planificar tareas periódicas, especificando el momento.
- anacron: es un programa libre que ejecuta asíncronamente tareas programadas en sistemas UNIX de manera periódica.
- at: es una herramienta que permite programar la ejecución de uno o varios programas en un momento futuro.

De todos el más importante y utilizado es cron. Anacron es útil en equipos que no funcionan todo el día, mientras que en servidores cuya ejecución no para, se utiliza

cron.cron es un administrador regular de procesos en segundo plano (*demonio*) que ejecuta procesos o guiones a intervalos regulares (por ejemplo, cada minuto, día, semana o mes). Los procesos que deben ejecutarse y la hora en la que deben hacerlo se especifican en el fichero crontab. El nombre *cron* viene del griego *chronos* que significa "tiempo".

Para que se ejecute las tareas es necesario que el servicio cron este corriendo. Anteriormente este servicio se llamaba crond.

Para utilizar el programador de tareas existen 3 opciones:

crontab –e : para añadir una nueva tarea.

crontab –l :para listar las tareas programadas.

crontab –r : para eliminar todas las tareas.

Un cron job del usuario consta de seis campos y luce como la siguiente línea:

1 2 3 4 5 /path/to/command arg1 arg2

Donde:

1: Minuto (0-59)

2: Horas (0-23)

3: Día (0-31)

4: Mes (0-12 [12 == diciembre])

5: Día de la semana (0-7 [7 or 0 == domingo])

```
minuto (0-59),
|  hora (0-23),
|  |  día del mes (1-31),
|  |  |  mes (1-12
|  |  |  |  día de la semana (0-6 donde 0=Domingo)
|  |  |  |  |   Script o comando.
15 02 *  *  * /path/to/command arg1 arg2
```

Ejemplo de tareas programadas seria:

Por ejemplo:

*30 10 * * 1 root /usr/bin/who >> /home/quien.tex*

Ejecuta la orden **who** todos los lunes a las **10:30** y guarda la salida en el fichero *quien.tex*

Para especificar dos o más valores en cada variable, estas deben estar separadas por comas, siguiendo con el ejemplo anterior:

*0,30 * * * 1 root /usr/bin/who >> /home/quien.tex*

Ejecuta la orden who todos los lunes cada media hora y guarda la salida en el fichero *quien.tex*

Si queremos que se ejecute cada 15 minutos sería:

*0,15,30,45 * * * * root /usr/bin/who >> /home/quien.tex*

o

**/15 * * * * root /usr/bin/who >> /home/quien.tex*

En este ejemplo veremos como pasarle más de un comando al cron y de paso como puede programarse una descarga:

*30 21 * * * root cd /media/sda7/dexter/distributions/isos;wget http://example.com/fichero_a_descargar.loquesea*

Este otro es para programar el apagado del PC. En este caso todos los sábados a las 21.30

*30 21 * * 6 root /sbin/shutdown -h now*

Hay varios valores predefinidos que se pueden utilizar para sustituir la expresión CRON.

Entrada	**Descripción**	**Equivale A**
@yearly	Se ejecuta una vez al año	0 0 1 1 *
@annually	(igual que @yearly)	0 0 1 1 *
@monthly	Se ejecuta una vez al mes	0 0 1 * *
@weekly	Se ejecuta una vez a la semana	0 0 * * 0
@daily	Se ejecuta una vez al día	0 0 * * *

@midnight	(igual que @daily)	0 0 * * *
@hourly	Se ejecuta una vez cada hora	0 * * * *

También está disponible @reboot, que permite a un trabajo ejecutarse una vez cada vez que el demonio cron se inicie, que eso típicamente coincidirá con el arranque del servidor.

Así se puede saber cada hora que usuarios están en el sistema:

@hourly root /usr/bin/who >> /home/quien.tex