

Análisis de Ficheros de Registro con el Plugin check\_logfiles de Nagios

# DIARIO DEL VIAJERO



Axel Teichmann, Fotolia

El plugin check\_logfiles de Nagios ayuda en la monitorización de los ficheros de registro, incluso si rotan y cambian de nombre. **POR GERHARD LAUSSER**

La herramienta de monitorización Nagios es una plataforma genérica para la observación. Nagios permite que puedan observarse ordenadores, procesos, dispositivos y servicios de red, además de los ficheros de registro. La colección de plugins de Nagios viene con diversas opciones para monitorizar los registros. Los plugins *check\_log* y *check\_log2*, por ejemplo, son conocidos por muchos administradores; sin embargo, a veces presentan problemas en situaciones en las que una aplicación o script rota los ficheros de registro. Las herramientas tienden a equivocarse ocasionalmente y pierden algunas líneas, algo que no se puede permitir en el caso de necesitar una cobertura del 100%. Por otro lado, el plugin *check\_logfiles* [1] se desarrolló para comprobar cada línea individualmente, incluso si un registro se mueve, cambia su nombre o desaparece dentro de un archivo comprimido durante la fase de monitorización.

Pero eso no es todo: *check\_logfiles* destaca también sobre sus predecesores por diversas características complejas que posee. Por ejemplo, puede funcionar con múltiples claves de búsqueda, manejar excepciones que identifican un subconjunto especial de claves de búsqueda como inocuas, aplicar umbrales que dis-

paran alertas tras un mínimo número de coincidencias e integrar programas externos.

En este artículo se mostrará cómo comenzar a monitorizar ficheros de registro con el plugin *check\_logfiles* de Nagios. Para comenzar, se supone que se poseen conocimientos básicos de Nagios. Si se necesita más información de base sobre las herramientas de monitorización de Nagios, véase el taller sobre Nagios de Junio de 2007 de la revista Linux Magazine [2].

## Instalación

El plugin *check\_logfile* está disponible en [1]. Tras descomprimir el paquete hay que entrar en el directorio *check\_logfiles-2.3.1.1* y seguir los pasos estándar de compilación e instalación del plugin, *configure; make; make install*. Se encuentran disponibles diversas opciones para el paso de configuración (véase el cuadro titulado “Opciones de Configuración”).

## Primer Caso

Una vez instalado y configurado el plugin *check\_logfiles*, se puede poner en funcionamiento para monitorizar los ficheros de registro.

Como primera prueba para verlo en acción, considérese el siguiente ejem-

plo, que realiza una búsqueda simple de la cadena *BIGERROR* en un fichero cuyo nombre es *rhubarbomat.log*. La llamada al plugin tendrá el siguiente aspecto:

```
check_logfiles -criticalpattern='BIGERROR' -logfile=rhubarbomat.log
```

Si aparece la cadena *BIGERROR* en una línea que se haya añadido tras la última ejecución de *check\_logfile*, el plugin devolverá el estado *CRITICAL*; en caso contrario, devolverá *OK*. La cadena es realmente una expresión regular.

En vez de utilizar *-criticalpattern* se podría haber utilizado *-warningpattern*, como *-warningpattern='SMALLEROR'*. El código de salida tras una búsqueda con éxito es *1* para *WARNING*. Por supuesto, nada impide que se puedan utilizar ambas opciones al mismo tiempo.

Este primer ejemplo no tiene en cuenta la rotación de ficheros. Aunque el plugin identificaría la clave de búsqueda, no buscaría en los ficheros de registro que hayan sido rotados sino que, por el contrario, simplemente se centraría en el último fichero.

## Opciones de Configuración

--with-perl si se prefiere una instalación Perl separada.

--prefix especifica el directorio *home* de la instalación de Nagios. El plugin se instala en el subdirectorio */libexec*.

--with-seekfiles-dir especifica el directorio en el que se salvará la información de estado mientras el programa se ejecuta.

Para permitir que la búsqueda cubra los ficheros de registro rotados entre dos llamadas a *check\_logfiles* y evitar de esta manera el posible hueco, el plugin necesita una pista para saber dónde encontrar los antiguos ficheros.

El parámetro que se encarga de la rotación de los ficheros es *-rotation*, que o bien se le pasa el nombre del fichero nuevo o bien contiene una expresión regular que coincida con los nombres de los ficheros rotados (Figura 1). Primero asume que el fichero *rhubarbomat.log* se renombra automáticamente a *rhubarbomat.log.0* cada día y se crea un fichero

nuevo vacío *rhubarbomat.log*. Tras esto, *rhubarbomat.log.0* se renombra a *rhubarbomat.log.1*, y *rhubarbomat.log.1* a *rhubarbomat.log.2*, etc. En este caso, el parámetro *-logfile=/var/log/rhubarbomat.log* *-rotation='rhubarbomat.log\.[0-9]+'* permitirá al plugin encontrar y buscar en las versiones previas. Como alternativa, se podría especificar explícitamente el nombre del fichero *rhubarbomat.log.0*.

El plugin *check\_logfiles* sólo investiga las líneas del fichero de registro que hayan cambiado desde la última vez que fue invocado, lo que implica que cada vez que se ejecute de nuevo el plugin devolverá un resultado diferente. Tras devolver como resultado *CRITICAL*, la siguiente llamada devolverá *OK*, tal y como se muestra en el Listado 1. En el Listado 2 puede verse la definición de un servicio para Nagios.

## Fichero de Configuración

Donde realmente destaca *check\_logfiles* es cuando se utiliza un fichero de

configuración en lugar de los parámetros de la línea de comandos. Para el ejemplo anterior se tendría el siguiente fichero de configuración:

```
$ cat rhubarb.cfg
@searches = (
  tag => '0815',
  logfiles => [
    '/var/log/rhubarbomat.log',
    criticalpatterns => '.*0815.*',
    rotation => 'loglog0log1',
    options => 'noprotoocol'
  ]
);
```

Para llamar ahora al plugin habría que introducir *check\_logfiles -f fichero\_configuración* en la línea de comandos. Es fácil ver que el fichero de configuración está formado por código Perl. Los elementos en el array *@searches* (al que nos referiremos como “búsqueda”) son referencias hash que ligan los ficheros de registro con las claves de búsqueda. La etiqueta, *tag*, es un identificador único de la ligadura. El plugin lo

## Listado 1: Llamadas Recurrentes

```
01 $ logger "test1 this is 0815"
02 $ logger "this isn't because its 0916"
03 $
04 $ check_logfiles
   -logfile=/var/log/rhubarbomat.log -tag=0815
   -criticalpattern='.*0815.*'
   -rotation='loglog0log1'
05
06 CRITICAL - (1 errors in
   check_logfiles.protocol-2007-10-10-15-10-02) - Oct
   10 15:09:56
07 localhost lausser: test1 das ist doch 0815
   |0815_lines=2
08 0815_warnings=0 0815_criticals=1
09 0815_unknowns=0
10 $
11 $ echo $?
12 2
13 $
14 $ logger "rhubarb"
15 $ check_logfiles
   -logfile=/var/log/rhubarbomat.log -tag=0815
   -criticalpattern=
16 '.*0815.*' -rotation='loglog0log1'
17
18 OK - no errors or warnings |0815_lines=1
   0815_warnings=0 0815_criticals=0
19 0815_unknowns=0
20 $ echo $?
21 0
```

## Listado 2: Definición del Servicio

```
01 define service {
02   service_description check_0815msgs
03   host_name logserver
04   max_check_attempts 1
05   is_volatile 1
06   check_command
07   check_logfiles_critical!0815!/var/log/rhubarbomat.lo
   g!loglog0log1!.*0815.*
08 }
09 define command {
10   command_name check_logfiles_critical
11   command_line $USER1$/check_logfiles $$
12   -logfile="$ARG2$"
13   -criticalpattern="$ARG4$" -tag="$ARG1$" $$
14   -rotation="$ARG3$"
15 }
```

requiere para desambiguar los ficheros que almacenan la información de estado para cada ejecución de *check\_logfiles*. Un array hace posible la búsqueda en múltiples ficheros de registro con una única llamada a *check\_logfiles*.

El código Perl para esta configuración se muestra en el Listado 3.

Por otro lado, si se desea que el plugin active una alarma si la clave de búsqueda no se encuentra en el fichero de registro, el patrón de búsqueda debe comenzar con un signo de exclamación (!). La siguiente sintaxis indica si el último backup se finalizó sin errores:

```
criticalpatterns=>[
  ['!backup successful']
```

También se pueden definir excepciones que se parezcan al mensaje que se está buscando pero que representan un caso especial:

```
criticalpatterns=>[
  ['SCSI Error'],
  criticalexceptions=>[
  ['SCSI Error.*disk0.*'],
```

Estas entradas activarían la alarma de Nagios para la línea *SCSI Error /dev/disk5 I/O Timeout*, pero al mismo tiempo le indican al plugin que ignore *SCSI Error /dev/disk0 I/O Timeout*.

### Rotación

Al final de cada ejecución el plugin almacena la última posición que haya leído

del fichero de registro junto con la fecha de modificación y el número del inodo del fichero. Esta información se almacena en lo que se conoce como fichero de búsqueda. *check\_logfiles* genera el nombre del fichero de búsqueda a partir del nombre del fichero de registro y del día.

La próxima vez que se llame al plugin comparará estos datos con las propiedades del fichero de registro actual y comprobará si el fichero ha crecido, ha sido borrado, ha sido rotado o ha sido creado de nuevo.

En el caso de que se haya producido una rotación, el plugin busca el archivo rotado, que tendrá que leer para asegurarse de que no se ha quedado ninguna línea sin monitorizar.

Dependiendo de cuanto tiempo haya pasado desde que se llamó al fichero por última vez, puede que se hayan realizado varias rotaciones. El plugin utiliza una marca de tiempo y los parámetros de rotación para encontrar los ficheros.

En la mayoría de los casos, el fichero de registro habrá crecido sólo unas cuantas líneas. *check\_logfiles* continúa por la posición que etiquetó como final del fichero la última vez que se ejecutó y leerá las líneas siguientes hasta llegar al final del fichero (Figura 2). Este diseño

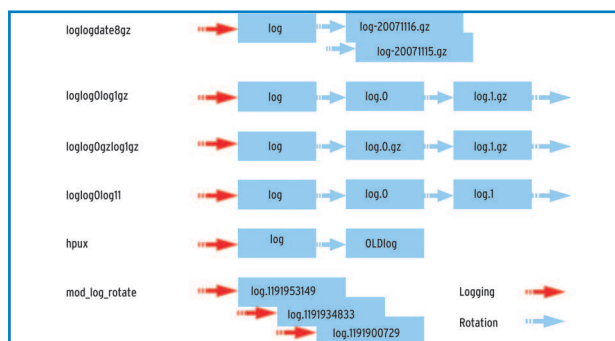


Figura 1: La rotación de los ficheros de registro complica el funcionamiento de los plugins. Las entradas pueden perderse debido a la copia y al cambio de nombre.

proporciona al plugin una ventaja de velocidad enorme con respecto a otras soluciones que se basan en *diff* para localizar las diferencias entre el fichero de registro actual y una copia almacenada del mismo, especialmente en el caso de un crecimiento rápido del fichero de registro.

Introduciendo *./configure with-seek-file-dir* se puede establecer un directorio para los ficheros de búsqueda o se puede modificar la ruta al vuelo con la variable *\$seekfilesdir* en el fichero de configuración.

El plugin utiliza */tmp* por defecto; sin embargo, es mejor cambiarlo por */var/tmp*, ya que algunos sistemas operativos no conservan el contenido del directorio */tmp* tras reiniciarse.

### Tipos

Además del parámetro de rotación, también se pueden realizar búsquedas con

#### Listado 3: Configuración de Ejemplo

```
01 @searches = (
02   {
03     tag => 'lamp-apache'
04     logfile => '/var/log/apache/error.log',
05     criticalpatterns => ['.*error.*', '.*fatal.*'],
06     rotation => 'solaris'
07   },
08   {
09     tag => 'lamp-mysql',
10     logfile => '/var/log/mysql.log',
11     criticalpatterns => ['corruption', 'you hit
12       a bug']
13   }
);
```

#### Listado 4: Búsqueda de Tipos de Ficheros de Registro Virtuales

```
01 @searches = (
02   {
03     tag => 'host0',
04     logfile =>
05       '/sys/class/scsi_host/host0/state',
06     criticalpatterns => [
07       'Link [^Up]+'
08     ],
09     options => 'noproduct',
10   },
11 );
```

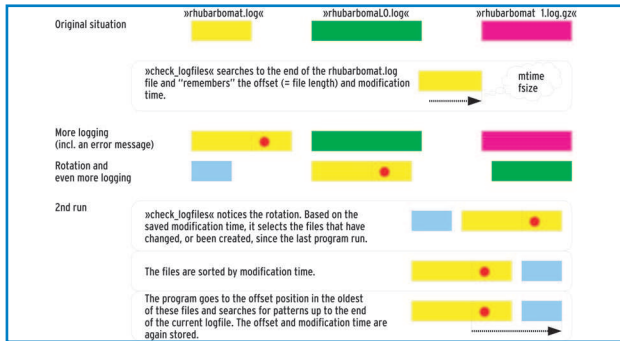


Figura 2: El Log Checker recuerda el fichero y la posición de lectura de cuando fue llamado y continúa exactamente en esa posición.

un parámetro de tipo, *type*, que especifica el tipo de fichero de registro. Si el parámetro de rotación existe, el parámetro de tipo toma el valor *rotation*. Esto significa que los ficheros archivados son relevantes para la búsqueda. Si el parámetro de rotación no está definido, *type* tomará el valor *simple*. Este parámetro puede ser útil si una aplicación genera continuamente ficheros de registro nuevos y borra los ficheros existentes, o si el administrador está preparado para aceptar el hecho de que las últimas líneas de un fichero de registro rotado no se van a tener en cuenta.

adido inmediatamente antes de su lectura. El plugin siempre investiga estos ficheros desde la primera línea (véase el Listado 4).

También el tipo *errpt* busca en los Informes de Error de AIX. Este tipo le indica al plugin que busque patrones en la salida del comando *errpt*, tal como si fuese un fichero de registro normal. El tipo *psloglist* también es experimental; permite que el plugin busque en el registro de eventos en una máquina Windows.

### Parámetros de Búsqueda

Hay disponible diversos parámetros para los patrones de búsqueda de los ficheros

Otro tipo de fichero de registro en el que se puede buscar es el *virtual*. Este tipo es utilizado por el sistema de ficheros */proc* de las máquinas Linux (y proporciona la opción de establecer fácilmente una monitorización del hardware).

Los ficheros en este sistema de ficheros no crecen; hay que tratarlos como si se hubiesen cre-

de registro. Los más importantes se listan en el cuadro titulado "Parámetros de Búsqueda". Una lista completa de todos los posibles puede encontrarse en línea en [1].

Los parámetros se utilizan en el fichero de configuración como se muestra en el fragmento que aparece en el Listado 5.

### Datos de Salida y Rendimiento

La salida de *check\_logfiles* contiene referencias a lo encontrado, por ejemplo:

```
CRITICAL - (3 errors in ↗
check_logfiles.protocol-2007↗
-10-10-16-21-09) InnoDB: ↗
Database page corruption on ↗
disk or a failed ...|↗
mysql_lines=12 ↗
mysql_warnings=0↗
mysql_criticals=3 ↗
mysql_unknowns=0
```

Además del código de salida típico de Nagios pueden observarse los tres puntos (...), lo que indica que existen más líneas con coincidencias.

### Listado 5: Parámetros en el Fichero de Configuración

```
01 $ cat rhubarb.cfg                               nix.*',
02 @searches = (                                  10 warningpatterns => ['.*failure.*',
03 {                                               '!successful'],
04 tag => '0815',                                  11 warningthreshold => 10,
05 logfile => '/var/log/rhubarbomat.log',          12 okpatterns => '.*cleared.*',
06 archivedir => '/var/log/archives',             13 options => 'case,noproto,script'
07 rotation => 'loglog0gzloglgz',                14 script => 'restart_rhubarbomat'
08 criticalpatterns => '.*0815.*',                15 });
09 criticalexceptions => '.*0815 macht aber
```

### Listado 6: Buscando Múltiples Patrones

```
01 @searches = (                                   thres',
02 {                                               10 'LINK ERROR',
03 tag => 'minor_errors',                          11 'SCSI BUS OR DEVICE ERROR',
04 type => 'errpt',                                 12 'SCSI DEVICE OR MEDIA ERROR',
05 criticalpatterns => ['ADAPTER ERROR',           13 'Possible malfunction on local adapter',
06 'The largest dump device is too small.',        14 'ETHERNET DOWN',
07 'The copy directory is too small.',            15 'UNABLE TO ALLOCATE SPACE IN KERNEL HEAP'
08 'Kernel heap use exceeds allocation            16 ],
count',                                           17 }
09 'Kernel heap use exceeds percentage            18 );
```

### Parámetros de Búsqueda

- *tag* Un descriptor único para la búsqueda actual.
- *logfile* El nombre del fichero de registro que se desea analizar.
- *archivedir* El directorio con los ficheros de registro rotados.
- *rotation* Una expresión regular que se utiliza para localizar los ficheros rotados. Existen valores predefinidos para la mayoría de los patrones más comunes.
- *criticalpatterns* Un patrón único que el plugin busca en el fichero de registro. Si se desea que el plugin busque múltiples patrones pertenecientes a una categoría, hay que especificarlos como elementos de un array (por ejemplo, veáse el Listado 6).
- *criticalexceptions* Soporta especificaciones de patrones de forma más granular: El parámetro ignora excepciones que no se contarán como errores.
- *warningthreshold* Los umbrales se utilizan cuando se quieren contar un cierto número de coincidencias antes de producir una alerta: *warningthreshold=>n* significa que la alerta se produce cada n coincidencias.
- *okpatterns* Reinicia el contador y borra todas las coincidencias críticas y de aviso anteriores.
- *nologfilenocry* Ignora los ficheros de registro no encontrados; de otra forma, si no se encuentra el fichero de registro, el plugin devolverá el estado UNKNOWN (desconocido).
- *syslogserver* Si el fichero de registro contiene mensajes de múltiples servidores, el plugin utiliza esta opción para buscar sólo los mensajes del ordenador local.
- *syslogclient=nombre\_pc* Como la opción servidor; sin embargo, en este caso, sólo los mensajes de un cliente determinado serán investigados. Esta opción es interesante para los servidores de Syslog centrales.
- *nocase* Ignora las mayúsculas/minúsculas en las expresiones regulares.
- *options* Múltiples opciones separadas por coma proporcionan un control más granular sobre las acciones del plugin. El prefijo *no* invierte el significado.
- *-nocase* Significa que los patrones no tienen en cuenta las mayúsculas/minúsculas.
- *-noprotocol* Impide que el plugin cree un fichero de protocolo. Normalmente, cualquier línea del fichero de registro que contenga el patrón de búsqueda se escribe en el fichero de protocolo. Esto ahorra tiempo de proceso en el caso de que se produzca una alerta.

Para cada búsqueda o día, el plugin también devuelve un conjunto de cuatro estadísticas de rendimiento:

- *\_lines* El número de líneas buscadas en el fichero de registro.
- *\_warnings* El número de líneas que contienen los patrones de aviso.
- *\_criticals* El número de líneas que contienen los patrones críticos.
- *\_unknowns* El número de líneas que

contienen los patrones desconocidos.

Estos parámetros proporcionan una referencia rápida del problema de densidad.

### Acciones

La opción *script* puede ejecutar un programa en caso de encontrar un patrón específico:

```
script=> 'nombre_del_programa'
```

o en la última versión:

```
script=> sub {código-Perl}
```

Las acciones pueden reiniciar una aplicación o enviar traps SNMP y mensajes NSCA. Esto significa que *check\_logfiles* puede ejecutarse como una aplicación

### Listado 7: Invocación de Scripts

```
01 $scriptpath = 17 tag => 'san',
   '/usr/bin/nagios/libexec:/usr/local/nagios/contri 18 logfile => '/var/adm/messages',
   b'; 19 criticalpatterns => [
02 $MACROS = { 20 'Link Down Event received',
03 CL_NSCA_HOST_ADDRESS => "lpmon1.muc", 21 'Loop OFFLINE',
04 CL_NSCA_PORT => 5778 22 'fctl:.*disappeared from fabric',
05 }; 23 '.*Lun.*disappeared.*'
06 24 ],
07 @searches =( 25 options => 'script',
08 { 26 script => 'send_nsca',
09 tag => 'rhubarb', 27 scriptparams => '-H $CL_NSCA_HOST_ADDRESS$
10 logfile => '/var/log/rhubarbomat.log', 28 -p $CL_NSCA_PORT$ -to $CL_NSCA_TO_SEC$
11 criticalpatterns => ['ERROR', 'crashed'], 29 -c $CL_NSCA_CONFIG_FILES',
12 script => 'restart_rhubarbomat', 30 scriptstdin =>
13 scriptparams => '-rhubarbprefix=bla', 31 '$CL_HOSTNAME$\t$CL_SERVICEDESC$\
14 options => 'script' 32 t$CL_SERVICESTATEID$\t$CL_SERVICEOUTPUT$\n',
15 }, 32 });
16 {
```

## Parámetros Globales

Además de los parámetros que sirven para una única entrada de búsqueda, todas las búsquedas leen una serie de parámetros globales adicionales que definen el comportamiento del plugin de forma independiente de las búsquedas individuales.

- *\$seekfilesdir* Especifica el directorio donde hay que guardar los ficheros con información del estado.
- *\$scriptpath* Una lista de rutas donde el plugin busca scripts externos, que posteriormente se ejecutarán con el parámetro *script*.
- *\$prescript* Especifica un programa externo que se ejecutará al comienzo.
- *\$postscript* Especifica un programa externo que se ejecutará al finalizar.
- *\$protocolsdir* Un directorio donde *check\_logfile* escribe los ficheros de protocolos con las líneas coincidentes.

independiente sin tener que depender del manejador de eventos de Nagios.

Los parámetros *scriptparams* y *scriptstdin* permiten a los usuarios ejecutar scripts con parámetros de la línea de comandos e incluso pasarles una entrada desde STDIN. El Listado 7 muestra un ejemplo. En él, cada vez que aparece un mensaje de error en una línea del fichero *messages*, la línea es enviada al servidor Nagios con el comando *send\_nscd* como resultado de un servicio pasivo.

En el caso más simple, el código de salida devuelto por el script externo será

irrelevante y no tendrá influencia en el código de salida de *check\_logfiles*. Por ejemplo, incluso si la aplicación Rhubarbomat del ejemplo del Listado 5 se reinicia con éxito, *check\_logfiles* aún devolverá el estado *Critical* a Nagios.

La opción *smartsript* pasa el código de salida del script externo al resultado de *check\_logfiles*. El plugin actúa como si hubiese descubierto otra línea tras la línea que ha disparado el script, que contiene el texto que fue la primera línea en el script de salida y el análisis del código de salida del script. Con esto se permite lanzar un error, pero no volver al mensaje original del fichero de registro o reevaluar el mensaje.

La tercera opción es *supersmartsript*. Los scripts de este tipo sobrescriben la coincidencia del script disparado en el fichero de registro con su código de salida y su salida, en vez de añadir una entrada. Hay disponible varias variables de entorno para estos scripts:

- *CHECK\_LOGFILES\_SERVICEOUTPUT* – el contenido de la línea que ha disparado el script
- *CHECK\_LOGFILES\_SERVICESTATE* – WARNING, CRITICAL o UNKNOWN
- *CHECK\_LOGFILES\_SERVICESTATEID* – 1, 2 ó 3

Con el uso de esta información y otros datos, como la fecha o los resultados de la aplicación reiniciada puede ser reevaluado entonces el mensaje de error. Con esto se permite al comprobador del fichero de registro degradar el estado de *CRITICAL* a *WARNING* o devolver un código de salida de 0, cancelándose con ello la alerta. El Listado 8 proporciona un ejemplo.

## Prescripts y Postscripts

Las acciones también pueden dispararse antes de comenzar a buscar en un fichero de registro o tras completar la búsqueda.

El parámetro *\$prescript*, que apunta a un script externo o a una subrutina Perl, sirve de ayuda a la hora de disparar las acciones. Los prescripts supersmart cancelan la ejecución de *check\_logfiles* si el código de salida es mayor que cero. Con esto se consigue ver si un proceso específico se está ejecutando.

Si el proceso no se está ejecutando, ¿por qué hay que molestarse en comprobar el fichero de registro de la aplicación en busca de errores? Los prescripts también pueden forzar a las aplicaciones a escribir (flush) sus ficheros de registro, asegurándose de que los datos están actualizados.

Los postscripts supersmart pueden reemplazar completamente los resultados de *check\_logfiles*, sin importar el número de mensajes de error que contenían originalmente.

O, si el formato estándar de la salida de *check\_logfiles* no se adecua a nuestro gusto, se podría ejecutar un postscript supersmart para modificarlo a nuestro antojo. ■

## RECURSOS

[1] *check\_logfiles*: <http://www.consol.com/opensource/nagios/check-logfiles/>

[2] "Network Monitoring: Watching your Systems with Nagios" por Julian Hein, Linux Magazine, Junio de 2007: <http://www.linux-magazine.com/issues/2007/79/nagios>

## Listado 8: Supersmart Script

```

01 @searches =(
02     {
03         tag => 'rhubarb',
04         logfile => '/var/log/rhubarbomat.log',
05         criticalpatterns => ['ERROR', 'crashed'],
06         script => sub {
07             if (`restart_rhubarbomat` =~ /successful/)
08                 if ($ENV{CHECK_LOGFILES_SERVICEOUTPUT}
09                     =~ /ERROR/) {
10                 printf "OK - restarted rhubarbomat\n";
11                 return 0;
12             } else {
13                 printf "WARNING - restarted crashed
14                 rhubarbomat\n";
15                 return 1;
16             } else {
17                 printf "CRITICAL - could not restart
18                 rhubarbomat\n";
19                 return 2;
20             }
21         },
22         options => 'supersmartsript'
23     },

```