Copias de seguridad

El objetivo es aprender las diferencias entre los distintos tipos de copias de seguridad (*full*, *diferenciales* e *incrementales*), aprender a programar tareas mediante *shell script* (en este caso un programa que genere copias de seguridad por si solo mediante el comando **tar** y nos notifique vía **email**) y por último crear un usuario restringido con el que generar y guardar estas copias de forma '*aislada*' en una unidad *USB* o disco duro externo.

Sobre las copias de seguridad.

Dependiendo del objetivo que tengamos a la hora de crear una copia de seguridad de nuestros datos, deberemos de seguir una u otra estrategia. Si lo que queremos es realizar solamente una copia completa o parcial como algo puntual de nuestros datos, podremos usar el comando **tar** en su uso más básico (opción 1) . En cambio si trabajamos con una gran cantidad de datos cambiantes y queremos seguir un control y respaldo de ellos de forma estricta por lo que pudiese ocurrir, podremos elegir la opción 2 de este tutorial.

1) Para la **primera** de las opciones bastaría con usar el comando *tar* o *cpio* en su uso normal para crear una full única:

- Respaldar el directorio *home* de mi usuario (empaquetado y compresión) con **tar**:

```
$ tar -cpvzf mibackup.tar.gz /home/miusuario
```

Nota: Podemos usar **-j** para comprimir con *bzip2* o **-J** para comprimir con *xz* (ambos suelen ofrecer mejor resultados que *gzip*, solo que este último es más común)

Importante: Si empaquetamos utilizando rutas absolutas deberemos de tener cuidado a la hora de desempaquetar, podremos machacar los archivos del *home*. Es preferible usar ruta relativa '*./**' o bien aislar el desempaquetado en un directorio.

- Respaldar mi directorio home con **cpio**.

```
$ find /home/miusuario | cpio -o | gzip /tmp/mibackup.cpio.gz
```

2) Para la **segunda** opción, seguramente una buena forma de respaldar los datos sería realizando copias diarias. Para no tener que realizar backups completos todos los días, algo que nos llevaría mucho tiempo y un consumo elevado de espacio en el disco*****, usaremos el comando *tar* con sus opciones **-g** o **-N** para crear backups *incrementales* o *diferenciales* respectivamente. Un ejemplo sería crear una copia full (*fullbackup*) de todo el directorio o dispositivos un determinado día de la semana (o del mes), por ejemplo un *Lunes* y el resto de la semana (*Martes*, *Miércoles*, *Jueves* y *Viernes*) realizar copias incrementales o diferenciales. Otra estrategia a seguir es la de combinar los 3 tipos de backups. Veremos esto más adelante.

***** Estos factores estarán determinados por el tamaño de los archivos a respaldar.

Backups diferenciales e incrementales

Los backups *diferenciales* copian todo el contenido cambiante basándose en una copia *full* anterior. Imaginemos que hemos realizado una copia *full* de un directorio un *Lunes,* y el *Martes*

realizamos una copia *diferencial* de ese mismo directorio, hasta aquí apenas existe diferencia entre el backup *diferencial* y el backup *incremental*. Pero el *Miércoles* volvemos a hacer otro backup *diferencial* del directorio, esta copia ya tendrá los cambios del *Martes* más los cambios del *Miércoles* por lo que su tamaño irá creciendo con respecto vayamos haciendo otras copias los próximos días (*Jueves* y *Viernes*).

En cambio, los backups *incrementales* siguen otro método. Lo que hacen es depender directamente de su copia anterior, es decir si tenemos una *full* realizada el *Lunes* y al día siguiente (*Martes*) se realiza un backup *incremental,* se guardarán los datos cambiantes entre el *Lunes* y el *Martes* hasta el momento de hacer la copia. Hasta aquí es similar al backup *diferencial,* pero la diferencia entra en juego al próximo día, es decir en la copia del *Miércoles*, donde la nueva copia *incremental* solo guardará las modificaciones realizadas desde el *Martes* hasta el *Miércoles* al momento de hacer el backup, por lo que el tamaño de este backup *incremental* será menor que el del *diferencial* del mismo día.

La ventaja de la copia *diferencial* radica a la hora de recuperar los datos, pero el ahorro de espacio en disco es menor. Si queremos volver a los datos del *Miércoles* pasado, deberemos de restaurar la copia *full* del *Lunes* y la *diferencial* del *Miércoles*. Por contra para recuperar los mismos datos mediante copias *incrementales*, deberemos de restaurar la *full* del *Lunes*, acto seguido la *incremental* del *Martes* y por último el backup *incremental* del *Miércoles*.

Combinar los tres tipos de backups: Comentábamos al inicio la posibilidad de combinar los tres tipos de copias. Esto es útil si en vez de crear una full todas las semanas, la creásemos únicamente el primer día del mes. En un caso hipotético de que el mes empezara en *Lunes* podemos entonces realizar un full backup el día 1 del mes (*Lunes*), el *Martes*, *Miércoles*, *Jueves* y *Viernes* creamos incrementales. El siguiente *Lunes* creamos una diferencial***** y volveremos a crear incrementales los siguientes días. Al siguiente *Lunes* volvemos a realizar un backup diferencial y repetimos está técnica hasta final de mes. Esta estrategia de backups nos permite restaurar el sistema a un determinado día desempaquetando la full del primer *Lunes* del mes y luego dependiendo de la semana, desempaquetaremos la diferencial del *Lunes* de esa semana y posteriormente las incrementales. Así no tenemos que desempaquetar la full del *Lunes* e ir restaurando una a una las copias incrementales hasta el día al que queremos volver (imaginar que este día es el día 21, deberíamos de restaurar la full mas 20 backups incrementales!!)

Una *ventaja* o *inconveniente* (según se mire) de combinar diferenciales e incrementales, es que si por ejemplo tenemos una full del día 1, la incremental del día 2 tiene nuevos archivos, la diferencial del día 3 contiene los archivos del día 2 y los del día 3, luego el día 4 eliminamos archivos y añadimos otros nuevos y hacemos una incremental, si queremos recuperar los eliminados, podremos volver a desempaquetar la diferencial del día 3. De no contar con ella no podríamos recuperarlos, ya que primero desempaquetaríamos la full del día 1, y luego las incrementales del día 2 y del día 4, en esta última no estarían los archivos que si había el día 3 (los cuales gracias a la backup diferencial SI podemos recuperarlos!!).

La desventaja es que a lo mejor estabas totalmente seguro de querer eliminarlos y cuando descomprimes la full y luego la diferencial, existen esos archivos que ya eliminaste.

Hacer backups incrementales con tar

- Creamos el *backup full* del directorio que queremos respaldar (hay que usar la opción **-g** como indicamos):

```
$ tar -cpvzf "fullbackup_`date +%d%m%Y`.tgz" -g /home/nebul4ck/history/backup.snap /home/nebul4ck/recursos/*
```

Nota: Podemos crear mejor el backup moviéndonos hasta *./recursos* e indicando en el comando *tar* la ruta relativa '***./****'. De la manera en la que lo hemos hecho deberemos de tener cuidado a la hora de desempaquetar. Si lo hacemos desde el directorio raíz machacaremos los archivos del *home*.

IMPORTANTE: El archivo *backup.snap* es el que guarda los metadatos que informan sobre los cambios que han ocurrido en el directorio. Si este archivo no existe se creará. Será el archivo que se lea cuando se haga el backup incremental. Si lo eliminamos, *tar* no encontrará información sobre los cambios realizados en el directorio, por lo que volverá a crear un full backup.

- Ahora podremos crear el backup incremental del *Martes*, los cambios que hayamos hecho serán grabados en el archivo *backup.snap (snapshot*).

```
$ tar -cpvzf "inc_backup_`date +%d%m%Y`.tgz" -g /home/nebul4ck/history/backup.snap /home/nebul4ck/recursos/*
```

Hacer backups diferenciales con tar

Para realizar backups diferenciales con *tar* usaremos su opción **-N**. Lo que nos permite esta opción es ordenar a *tar* que solo archive aquellos datos que han sido cambiados o agregados desde una determinada fecha, hasta la fecha de ejecución del comando. Por ejemplo imaginemos que la fecha del fullbackup es del *Lunes 20 de Febrero del 2017* y hoy (día en el que realizamos el backup diferencial es Miércoles 22 de Febrero del mismo año), haremos lo siguiente:

- Crear la copia full (la del día 20 de Febrero):

```
$ tar -cpvzf "fullbackup_`date +%d%m%Y`.tgz" /home/nebul4ck/recursos/*
```

- Crear copia diferencial con los cambios ocurridos desde el *Lunes 20 al Miércoles 22 del 2017*

```
$ tar -cpvzf "dif1_backup_`date +%d%m%Y`.tar.gz" /home/nebul4ck/recursos/* -N 20-feb-17
```

Nota: Si volviésemos a crear otro diferencial el Jueves o el Viernes con fecha 20-feb-17, contendrían también los cambios reflejados en dif1 es por ello que su tamaño va aumentando en comparación con los backups incrementales.

Programación de tareas con cron y atd

Cron es el demonio responsable de ejecutar tareas programadas y recurrentes (todos los días,

todas las semanas, etc.); atd está encargado de los programas a ejecutar una sola vez pero en un momento específico en el futuro.
En un sistema Unix, muchas tareas están programadas para ejecutarse regularmente:

- rotar los archivos de registro;
- actualizar la base de datos del programa locate;
- respaldos;
- scripts de mantenimiento (como limpiar los archivos temporales).

De forma predeterminada, todos los usuarios pueden programar tareas para ejecutar. Cada usuario tiene su propio «crontab» en el que pueden almacenarlas. Puede editarlo ejecutando crontab -e, el contenido del mismo es almacenado en el archivo /var/spool/cron/crontabs/usuario

---

SEGURIDAD

Restricción de cron o atd

Se puede restringir el acceso a cron si se crea un archivo de autorización explícita (una lista blanca) en /etc/cron.allow donde se indique sólo los usuarios autorizados a programar tareas. Todos los demás usuarios automáticamente quedarán excluidos de dicha funcionalidad. A la inversa, si sólo desea bloquear unos pocos usuarios problemáticos, se podría agregar sus nombres de usuario en el archivo de prohibición explícita /etc/cron.deny. Esta misma funcionalidad está disponible para atd con los archivos /etc/at.allow y /etc/at.deny.

El usuario root tiene su propio «crontab», pero también puede utilizar el archivo /etc/crontab o escribir archivos «crontab» adicionales en el directorio /etc/cron.d. Estas dos últimas soluciones tienen la ventaja de poder especificar el usuario bajo el que se ejecutará el programa.

De forma predeterminada, el paquete cron incluye algunas tareas programadas que ejecutan:

- programas en el directorio /etc/cron.hourly/ una vez por hora;
- programas en el directorio /etc/cron.daily/ una vez por día;
- programas en el directorio /etc/cron.weekly/ una vez por semana;
- programas en el directorio /etc/cron.monthly/ una vez por mes.

Muchos paquetes Debian dependen de este servicio: agregan sus scripts de mantenimiento en estos directorios, los cuales garantizan un funcionamiento óptimo de sus servicios.

Formato de un archivo crontab

Cada línea significativa de un archivo crontab describe una tarea programada con los siguientes seis (o siete) campos:

- el valor del minuto (número de 0 a 59);
- el valor de la hora (de 0 a 23);
- el valor del día del mes (de 1 a 31);
- el valor del mes (de 1 a 12);

- el valor de los días de la semana (de 0 a 7, donde 1 es el lunes y el domingo es tanto el 0 como el 7; también es posible utilizar las tres primeras letras del nombre del día en inglés, como Sun, Mon, etc.);
- el nombre de usuario bajo el que se ejecutará el programa (en el archivo /etc/crontab y en los fragmentos ubicados en /etc/cron.d/, pero no en los archivos de cada usuario);
- el programa a ejecutar (cuando se cumpla la condición definida por los primeros cinco campos).

Todos estos detalles están documentados en la página de manual crontab (info crontab).

Puede expresar cada valor como una lista de valores posibles (separados por coma). La sintaxis a-b describe el intervalo de todos los valores entre a y b. La sintaxis a-b/c describe el intervalo con un incremento de c (por ejemplo: 0-10/2 es lo mismo que 0,2,4,6,8,10. Un asterisco «*» es un comodín y representa todos los valores posibles.

```
#Formato
#min hora dia mes dds programa

# Descargar los datos todas las noches a las 19:25
25 19 * * * $HOME/bin/descargar.pl

# 08:00 en días de semana (Lunes a Viernes)
00 08 * * 1-5 $HOME/bin/haceralgo

# Reiniciar el proxy IRC luego de cada reinicio
@reboot /usr/bin/dircproxy
```

SUGERENCIA

Ejecución de un programa durante el inicio

Para ejecutar un programa sólo una vez, justo después de iniciar el equipo, puede utilizar el macro @reboot (reiniciar cron no disparará aquello programado con @reboot). Este macro reemplaza los primeros cinco campos de un elemento en el archivo «crontab».

Utilización del programa at

La orden at ejecuta un programa en un momento específico en el futuro. Obtiene la fecha y hora deseada como parámetros y el programa a ejecutar en su entrada estándar. Ejecutará el programa como si hubiese sido ingresado en la consola actual. at incluso se encarga de mantener el entorno para poder reproducir las mismas condiciones al ejecutar el programa.

Puede indicar la hora con las convenciones usuales: 16:12 o 4:12pm representan 12 minutos pasadas las 4 de la tarde. También puede especificar la fecha en varios formatos europeos u occidentales, incluyendo DD.MM.AA (27.07.17 representaría el 27 de Julio de 2017), AAAA-MM-DD (la misma fecha se representaría como 2017-07-27), MM/DD/[CC]AA (es decir: 12/25/17 o 12/25/2017 representan, ambas, el 25 de Diciembre de 2017) o simplemente MMDDCCAA (de forma que 122517 o 12252017 también representaría el 25 de Diciembre de 2017). Sin fecha, ejecutará el programa tan pronto como el reloj indique la hora especificada (el mismo día o el siguiente si ya pasó dicha hora ese día). También puede ingresar simplemente «today» o «tomorrow» representando el día actual o el día siguiente, respectivamente.

```
joaalsai@PCMint ~ $ at 09:00 28.02.17
warning: commands will be executed using /bin/sh
at> touch /home/joaalsai/hola.txt
at> <EOT>
job 8 at Tue Feb 28 09:00:00 2017
```