Programación de tareas

Hasta ahora, la mayoría de tareas de administración que hemos visto las hemos realizado conectándonos al equipo (bien en local o remoto) e interactuando directamente con él a través de un intérprete de comandos, interfaz gráfica, etc. Sin embargo, hay otras tareas de administración que vamos a querer ejecutar justo cuando no estamos trabajando en los equipos, sirva como ejemplo diferentes tareas de mantenimiento, como pueden ser copias de seguridad, migraciones, baterías de tests, auditorías de seguridad, etc.

Estas tareas suelen ser pesadas y de duración relativamente larga, por lo que nos interesará realizarlas cuando la máquina esté lo más "ociosa" posible para interferir lo mínimo con los procesos de los usuarios y del sistema. Si analizamos el estado de carga de nuestros sistemas, veremos que estos periodos de baja utilización de recursos suelen ser, en la mayoría de casos, a altas horas de la madrugada, cuando pocos o ningún usuario esté haciendo uso de las mismas. Evidentemente, lo que no vamos a requerir es tener a un administrador del sistema esperando a que sea medianoche para simplemente lanzar el comando que ejecuta la copia de seguridad.

En estos casos, la solución perfecta son las **tareas programadas**. De este modo, dejaríamos todo preparado para que los comandos necesarios (preferiblemente organizados en un script que además compruebe que los datos sean correctos, que no hubo errores, etc.) se ejecutaran en el momento indicado, ahorrándonos tener que estar pendientes de que alguien tenga que lanzarlos de forma manual.

Instintivamente, al pensar en tareas programadas, quizá la primera idea que nos viene a la cabeza es la **programación de tareas en tiempo**, de forma que se ejecuten a la hora y en la fecha indicada. Pero además de la programación en tiempo, hay otros tipos, como por **eventos**, de forma que cuando suceda algún hecho determinado en el equipo, se ejecute la acción programada. Este segundo tipo de tareas es tan importante en la administración de sistemas como el primero.

Dado que las tareas se pueden ejecutar en cualquier momento, es lógico pensar que serán **realizadas por servicios** que deberán estar activos en todo momento monitorizando el tiempo o los eventos que "dispararen" (a esto se le llama ***trigger*** en inglés) o desencadenen las tareas. Si el servicio no estuviera activo en el momento exacto en el que se deba desarrollar la tarea, esta quedará sin ejecutar. En estos casos, cuando vuelva a ejecutarse el servicio y dependiendo de sus características y configuración, puede suceder que se ejecuten las tareas retrasadas o que simplemente se ignoren y eliminen.

Recordemos que los servicios no disponen de interfaz ni comunicación directa con el usuario, pero deben recibir los datos de la tarea (hora/fecha o evento, comandos y/o scripts a ejecutar, etc.). Por este motivo, los diferentes servicios de programación de tareas disponen de distintas estrategias para acceder a estos datos, como un programa cliente que interactúa con el usuario y prepara los datos de forma que el servicio los pueda consultar, la escritura de datos en ficheros por parte del usuario para ser consultados por el servicio, etc.

IMPORTANTE: En general, los servicios ejecutarán las tareas en una especie de "terminal virtual". Esto quiere decir que, normalmente, los datos de salida y errores que se suelen imprimir en pantalla (stdout y stderr), se perderán al finalizar la ejecución. Por otro lado, el directorio de trabajo en el que se ejecute la tarea, puede ser uno distinto al esperado por el usuario, así que si no se indicaron rutas o se usaron rutas relativas para localizar ficheros de entrada/salida, scripts, etc, es probable que no se encuentren estos ficheros y la tarea falle. Por ello, es muy recomendable que al programar tareas, se tenga en cuenta lo siguiente:

1. Usar siempre **rutas absolutas** para especificar los scripts, ficheros de entrada, de salida, etc.
2. Usar **redirecciones** para almacenar en ficheros los mensajes y datos de salida (stdout) y/o errores (stderr) si deseas conservarlos al finalizar la tarea. Si hubiera entradas por teclado, como evidentemente no va a ser posible usarlo, deberíamos redirigir también la entrada (stdin).
3. Al no ejecutarse las tareas en terminales "normales", no podemos confiar en que cargue la configuración que hayamos indicado en nuestra shell habitual, así que **NO debemos usar variables de entorno propias, alias** o cualquier otro tipo de configuración que se ejecuta al iniciar la shell.

GNU/Linux

Tareas por tiempo: puntuales

En GNU/Linux no suele venir instalada por defecto ninguna utilidad para tareas puntuales, si bien podemos instalar diferentes paquetes que ofrecen este servicio. Entre ellos quizá el más famoso sea `at` (se puede instalar con `apt install at`).

Normalmente al invocar el comando `at` se suele indicar la hora y/o la fecha en la que se va a ejecutar el o los comandos. El comando `at` permite diferentes formas de especificar la hora (formato 12h con AM o PM, o bien formato 24h, `noon` [12:00], `midnight` [0:00], `teatime` [16:00], etc. y, sobre todo las fechas, pudiendo indicar una fecha absoluta (se recomienda formato japonés: `[CC]YY-MM-DD`, aunque también soporta formato americano `MM/DD/[CC]YY`, o europeo `DD.MM.[CC]YY`), pero también fechas relativas usando un lenguaje "natural", aunque sólo funciona en inglés: `tomorrow`, `tuesday`, `next week`, `next month + 1 week`, `now + 10 minutes`, `now + 15min`, `now + 2 hours`, `next week - 2`

`days`, etc. Si especificamos una fecha incorrecta o en un formato no válido, obtendremos un error "`Garbled time`" (tiempo confuso).

Una vez invocado el comando `at`, se abrirá un pequeño intérprete de comandos donde iremos especificando qué comandos queremos ejecutar, pudiendo indicar varios de ellos. Para salir de este intérprete de comandos y programar las tareas, pulsamos *`Ctrl+D`*.

```
at 18:00 tomorrow
> who  >> ~/usuarios.txt 2>> ~/usuarios.err
> last >> ~/usuarios.txt 2>> ~/usuarios.err
Ctrl+D
```

También podemos especificar una sola tarea programada en una única línea directamente, sin el intérprete de comandos, pero debemos tener cuidado de que los comandos se usan adecuadamente y la tarea se crea de forma correcta. Una primera opción para progrmar la tarea en una línea es usando la redirección de entrada (<<<) y poniendo el comando entre comillas. Por ejemplo, para almacenar la carga del sistema dentro de 10 minutos:

```
at now + 10min <<< "uptime > ~/carga.txt 2> carga.err"
```

Otra opción para programar una tarea en una sola línea, es mostrar el comando con un `echo` y luego usar un pipe:

```
echo "uptime > ~/carga.txt 2> carga.err" | at now + 10min
```

Si en vez de comandos deseamos ejecutar un script, lo podemos hacer con la opción **`-f`**:

```
at next month + 1 day -f ~/scripts/copia_seguridad.sh >> ~/backup.txt 2>> ~/backup.err
```

Una vez programadas las tareas, será el demonio `atd` el encargado de comprobar cuándo deben ejecutarse y llevarlas a cabo en el momento adecuado. Si el demonio `atd` no está activo en ese momento programado (por ejemplo, el equipo está apagado), en general la tarea programada será ejecutada tan pronto como sea posible, normalmente al encender el equipo o activar de nuevo el demonio `atd`.

Algunas opciones útiles de `at` son:

- Mostrar todas las tareas programadas: **`atq`** (o también: `at -l`)
- Mostrar los detalles de la tarea con ID X: **`at -c`** X (al principio se muestran todas las variables que se almacenan, y al final el comando a ejecutar)
- Borrar una tarea con ID X: **`atrm`** X (o también: `at -r X` o bien `at -d X`)

Más información sobre `at`: **`man at`**, https://www.computerhope.com/unix/uat.htm

Tareas por tiempo: repetitivas

Para las tareas repetitivas, la opción más habitual es `cron` (el nombre hace referencia a Crono o Chronus, dios del calendario, las edades, etc.) que suele estar presente en prácticamente todas las distribuciones porque el propio sistema lo utiliza para ejecutar sus propias tareas de forma periódica. Normalmente viene instalada por defecto la versión de `cron` creada por Paul Vixie, pero hay otras versiones más modernas con más funcionalidades, que pueden estar instaladas como alternativa o complemento añadiendo nuevas funcionalidades (`anacron`, `cronie`, `hcron`, `fcron`, etc.).

El `cron` tradicional no dispone de un cliente para especificar el momento en el que se van a ejecutar las tareas, sino que debemos escribirlo en una tabla (`crontab`) con un formato especial, y luego esta tabla es leída por el demonio llamado `cron` (en algunos sistemas se usa `crond`), que es el que ejecuta las tareas. Para editar esta tabla, podemos usar el comando `crontab -e` (la primera vez que lo ejecutemos nos pedirá que indiquemos el editor que deseamos usar para editar la tabla. Si no tenemos conocimientos avanzados en editores, la mejor opción es elegir el editor más simple, que suele ser `nano`).

Una vez ejecutado `crontab -e`, para añadir tareas sólo tenemos que añadir una línea, especificando la tarea con el siguiente formato (se puede usar como separador cualquier número de espacios):

```
m  h  dom  mon  dow  command
```

`m`: minutos (de 0 a 59)
`h`: hora (de 0 a 23)
`dom`: día del mes (del 1 al 31, dependiendo del mes)
`mon`: mes (de 1 a 12)
`dow`: día de la semana, donde 0: domingo, 1: lunes, 2: martes, ..., 6: sábado
`command`: comando o script a ejecutar

Los valores se pueden especificar también usando algunos símbolos:

- Cualquier valor: *
- Rangos (`-`): 4-8 (del 4 al 8 ambos inclusive)
- Lista (`,`): 3,6,12,18

- Intervalos (/): */5 (cada 5 = 0,5,10,15,...)
- También podemos realizar algunas combinaciones: 6-12/2 = 6,8,10,12

Ejemplos:

```
# Usar un script para borrar ficheros temporales todos los días a las 12:15
#m  h  dom mon dow  cmd
15  12  *   *   *   ~/borrar_temporales.sh >> ~/borrar_temporales.out 2>> ~/borrar_temporales.err

# Guardar la carga cada 20 minutos de 8:00 a 19:40 de lunes a viernes
# durante los días impares de la primera quincena de los meses de febrero, mayo, junio y septiembre
# m    h      dom     mon    dow  cmd
*/20  8-19  1-15/2  2,5,6,9  1-5  uptime >> ~/carga.txt

# Ejecutar un script cada minuto
#m  h  dom mon dow  cmd
 *  *   *   *   *   ~/mi_script.sh >> ~/mi_script.out 2>> ~/mi_script.err
```

Algunas versiones más recientes de cron soportan formas no estándar, como indicar intervalos que empiecen por cualquier valor (`6/10`: 6,16,26, ...) o atajos como `@hourly`, `@daily`, `@weekly`, `@monthly`, `@yearly`, o incluso `@reboot`. Antes de usar este formato no estándar, hay que asegurarse que nuestro cron lo soporta, ya que para poder guardar los cambios e instalar las nuevas tareas, se comprobará que el formato sea correcto. Si hubiera cualquier error, no podríamos instalar las nuevas tareas.

Otra forma de indicar las tareas es escribiéndolas en un fichero y luego instalar el fichero en el cron (¡cuidado! se perderán todas las tareas previas). Por ejemplo, si hemos escrito todas nuestras tareas en el fichero `mis_tareas.txt`, para instalarlo sólo tenemos que hacer `crontab mis_tareas.txt`.

Las tareas se instalan a nivel de usuario, por lo que cada usuario tendrá su propia tabla de tareas. Normalmente, las tareas del sistema se almacenan en la tabla de tareas del usuario `root`. Hay que tener en cuenta que si el demonio no está activo en el momento de ejecutar la tarea (ha sido desactivado, el equipo está apagado o no responde, etc.), por defecto la tarea se perderá y no será ejecutada.

Otras operaciones útiles son:

- Listar todas las tareas: `crontab -l`
- Borrar todas las tareas: `crontab -r`
- Borrar algunas tareas: podemos editar la tabla de tareas con crontab -e y borrar las líneas de las tareas a eliminar, o bien comentarlas con el caracter #, lo que nos permitirá recuperarlas en otro momento.

Más información sobre cron: **`man cron`**, https://www.tutorialspoint.com/unix_commands/crontab.htm
Practicar el formato de crontab: https://crontab.guru/

Comando watch

Cron tiene una precisión de minutos, por lo que no nos permite repetir tareas cada pocos segundos. Para estos caso podemos usar el comando **`watch`**, si bien no es esa su principal función. El comando `watch` permite ejecutar un comando de forma periódica, mostrando la salida a pantalla completa.

Entre las diferentes opciones que ofrece este comando, quizá las más útiles sean: `-n` *X* (ejecuta el comando cada *X* segundos), **`-d`** (resalta las diferencias en la salida). Otras opciones pueden ser útiles en determinados casos, como **`-p`** (aumenta la precisión en el cálculo de los intervalos de tiempo), **`-b`** (emite un sonido si el comando da algún error, es decir, si el código de error no es cero).

Ejemplo: simular `top` ejecutando el comando `ps` cada segundo, mostrando las diferencias:
`watch -n 1 -d ps aux`

Tareas por eventos

Existen diversas formas de especificar tareas cuando sucede algún evento. Si bien hay utilidades que soportan algunos eventos, puede ser necesario realizar nuestros propios scripts para estos casos. Algunos ejemplos de tareas por eventos son:

- **Tareas cuando el sistema está "ocioso" (con baja carga, *idle*):** el comando `batch` (está dentro del paquete at) ejecuta tareas cuando la máquina tiene poca carga, por lo que es ideal para tareas de tipo "backfill" (de relleno). Por defecto las tareas se ejecutan cuando la carga baja por debajo de 1.5, aunque este valor es configurable en el momento de lanzar el demonio `atd`.
- **Tareas relacionadas con el sistema de archivos:** el paquete `incron` permite programar tareas relacionadas con eventos el sistema de ficheros (ejecutar una tarea cuando se crea un nuevo fichero, un archivo cambia, se borra, etc.)
- **Tareas durante el arranque de la máquina:** se pueden indicar con cron si soporta el atajo `@reboot`, o también con `update-rd.d`, si está disponible en nuestra distribución. Por ejemplo para realizar una tarea 5 minutos después de arrancar la máquina usando `crontab`, se puede especificar: `@reboot  sleep 300 && ~/mi_script.sh`
- **Tareas cuando se conecta algún usuari**o: escribiendo los comandos en `/etc/profile.d/` (en algunas distribuciones se usan en otros

directorios)
- **Tareas cada vez que se abre una terminal:** esto dependerá de qué shell use el usuario. Por ejemplo, en `bash` cada vez que se abre una terminal se ejecuta el fichero `~/.bashrc`, por lo que este fichero es el sitio ideal para definir la configuración, por ejemplo indicando alias de comandos (`alias ll='ls -lh --color'`), variables globales (`export LANG=es`), etc. Hay que tener mucho cuidado al editar estr ficheros, ya que el sistema los usa para su funcionamiento, por lo que si nos equivocamos al editarlos, es posible que el sistema se vea afectado. Normalmente es mejor no tocar estos ficheros de configuración, a no ser que seamos usuarios avanzados y sepamos lo que hacemos. Los cambios hechos en estos ficheros se aplicarán cuando se abra una nueva terminal. Para activar los cambios en la misma terminal, hay que usar el comando `source` o `.` (`source ~/.bashrc` o bien `. ~/.bashrc`).
- etc.

MS Windows

En MS Windows tenemos varias formas de programar tareas por comandos.

1) Para la consola `CMD`, podemos usar el comando **`schtasks`**, que permite indicar tareas programadas por tiempo (puntuales y repetitivas), por eventos, etc.

Por ejemplo: abrir la calculadora a las 20:38

`schtasks` `/create /sc once /st 20:38 /tn tarea1  /tr c:\Windows\System32\calc.exe`
donde:

- `/create`: crear la tarea
- `/sc once`: tarea puntual. Con `/sc` podemos indicar la periodicidad, puede ser puntual (`once`), diaria (`daily`), semanal (`weekly`), mensual (`monthly`), al inicio (`onstart`), cuando cualquier usuario abre sesión (`onlogon`), cuando el sistema tiene baja carga (`onidle`), etc.
- `/st 20:30`: hora de inicio
- `/tn tarea1`: nombre de la tarea
- `/tr c:\Windows\System32\calc.exe`: aplicación a ejecutar

Nota, si tenemos dudas de la ruta donde se encuentra alguna aplicación, podemos usar el comando `where`. Por ejemplo, `where calc.exe --> c:\Windows\System32\calc.exe`

Más información sobre `schtasks`: **`schtasks /?`**, https://docs.microsoft.com/es-es/windows-server/administration/windows-commands/schtasks

2) Para PowerShell, tenemos toda la familia de *cmdlet* **`-ScheduledTask`**: `New-ScheduledTask`, `Get-ScheduledTask`, etc.
Puedes consultar más información aquí: https://learn.microsoft.com/es-es/powershell/module/psscheduledjob/about/about_scheduled_jobs?view=powershell-5.1

Resumen

En la siguiente tabla se puede consultar un resumen de los diferentes comandos que podemos usar para programar tareas, tanto en GNU/Linux como en MS Windows (los comandos y rutas mostrados se corresponden con los que usaremos en las máquinas virtuales de clase, en otros sistemas y distribuciones, los comandos y rutas pueden ser diferentes):

Tipo	GNU/Linux	MS Windows (cmd)
Por tiempo (**tareas puntuales**)	**at** (-c, **atq**, **atrm**)	**schtasks** (/sc **ONCE**)
Por tiempo (**tareas repetitivas**)	**crontab** (-l, -e) anacron, ...	**schtasks** (/sc **HOURLY**, **DAILY**, **MONTHLY**, etc.)
Por evento (**baja carga**)	**batch**	**schtasks** (/sc **ONIDLE**)
Por evento (**arranque**)	crontab (@reboot) *update-rc.d*	**schtasks** (/sc **ONSTART**)
Por evento (**logon, cualquier usuario**)	/etc/profile.d/	**schtasks** (/sc **ONLOGON**) shell:common startup
Por evento (**login, usuario concreto**)	$HOME/.profile $HOME/.bashrc	shell:startup
Por evento (**cambio en directorios, ficheros, ...**)	incron	
Por evento (**otros eventos**)	*Programación propia*	schtasks (/sc ONEVENT)

NOTAS:

- En negro se muestran los comandos para programar las tareas. En **negrita** mostramos los comandos más usados y recomendados para un usuario con conocimientos básicos/intermedios. En los enlaces de este apartado encontrarás información y ejemplos de uso de estos comandos.

- En azul se muestran las rutas donde habría que colocar los scripts para que se ejecuten en los eventos adecuados, mientras que en verde aquellos scripts o runcoms (*rc) de GNU/Linux que habría que modificar añadiendo los comandos que queremos que se ejecuten en los eventos deseados.
  IMPORTANTE: estos directorios y ficheros suelen pertenecer al sistema y el propio sistema los utiliza para el correcto funcionamiento d equipo. En la mayoría de caso **se requieren conocimientos avanzados** para poder trabajar con estos directorios y ficheros, pudiendo .r a ocasionar serios problemas de configuración si no se realizan las acciones de forma adecuada. Por lo tanto, **SÓLO es recomendado para usuarios avanzados**.