Detecting system changes with Dnotify

FILE RESCUE

We'll show you how you can avoid the tragedy of lost files with a transparent, Perl-based version control system.

BY MICHAEL SCHILLI

During early phases of a project, developers tend to experiment with various options, and sometimes it is too early to save prototypes in the version control system. If you haven't set up a repository, or if you haven't been able to agree on its structure, you might find yourself working without a safety net. In this case, good code might fall victim to an over-zealous *rm* * or your editor's delete command.

This month's Perl script, *noworries*, can give you automatic version control. Whenever you save a file with your editor, and whenever you use the shell to manipulate files using commands like *rm* or *mv*, a daemon hidden in the background receives a message. When it does, it picks up the new or modified file, and uses RCS to version the file. All of this is transparent to the user. Figure 1 shows a user creating and then deleting a new file in the Shell. Without some Perl wizardry, the file, *myfile* would have been gone for good, but calling *noworries -l myfile* tells us that the versioner created a backup copy just 17 seconds earlier. *noworries -r 1.1 myfile* retrieves the file and writes its content to STDOUT.

The script does not use manipulated shell functions or any other dirty tricks. Of course, an instance of the script needs to be running in the background – the *-w* (for "watch") option handles this – to start the File Alteration Monitor (FAM) utility [2], which in turn subscribes to the operating system kernel's Dnotify interface. Whenever the file system creates, moves, or deletes a directory or file, or manipulates file content, the kernel is notified of the event. The File Alteration Monitor (FAM) tells Dnotify that it is interested in what is going on in various directories and receives notifications in return. CPAN has a Perl module (SGI::FAM) that moves FAM's C interface to Perl. It is event-based and does not require CPU-intensive polling. Calling the *next_event()* method blocks the daemon until the next event occurs.

Figure 2 shows another example. In this case a file is created, and then modified twice in a row. The daemon receives a message for each event and creates three versions of the files in RCS (1.1, 1.2, and 1.3). Calling *noworries -l myfile* displays these versions, even if the file has been deleted in the meantime.

Asking for revision 1.2 by specifying *-r 1.2* and the filename *file* lets *noworries* retrieve the version after the first modification and prints its content to STDOUT. The shell command shown in Figure 2 redirects the output back to a file named *file*, which is immediately versioned again by the daemon. Figure 3 shows the daemon's activity: just to be on the safe side, the daemon logs its activities in the file */tmp/noworries.log*.

The *noworries* script takes care of files and directories, no matter how deeply they are nested, below *~/noworries* in the user's home directory. This is where you would typically set up new directo-

THE AUTHOR

Michael Schilli works as a Software Developer at Yahoo!, Sunnyvale, California. He wrote "Perl Power" for Addison-Wesley and can be contacted at *mschilli@perlmeister.com*. His homepage is at *http://perlmeister.com*.

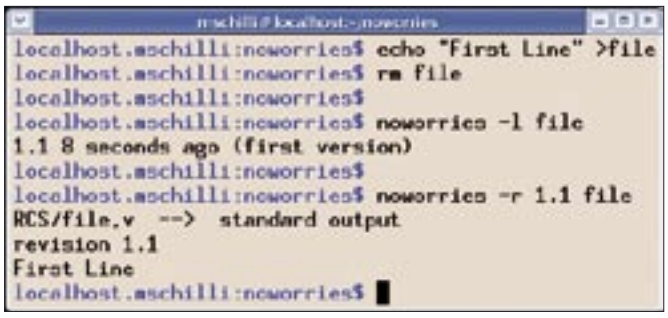

Figure 1: A Perl daemon works behind the scenes to bring a file back to life after it was deleted by a user.

ries, or extract tarballs if you wanted the protection of a safety net. The daemon creates a structure below *~/.noworries.rcs* to record the changes behind the scenes. Each subdirectory contains a *RCS* directory with the versioned files, traditionally named ending in *,v*. RCS has been a Unix tool from day one and is still used today by version control systems such as CVS or Perforce. The following set of commands checks in a version of *file*:

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

The program *ci* from the RCS command set creates *RCS/file,v* in the delta format used by RCS. The *co* command at the end, in combination with the *-l* (for "lock") option, restores the current version to the current directory. If you then modify *file*, and follow this up with another *ci/co* command sequence, you end up with two versions in *RCS/file,v*, which can be retrieved separately using *co*. The *rlog* program, another member of the RCS family, lets you view the meta-data for file versions you have checked in.

The *noworries* listing (Listing 1) defines the names of these tools in Lines 25 through 27. If you pass them to the script in this way, make sure they reside in your *PATH* to allow *noworries* to call them. If needed, you can hard code the full paths.

noworries uses the *mkd* (make directory), *cp* (file copy), *cd* (change directory), *cdback* (go back to original directory), and *tap* (execute a program and collect output) functions exported by *Sysadm::Install*. Regular readers of my Perl column may recall them from [4].

Noworries is Watching You!

Before *SGI::FAM* can receive messages about modified files below a directory, FAM first has to let the kernel know that it is interested in doing so. Events start to roll in after calling *$fam->monitor(...)* with *~/noworries* as its argument, whenever a new directory or file is created directly in *~/noworries*. However, this does not apply to any subdirectories. For this reason, *SGI::FAM* immediately launches another monitor for subdirectories whenever it notices that a new subdirectory has been created. A similar trick applies if *noworries* starts up when a deeply nested directory structure below *~/noworries* already exists. (We'll get to that in a moment.)

Setting the *-w* option launches *noworries* in daemon mode and runs the infinite loop defined in the *watcher* function in Line 88 of Listing 1. The call to the

next_event() method in Line 98 blocks the execution flow until one of four FAM-monitored events occurs. To find out which one of potentially many active directory monitors has triggered, the *SGI::FAM* object's *which()* method, which is called in Line 101, returns the directory that triggered the event. The event's *filename()* method returns the name of the new, existing, modified, or deleted object, which can be a directory or a file.

The *type()* method gives us the event type. The types that *noworries* is interested in are *create* and *change*. The *monitor()* method adds new directories to the list of things to watch, while the *check_in()* function defined in Line 170

Listing 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:wl",
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
```

handles new or modified files. A similar approach is used for adding directories. The daemon uses *find* to locate directories when launched, assuming that *~/noworries* already exists. The *subdirs()* helper function in Line 153 digs down deeper and deeper into the directory tree and returns any directories it finds no matter how deeply nested they may be. The *watch_subdirs()* function iterates over all of them and passes their relative pathnames to FAM for surveillance.

The documentation section in Line 266 is not just for convenient access to a nicely formatted manual page whenever a user calls *perldoc noworries*. It is also output by the *pod2usage()* function, if the user fails to provide the required

Listing 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       $dur->in_units($_);
223     return "$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
```

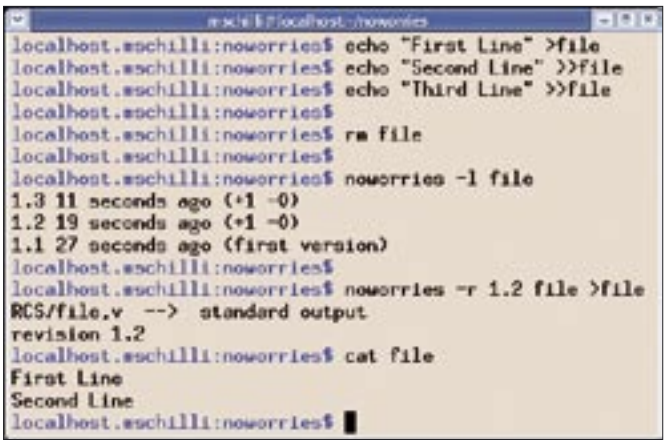

Figure 2: Two lines are added to a newly created file in two subsequent editing sessions. Noworries retrieves version 2 on request.

command line options. It does not make much sense to version temporary *vi* or *emacs* files, so they are filtered out in Lines 107 through 112.

When a file needs to be checked into the version control system, *check_in* in Line 170 first checks if the file is a text file. *check_in* discards binary files in Line 174. The function is called with a pathname relative to *~/noworries*, as this is where *watcher()* jumps to in Line 90. Line 189 copies the original file to the RCS tree, and Line 195 calls the *ci* tool with the *-t* and *-m* options. It passes a value of - to both, as the first – and all following – check-in comments are meaningless. But you have to give *ci* something to chew on to avoid an interactive prompt. Line 204 checks the file out, as described earlier on. The next time a change occurs, the checked out copy is overwritten, and the new version is checked in by *ci*.

What's the Date, Today?

noworries calls the RCS *rlog* function to find out which versions of a file are available. *rlog* returns the version numbers with the date (formated as *yyyy/mm/dd hh:mm:ss*) and also reveals the number of lines that have changed in comparison to the previous version. Of course, it can't give us this information for the initial version, but if you are told that version 1.2 has *lines: +10 -0*, this means there are 10 new lines in comparison to 1.1, and that no lines have been deleted.

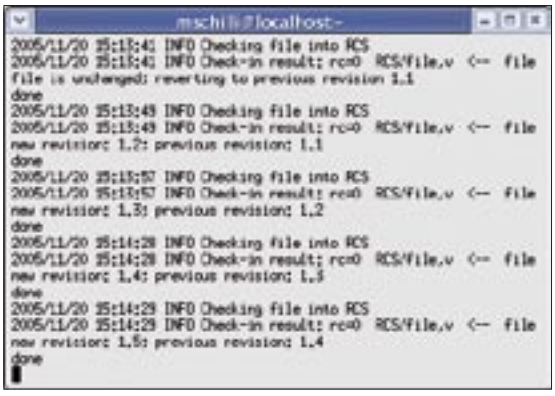

Figure 3: Behind the scenes, the daemon monitors the file system and creates a versioned backup copy whenever a change occurs in the monitored directories.

The *DateTime* module from CPAN helps tremendously with date calculations. The *DateTime::Format::Strptime* module parses the RCS date information, and converts the value to seconds after the epoch. To do this, the constructor expects a format string with the following pattern: "*%Y/%m/%d %H:%M:%S*", and the call to *parse_datetime()* returns a fully initialized *DateTime* object if successful. The *while* loop that starts in Line 241 navigates the slightly overwhelming output by the *rlog* helper, using a multiple-line regular expression to do so.

The *time_diff()* function in Line 211 expects a *DateTime* object and calculates how old a version is in seconds, minutes, hours, days, or weeks. This is easier to read for the heavy *noworries* user.

Unfortunately, *Dnotify*, the mechanism used by FAM, doesn't scale well and bows out at around two hundred subdirectories. To solve this problem *dnotify* has been replaced by *inotify* in more recent kernels. *inotify* makes better use of resources and scales more easily. FAM is also obsolete, and Gamin [3] its designated successor.

The kernel's *Dnotify* mechanism does not use file system inodes, but filenames, so that *mv file1 file2* triggers two events: a delete type and a create type event. This does not bother *noworries*, as the script ignores delete events, and if the same file appears some time later, it is just checked in as the latest version.

The script should only be used on your local hard disk, and not with NFS, as FAM can only be efficient if the NFS target is also running a FAM. If not, it polls the target at regular intervals, and this makes the whole thing somewhat ineffective.

Installation

You need to install the SGI::FAM, Sysadm::Install, DateTime, DateTime::Format::Strptime, and Pod::Usage CPAN modules; a CPAN Shell scan will help to quickly resolve the dependencies. If you see a *FAM.c:813: error: storage size of 'RETVAL' isn't known* error when building SGI::FAM, change Line 813 in *FAM.c* from *enum FAMCodes RETVAL;* to *FAMCodes RETVAL;*; re-running *make* should then give you the goodies.

To make sure that the daemon is always running, add a line such as *x777:3:respawn:su mschilli -c "/home/mschilli/bin/noworries -w"* to */etc/inittab*, and then let the *Init* daemon know by running *init q*. The process has to run with the ID of the current user (*mschilli* in this case) to ensure that *$ENV{HOME}* in the script points to the right home directory. In this case, the *init* process launches the *noworries* daemon when you boot your machine, and the *respawn* option ensures that the process restarts immediately if for some reason it is inadvertently terminated. But before you do all of this, test the daemon on the command line to see if everything is working properly.

The *-d* for *debug* option might be a help if you are experiencing problems; it displays detailed status information on standard output rather than logging in */tmp/noworries.log*. ■

INFO

[1] Listing for this article: *http://www.linux-magazine.com/Magazine/Downloads/63/Perl*

[2] FAM Homepage: *http://oss.sgi.com/projects/fam/*

[3] Gamin Homepage: *http://www.gnome.org/~veillard/gamin/*

[4] "Perl Shell Scripts," Michael Schilli: *http://www.linux-magazine.com/issue/52/Perl_Shell_Scripts.pdf*