

Detectar cambios en el sistema con Dnotify

RESCATE DE ARCHIVOS

Este mes veremos cómo evitar la tragedia de la pérdida de archivos con un sistema de control transparente basado en Perl.

POR MICHAEL SCHILLI

Durante las fases tempranas de un proyecto, los desarrolladores suelen experimentar con varias opciones, por lo que a veces resulta demasiado precipitado guardar prototipos en el sistema de control de versiones. Si aún no hemos configurado un repositorio, o si no estamos de acuerdo con su estructura, puede que acabemos trabajando sin ningún tipo de seguridad. En este caso, nuestro código puede caer víctima de un fervoroso `rm *` o del comando de borrado de nuestro editor.

El script Perl de este mes, *noworries*, nos ofrece un control automático de versiones. Cada vez que guardemos un archivo con nuestro editor, y cada vez que usemos el shell para manipular archivos usando comandos como `rm` o `mv`, un demonio escondido en segundo plano recibirá un mensaje. Al recibirlo, coge el archivo nuevo o modificado y usa RCS para darle una

versión. Todo esto se hace de manera transparente para el usuario. La Figura 1 muestra un usuario creando, y a continuación, borrando un archivo desde el intérprete de comandos. Sin la magia de Perl, el archivo, *myfile* habría pasado a mejor vida, pero ahora podemos teclear `noworries -l myfile` que nos indica que se creó una copia de seguridad justo 17 segundos antes. `noworries -r 1.1 myfile` rescata el archivo y escribe su contenido en la salida estándar.

El script no usa funciones manipuladas de shell ni otros sucios trucos. Por supuesto, una instancia del script tiene que ejecutarse en segundo plano (la opción `-w`, de “watch”, se encarga de esto) para iniciar la utilidad File Alteration Monitor (FAM), que en respuesta se suscribe a la interfaz del kernel del sistema operativo Dnotify. Cada vez que el sistema de archivos crea, mueve o borra un directorio o

archivo, o se modifica el contenido de un archivo, se notifica al kernel del evento. El File Alteration Monitor (FAM) le indica a Dnotify que está interesado en lo que está pasando en diversos directorios y recibe notificaciones como respuesta. CPAN tiene un módulo Perl (SGI::FAM) que traduce la interfaz C de FAM a Perl. Está basada en eventos y no requiere de sondeo intensivo por parte de la CPU. Al llamar al método `next_event()` se bloquea al demonio hasta que ocurra el siguiente evento.

La Figura 2 muestra otro ejemplo. En este caso se crea un archivo, y se modifica dos veces en la misma línea. El demonio recibe un mensaje para cada evento y crea tres versiones de los archivos en RCS (1.1, 1.2 y 1.3). Tecleando `noworries -l myfile` se mostrarán las tres versiones, aunque el archivo se haya borrado mientras tanto.

Si solicitamos la versión 1.2 con la opción `-r 1.2` y el nombre de archivo *file*, *noworries* nos la recupera e imprime su contenido por la salida estándar. El comando de shell mostrado en la Figura 2 redirecciona la salida hacia un archivo llamado *file*, que de nuevo se versiona con el demonio. La

```

mschilli@localhost:~/noworries
localhost.mschilli:noworries$ echo "First Line" >file
localhost.mschilli:noworries$ rm file
localhost.mschilli:noworries$
localhost.mschilli:noworries$ noworries -l file
1.1 8 seconds ago (first version)
localhost.mschilli:noworries$
localhost.mschilli:noworries$ noworries -r 1.1 file
RCS/file.v --> standard output
revision 1.1
First Line
localhost.mschilli:noworries$

```

Figura 1: El demonio en Perl trabaja "tras bastidores" para ofrecer un seguro de vida a los archivos por si son borrados por el usuario.

Figura 3 muestra la actividad del demonio: para estar más seguros, el demonio registra su actividad en el archivo `/tmp/noworries.log`.

El script `noworries` cuida de los archivos y directorios, sin importar lo escondido que estén, bajo `~/noworries` en el directorio de inicio del usuario. Allí es donde normalmente vamos a ubicar nuevos directorios o extraer tarballs si deseamos proteger nuestro sistema. El demonio crea una estructura bajo `~/noworries.rcs` para registrar los cambios tras las bambalinas. Cada subdirectorio contiene un directorio `RCS` con las versiones de los archivos, cuyos nombre terminarán en `v`. `RCS` fue una herramienta de UNIX desde sus comienzos y aún se usa para los sistemas de control de versión como `CVS` o `Perforce`. La siguiente secuencia de comandos registra una versión de `file`:

```

echo "Data!" >file
mkdir RCS
ci file
co -l file

```

El programa `ci` del conjunto de comandos de `RCS` crea `RCS/file,v` en el formato delta que usa `RCS`. El comando `co` del final, en combinación con la opción `-l` (de "look"), restablece la versión actual al directorio actual. Si modificamos ahora `file`, y a continuación ejecutamos otra secuencia de comandos `ci/co`, acabaremos con dos versiones en `RCS/file,v`, que pueden ser recuperadas por separado con `co`. El programa `rlog`, otro miembro de la familia `RCS`, nos permite visualizar los metadatos de las versiones de los archivos que hemos registrado.

El listado `noworries` (Listado 1) define los nombres de estas herramientas en las líneas 25 a 27. Si los

pasamos al script de esta manera, debemos asegurarnos que están ubicados en nuestro `PATH` para permitir a `noworries` que pueda llamarlos. En caso necesario, podemos codificar las rutas completas.

`noworries` usa las funciones `mkd` (crear directorio), `cp` (copiar archivo), `cd` (cambiar de directorio), `cdback` (vuelve al directorio original), y `tap` (ejecuta un programa y recupera el resultado) que exporta `Sysadm::Install`. Los lectores habituales de mi columna de Perl puede que recuerden esto de [4].

Noworries Te Vigila

Antes de que `SGL::FAM` pueda recibir mensajes acerca de archivos modificados en un directorio, `FAM` tiene que avisar al kernel de que quiere hacer esto. Los sucesos se desencadenan tras la llamada `$fam->monitor(...)` con `~/noworries` como argumento, cada vez que se crea un directorio o archivo nuevo directamente en `~/noworries`. Sin embargo, esto no se aplica en los subdirectorios. Por esta razón, `SGL::FAM` arranca inmediatamente otro monitor para los subdirectorios cada vez que detecta que se ha creado un nuevo subdirectorio. Un truco similar se aplica si `noworries` arranca cuando ya existe una estructura de directorios muy anidada bajo `~/noworries`.

(Volveremos a esto dentro de un momento).

Si especificamos la opción `-w` arrancamos `noworries` en modo demonio y se ejecuta el bucle infinito definido en la función `watcher` en la línea 88 del Listado 1. La llamada al método `next_event()`

en la línea 98 impide el flujo de ejecución hasta que ocurra uno de los cuatro eventos monitorizados por `FAM`. Para averiguar cuál de los potencialmente muchos directorios activos ha sido, el método `which()` de `SGL::FAM`, que es llamado en la línea 101, devuelve el directorio que ha causado el evento. El método del evento `filename()` devuelve el nombre del objeto nuevo, existente, modificado o borrado, que puede ser un directorio o un archivo.

El método `type()` nos devuelve el tipo de evento. Los tipos en los que `noworries` está interesado son `crate` y `change`. El método `monitor()` añade nuevos directorios a la lista de cosas a vigilar, mientras que la función `check_in()` definida en la línea 170 controla los archivos nuevos o modificados. Un método similar se usa para añadir directorios. El demonio utiliza `find` para localizar directorios cuando se inician, suponiendo que `~/noworries` ya existe. La función de apoyo `subdirs()` de la línea 153 excava hondo por el árbol de directorios y devuelve cualquier directorio que encuentre sin importar lo anidado que pueda estar.

La función `watch_subdirs()` itera sobre todos ellos y pasa las rutas relativas a `FAM` para su seguimiento. La sección de documentación de la

línea 266 no está sólo para que el usuario obtenga una página de manual adecuadamente formateada cada vez que la llame con `perldoc noworries`. También es la salida de la función `pod2usage()`, si el usuario no es capaz de proporcionar las opciones necesarias en línea de comandos. No tiene mucho sentido versionar temporalmente los archivos de `vi` o `emacs`, por lo que se filtran en las líneas 107 a 112.

Cuando un archivo tiene que registrarse en el sistema de control de versiones, `check_in` en la



línea 170 verifica primero que es un archivo de texto. *check_in* descarta los archivos binarios en la línea 174. Se llama a la función con una ruta relativa a *~/noworries*, ya que es donde salta *watcher()* en la línea 90. La línea 189 copia la fila original al árbol RCS, y la línea 195 llama al programa *ci* con

las opciones *-t* y *-m*. La pasa el valor *-a* a ambos, ya que el primero y subsiguientes comentarios de registro no tienen sentido. Pero tenemos que darle a *ci* algo para procesar, para evitar un prompt interactivo. La línea 204 comprueba la salida del archivo, como se describió anteriormente. La siguiente

vez que ocurra un cambio, la copia verificada se sobrescribe, y se registra la nueva versión con *ci*.

¿Qué Día es Hoy?

noworries llama a la función de RCS *rlog* para averiguar qué versiones de

Listado 1: noworries

```

001 #!/usr/bin/perl -w
002 #####
003 # noworries -
004 # m@perlmeister.com
005 #####
006 use strict;
007 use Sysadm::Install qw(:all);
008 use File::Find;
009 use SGI::FAM;
010 use Log::Log4perl qw(:easy);
011 use File::Basename;
012 use Getopt::Std;
013 use File::Spec::Functions
014 qw(rel2abs abs2rel);
015 use DateTime;
016 use
017 DateTime::Format::Strptime;
018 use Pod::Usage;
019
020 my $RCS_DIR =
021 "$ENV{HOME}/.noworries.rcs";
022 my $SAFE_DIR =
023 "$ENV{HOME}/noworries";
024
025 my $CI = "ci";
026 my $CO = "co";
027 my $RLOG = "rlog";
028
029 getopts("dr:w|",
030 \my %opts );
031
032 mkd $RCS_DIR
033 unless -d $RCS_DIR;
034
035 Log::Log4perl->easy_init({
036 category => 'main',
037 level => $opts{d}
038 ? $DEBUG
039 : $INFO,
040 file => $opts{w} &&
041 !$opts{d}
042 ? "/tmp/noworries.log"
043 : "stdout",
044 layout => "%d %p %m%n"
045 }
046 );
047
048 if ( $opts{w} ) {
049 INFO "$0 starting up";
050 watcher();
051
052 } elsif(
053 $opts{r} or $opts{l} ) {
054
055 my ($file) = @ARGV;
056 pod2usage("No file given")
057 unless defined $file;
058
059 my $filename =
060 basename $file;
061
062 my $absfile =
063 rel2abs($file);
064 my $relfile =
065 abs2rel( $absfile,
066 $SAFE_DIR );
067
068 my $reldir =
069 dirname($relfile);
070 cd "$RCS_DIR/$reldir";
071
072 if ( $opts{l} ) {
073 rlog($filename);
074 } else {
075 sysrun(
076 $CO, "-r$opts{r}",
077 "-p", $filename
078 );
079 }
080 cdback;
081
082 } else {
083 pod2usage(
084 "No valid option given");
085 }
086
087 #####
088 sub watcher {
089 #####
090 cd $SAFE_DIR;
091
092 my $fam = SGI::FAM->new();
093 watch_subdirs( ".", $fam );
094
095 while (1) {
096 # Block until next event
097 my $event =
098 $fam->next_event();
099
100 my $dir =
101 $fam->which($event);
102 my $fullpath =
103 $dir . "/" .
104 $event->filename();
105
106 # Emacs temp files
107 next
108 if $fullpath =~ /~/;
109
110 # Vi temp files
111 next if $fullpath =~
112 /\sw[px]x?$/;
113
114 DEBUG "Event: ",
115 $event->type, "(.",
116 $event->filename, ")";
117
118 if ( $event->type eq
119 "create"
120 and -d $fullpath ) {
121 DEBUG "Adding monitor",
122 " for directory ",
123 $fullpath, "\n";
124 $fam->monitor(
125 $fullpath);
126 }
127 elsif ( $event->type =~
128 /create|change/
129 and -f $fullpath ) {
130 check_in($fullpath);
131 }
132 }
133 }
134
135 #####
136 sub watch_subdirs {
137 #####
138 my ($start_dir, $fam) = @_;
139
140 $fam->monitor($start_dir);
141

```

un archivo están disponibles. *rlog* devuelve los números de versión con la fecha (en formato *yyyy/mm/dd hh:mm:ss*) y también revela el número de líneas que han cambiado en comparación con la versión anterior. Obviamente, no puede dar esta información de la versión inicial, pero sí se

nos indica que la versión 1.2 tiene *lines: +10 -0*, esto significa que hay 10 nuevas líneas en comparación a la 1.1, y que no se ha borrado ninguna.

El módulo *DateTime* de CPAN nos ayuda tremendamente con los cálculos de fechas. El módulo *DateTime::Format::Strptime* analiza la

información de fechas de RCS, y convierte el valor a segundos transcurridos desde el 1 de enero de 1970. Para hacer esto, el constructor espera recibir un cadena con el siguiente formato: "%Y/%m/%d %H:%M:%S", y la llamada a *parse_datetime()* devuelve un objeto *DateTime* comple-

Listado 1: noworries

```

142 for my $dir (
143 subdirs($start_dir) ) {
144 DEBUG "Adding monitor ",
145 "for $dir";
146 $fam->monitor($dir);
147 }
148
149 return $fam;
150 }
151
152 #####
153 sub subdirs {
154 #####
155 my ($dir) = @_ ;
156
157 my @dirs = ();
158
159 find sub {
160 return unless -d;
161 return if /\.\.?$/;
162 push @dirs,
163 $File::Find::name;
164 }, $dir;
165
166 return @dirs;
167 }
168
169 #####
170 sub check_in {
171 #####
172 my ($file) = @_ ;
173
174 if ( !-T $file ) {
175 DEBUG "Skipping non-",
176 "text file $file";
177 return;
178 }
179
180 my $rel_dir =
181 dirname($file);
182 my $rcs_dir =
183 "$RCS_DIR/$rel_dir/RCS";
184
185 mkd $rcs_dir
186 unless -d $rcs_dir;
187
188 cd "$RCS_DIR/$rel_dir";
189 cp "$SAFE_DIR/$file", ".";
190 my $filename =
191 basename($file);
192
193 INFO "Checking $filename",
194 " into RCS";
195 my ($stdout, $stderr,
196 $exit_code) = tap(
197 $CI, "-t-",
198 "-m-", $filename
199 );
200 INFO "Check-in result: ",
201 "rc=$exit_code ",
202 "$stdout $stderr";
203
204 ($stdout, $stderr,
205 $exit_code) = tap(
206 $CO, "-l", $filename);
207 cdback;
208 }
209
210 #####
211 sub time_diff {
212 #####
213 my ($dt) = @_ ;
214
215 my $dur =
216 DateTime->now() - $dt;
217
218 for (
219 qw(weeks days hours
220 minutes seconds)) {
221 my $u =
222 $dur->in_units($u);
223 return "$u $u" if $u;
224 }
225 }
226
227 #####
228 sub rlog {
229 #####
230 my ($file) = @_ ;
231
232 my ( $stdout, $stderr,
233 $exit_code )
234 = tap( $RLOG, $file );
235
236 my $p =
237 DateTime::Format::Strptime
238 ->new( pattern =>
239 '%Y/%m/%d %H:%M:%S' );
240
241 while ( $stdout =~
242 /^revision\s(\S+).*?
243 date:\s(.*)?;
244 (.*)$/gmxs) {
245
246 my ($rev, $date, $rest)
247 = ($1, $2, $3);
248
249 my ($lines) = ($rest =~
250 /lines:\s+(.*)/);
251 $lines ||=
252 "first version";
253
254 my $dt =
255 $p->parse_datetime(
256 $date);
257
258 print "$rev ",
259 time_diff($dt),
260 " ago ($lines)\n";
261 }
262 }
263
264 __END__
265
266 =head1 NAME
267
268 noworries - Dev Safety Net
269
270 =head1 SYNOPSIS
271
272 # Print previous version
273 noworries -r revision file
274
275 # List all revisions
276 noworries -l file
277
278 # Start the watcher
279 noworries -w

```

```

mschilli@localhost:~/noworries
localhost.mschilli:noworries$ echo "First Line" >file
localhost.mschilli:noworries$ echo "Second Line" >>file
localhost.mschilli:noworries$ echo "Third Line" >>file
localhost.mschilli:noworries$
localhost.mschilli:noworries$ rm file
localhost.mschilli:noworries$
localhost.mschilli:noworries$ noworries -l file
1.3 11 seconds ago (+1 -0)
1.2 19 seconds ago (+1 -0)
1.1 27 seconds ago (first version)
localhost.mschilli:noworries$
localhost.mschilli:noworries$ noworries -r 1.2 file >file
RCS/file,v --> standard output
revision 1.2
localhost.mschilli:noworries$ cat file
First Line
Second Line
localhost.mschilli:noworries$

```

Figura 2: Se añaden dos líneas a un archivo de nueva creación en dos sesiones consecutivas. Noworries recupera la versión 2 cuando se le requiere.

tamente inicializado si no hubo problemas. El bucle *while*, que comienza en la línea 241, navega por el complicado resultado con la ayuda de *rlog*, y para ello hace



uso de una expresión regular de varias líneas.

La función *time_diff()* de la línea 211 aguarda un objeto *DateTime* y calcula su edad en segundos, minutos, horas, días o semanas. Así será más sencillo de leer para el usuario de *noworries*.

Desafortunadamente, *Dnotify*, el mecanismo que usa FAM, no escala bien y sucumbe al llegar a unos doscientos subdirectorios. Para solventar este problema *dnotify* se ha remplazado por *inotify* en los kernels más recientes. *inotify* hace mejor uso de los recursos y escala con mayor

```

mschilli@localhost:~
2005/11/20 15:13:41 INFO Checking file into RCS
2005/11/20 15:13:41 INFO Check-in result: rc=0 RCS/file,v <-- file
file is unchanged: reverting to previous revision 1.1
done
2005/11/20 15:13:49 INFO Checking file into RCS
2005/11/20 15:13:49 INFO Check-in result: rc=0 RCS/file,v <-- file
new revision: 1.2; previous revision: 1.1
done
2005/11/20 15:13:57 INFO Checking file into RCS
2005/11/20 15:13:57 INFO Check-in result: rc=0 RCS/file,v <-- file
new revision: 1.3; previous revision: 1.2
done
2005/11/20 15:14:28 INFO Checking file into RCS
2005/11/20 15:14:28 INFO Check-in result: rc=0 RCS/file,v <-- file
new revision: 1.4; previous revision: 1.3
done
2005/11/20 15:14:29 INFO Checking file into RCS
2005/11/20 15:14:29 INFO Check-in result: rc=0 RCS/file,v <-- file
new revision: 1.5; previous revision: 1.4
done

```

Figura 3: Tras las bambalinas, el demonio monitoriza el sistema de archivos y crea una copia de seguridad con las versiones cada vez que ocurre un cambio en los directorios vigilados.

facilidad. FAM está también algo obsoleta, y se ha designado a Gamin [3] como su sucesor.

El mecanismo *Dnotify* del kernel no usa inodos del sistema de archivos, sino nombres de fichero, por lo que *mv file1 file2* activa dos eventos: un evento de tipo borrar y otro de tipo crear. Esto no le preocupa a *noworries*, ya que

el script ignora los eventos tipo borrar, y si aparece el mismo archivo tiempo después, simplemente se registra como la última versión.

El script sólo debería usarse en nuestro disco duro local, y no con NFS, ya que FAM sólo podría ser eficiente si el NFS

objetivo también está ejecutando FAM. En caso contrario, sondea el destino a intervalos regulares y esto provoca que el sistema completo sea ineficiente.

Instalación

Debemos instalar los módulos *Sgi::FAM*, *Sysadm::Install*, *DateTime*, *DateTime::Format::Strptime*, y *Pod::Usage* de CPAN. Un CPAN Shell nos ayudará a resolver las dependencias con rapidez. Si observamos un error tal que *FAM.c:813: error: storage size of 'RETVAl' isn't known* al compilar *Sgi::FAM*, debemos cambiar la línea 813 de *FAM.c* de *enum FAMCodes RETVAL* a *FAMCodes RETVAL*. Ejecutamos de nuevo *make* y debería funcionar.

Para asegurarnos que el demonio está siempre ejecutándose, añadimos una línea similar a *x777:3:respawn:su mschilli -c "/home/mschilli/bin/noworries -w" a /etc/inittab*, y a continuación se lo hacemos saber al demonio *init* ejecutando *init q*. El proceso tiene que ejecutarse con la ID del usuario en uso, (*mschilli* en este caso), para asegurar que *\$ENV{HOME}* en el script apunta al directorio de usuario correcto. También en este caso, el proceso *init* arranca el demonio *noworries* cuando encendemos el ordenador, y la opción *respawn* asegura que el proceso se reinicia inmediatamente si por alguna razón termina inadvertidamente. Pero antes de hacer todo esto, debemos probar el demonio desde línea de comandos para ver si todo funciona adecuadamente.

La opción *-d* de depuración puede servir de ayuda si encontramos problemas. Muestra el estado detallado en la salida estándar en lugar de volcarlo a */tmp/noworries.log*.

RECURSOS

- [1] Listados de este artículo: <http://www.linux-magazine.es/Magazine/Downloads/16>
- [2] Página de FAM: <http://oss.sgi.com/projects/fam/>
- [3] Página de Gamin: <http://www.gnome.org/~veillard/gamin/>
- [4] "Comandos Shell desde Perl", Michael Schilli: <http://www.linux-magazine.es/issue/05/PerlShell.pdf>