

## El futuro de las actualizaciones de Linux

## UPGRADE 2.0

A base de constantes actualizaciones de seguridad se puede llegar a alcanzar cierta paz mental, pero el hecho de tener que descargar-instalar-reiniciar continuamente no deja de ser un inconveniente. Mostramos cómo permanecer actualizados a la vez que evitamos paradas del sistema. **POR KURT SEIFRIED**

**H**ay pocas cosas más frustrantes que estar probando algo y que te cambien las versiones de un paquete de software inesperadamente, de modo que al final optamos por deshabilitar las actualizaciones automáticas en nuestra máquina de pruebas Fedora 11. Recientemente, sin embargo, al hacer un **yum update** vimos algo que nos sorprendió bastante: a pesar de que hacía poco que actualizamos la máquina por última vez, había 250MB para descargar e instalar (Figura 1). Dios bendiga a la banda ancha, porque si aún tuviésemos aquellas conexiones de acceso telefónico, o alguna otra conexión lenta, estaríamos con el agua al cuello.

**Más Pequeño es Mejor**

Qué importa si hay que descargar 100MB a la semana y reiniciar ocasionalmente la máquina para mantener el sistema actualizado. Si se pueden preparar actualizaciones lo suficientemente pequeñas como para que no afecten de manera significativa al ancho de banda empleado (algo que en sistemas inalámbricos supone una mayor duración de la batería), entonces es mucho más probable que podamos automatizar las actua-

lizaciones y mantener a todo el mundo a salvo. Es más, con los dispositivos integrados, que cada vez son más complejos (ejecutando Java, servidores web, servidores de correo, etc.), hay bastantes posibilidades de que sea necesario aplicar parches de seguridad.

Con archivos de paquete como RPM o DPKG, una posible solución pasa por excluir de la actualización todos los archivos que no hayan cambiado. Por ejemplo, en Fedora 11 el paquete *openoffice.org-core* ocupa 92MB, pero entre su publicación y la primera actualización hay alrededor de 30.3MB de archivos idénticos (imágenes, archivos XML, etc), además de la sobrecarga añadida por la propia elaboración del paquete (información sobre rutas, valores de comprobación de los archivos, etc), que no supone un aumento excesivo.

¿Cómo podemos reducir verdaderamente el tamaño de la actualización? Se pueden utilizar herramientas de comparación a nivel de bit para comparar dos archivos ejecutables y generar un archivo *diff*, relativamente pequeño, con el que actualizar el binario. Por ejemplo, el binario de *httpd*, tal cual se publicó con el paquete *Apache* en Fedora, es un

ejecutable de 322KB; sin embargo, el *bsdif* de la actualización más reciente no pasa de 8KB.

Por desgracia, *bsdif* no siempre nos vale. Un simple cambio en la llamada a alguna función al principio del archivo binario puede derivar en dos archivos binarios lo suficientemente distintos como para que el tamaño del parche sea demasiado grande (por ejemplo, al comparar dos archivos recientes de 1.8MB de tamaño del kernel de Fedora 11, se genera un parche *bsdif* de 1.8MB de tamaño).

**Una Forma Mejor**

El nuevo navegador web de Google, *Chrome*, incluye una función de actualización automática llamada *Courgette* [2], que sin ninguna duda pasará a formar parte de su sistema operativo *Google Chrome* (una plataforma Linux ligera con el navegador web Chrome como elemento protagonista). *Courgette* hace prácticamente lo mismo que *bsdif*, pero de una forma algo más inteligente. En vez de simplemente comparar en bruto dos ejecutables en formato binario, *Courgette* convierte primero los binarios a lenguaje ensamblador primitivo y hace la comparación en ese nivel. Elevando el asunto un nivel, *Courgette* es capaz de comparar las tablas de símbolos de los ejecutables con muchas más opciones de encontrar cadenas coincidentes y dando lugar a parches más pequeños gracias a la posibilidad de desechar todo aquello que en realidad no ha cambiado, incluso aunque su ubicación sí haya cambiado ligeramente. Google dice reducir de este modo en un noventa por ciento el tamaño de los parches en comparación con *bsdif*.

Aunque el problema del tamaño haya quedado resuelto, aún queda por resolver el engorroso problema de tener que lidiar con las actualizaciones del kernel que requieren reiniciar el sistema. A pesar de que un kernel puede llegar a

reiniciar bastante rápido, algunos servicios, como *VMware* o ciertas bases de datos, pueden llegar a tardar varios minutos en volver a estar operativas. En el caso de las bases de datos, pueden llegar a tardar incluso horas en volver a poblar la caché adecuadamente. Otros servicios, tales como los servidores de VoIP, puede que simplemente tengan que estar funcionando las veinticuatro horas del día, los siete días de la semana, sin posibilidad de parada. Por suerte, este problema ha llevado a la creación de *Ksplice* [3]. *Ksplice*, que en teoría es relativamente simple, crea un código de reemplazo (lo que obtenemos tras comparar el código actual con el de la actualización), resuelve los símbolos en el código de reemplazo y luego comprueba si el parche es seguro (por ejemplo, determinando si se están reemplazando funciones que hayan sido incluidas en algún otro sitio y que necesitan actualización allí también).

*Ksplice* introduce entonces instrucciones de salto (*JMP*) en el kernel original apuntando al nuevo código, y ya disponemos de un kernel actualizado sin haber tenido que reiniciar. Lo más interesante de *Ksplice* es que se puede aplicar a servicios que estén ejecutándose en espacio de usuario, pudiendo dar lugar a sistemas en los que no es necesario reiniciar ni siquiera los servicios para llevar a cabo una actualización. Aquellos que deseen probarlo pueden conseguir *Ksplice* del sitio web del archivo de paquetes de Ubuntu [4].

## El Papel de los Fabricantes

Por otro lado, ¿por qué los fabricantes no proveen un archivo completo y uno parcial que solamente contenga las actualizaciones? Parte del problema consiste en la sobrecarga: conservar paquetes parciales para cada actualización posible (por ejemplo, de la versión parcial 1.0 a la 1.1, de la 1 a la 1.1.1, de la 1.0 a la 1.2, etc.), o sólo para cada cada paso de la actualización (por ejemplo, de la versión 1.0 a la 1.1, de la 1.1 a la 1.1.1, de la 1.1.1 a la 1.2, etc). Probablemente los fabricantes adopten herramientas como *Courgette* y *Ksplice* conforme vayan madurando éstas.

## ¿Qué Hacemos?

Si se dispone de una sola máquina, no es que se pueda hacer mucho para reducir

el tamaño de las actualizaciones. En el caso de que fuesen dos o más máquinas (ejecutando el mismo software), hay varios trucos sencillos con los que reducir el número de descargas y el tiempo empleado en las actualizaciones. La mayoría del software existente recibe sus actualizaciones a través de HTTP, permitiendo el uso de un proxy web para gestionar las peticiones y hacer caché de

los datos. Claro está que habrá que cambiar la configuración predeterminada de forma significativa, permitiendo archivos de hasta 100MB (o más) y reservando un tamaño de caché de varios gigas. La ventaja de todo esto es que se puede instalar un servidor proxy transparente, como *Squid* [5], sin tener que modificar la configuración de los sistemas.

Otra estrategia efectiva consiste en montar el directorio de las actualizaciones desde un servidor central. Debemos tener cuidado con los sistemas basados en RPM y asegurarnos de que sólo compartimos el directorio con los paquetes (es decir, `/var/cache/yum/updates/packages/`). Compartir otros directorios por error, como por ejemplo el directorio `/var/cache/yum/updates/`, sería un desastre, puesto que los distintos sistemas compartirían archivos como `filelists.xml.gz.sqlite`. Estos archivos no están diseñados para que se pueda acceder a ellos desde varios sistemas a la vez de manera concurrente. En nuestro servidor principal tenemos NFS habilitado y el archivo `/etc/exports` contiene:

```
/var/cache/yum/base/packages *
*(rw,no_root_squash)
/var/cache/yum/updates/packages *
*(rw,no_root_squash)
```

Y, ¿cualquiera podría montar estos directorios y escribir en ellos como si fuese root?

RPM proporciona seguridad punto-a-punto en forma de paquetes firmados, de modo que si está habilitada la comprobación GPG (`gpgcheck = 1`) en

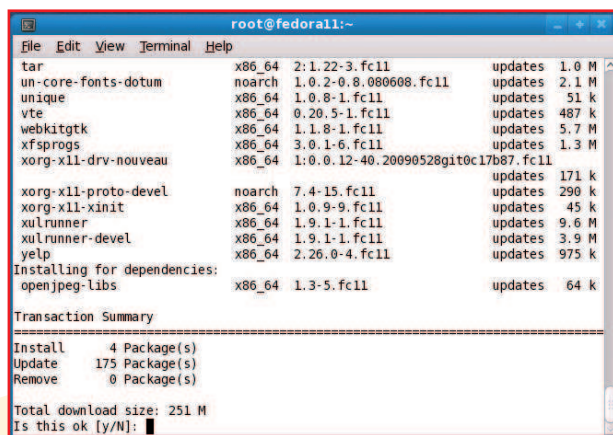


Figura 1: Ventana de confirmación mostrando el tamaño total de la descarga para las actualizaciones.

`yum.conf`, nos daremos cuenta rápidamente si alguien intenta modificar un paquete. La ventaja de permitir escribir en el repositorio central a todo el mundo es que si un cliente descarga un paquete para actualizarlo, ese paquete estará disponible de inmediato para el resto de las máquinas, incluido el servidor.

Por último, es posible crear medios de instalación personalizados con las actualizaciones ya incluidas en ellos, en vez de instalar primero el sistema operativo y después las actualizaciones. En instalaciones basadas en RPM/Anaconda podemos lograrlo mediante el uso del comando *createrepo* [6], creando nuevos archivos (normalmente contenidos en el directorio `repodata` del CD o DVD de instalación). Basta con copiar la imagen de instalación `.iso`, incluir en ella los nuevos paquetes, deshacernos de los viejos (puesto que probablemente necesitaremos el espacio que ocupan), ejecutar *createrepo* y quemarla en un CD o un DVD. De este modo disfrutaremos de un medio de instalación fresco con paquetes actualizados. ■

## RECURSOS

- [1] Utilidad diff/patch binaria: <http://www.daemonology.net/bsdifff>
- [2] Courgette: <http://dev.chromium.org/developers/design-documents/software-updates-courgette>
- [3] Ksplice: <http://www.ksplice.com>
- [4] Paquete Ksplice de Ubuntu: <http://packages.ubuntu.com/jaunty/ksplice>
- [5] Squid: <http://www.squid-cache.org>
- [6] createrepo: <http://createrepo.baseurl.org>