Integrating Mantis and CVS/CVSNT (http://www.sandon.it/mantis_cvsnt)

Posted on: Thu, 12/28/2006 - 21:46 **By:** Administrator

Information Technology

Development (http://www.sandon.it/taxonomy/term/1)

Tools (http://www.sandon.it/taxonomy/term/3)

- Log in (http://www.sandon.it/user/login?destination=/comment/reply/node/6/comment_node_page%23comment-form) to post comments

Currently CVS comes in two versions: the original CVS (available at http://www.nongnu.org/cvs/), and CVSNT (www.cvsnt.org), originally a native port to Windows systems, but then become a whole project on its own available on many platforms (Windows, Linux, Unix, MacOSX) and with several new features. Both versions share a common architecture, and from now on "CVS" will refer to both versions, but where stated explicitly. **CVS triggers** During the execution of CVS commands CVS can invoke and execute "triggers", external scripts or programs, at specific times, before or after particular events. Details about triggers could be found in CVS documentation. The "commit support files", files containing triggers invoked during a commit are those interesting to Mantis administrators. All these files support a "common syntax". Each file is made up of lines. Blank lines are ignored. Lines starting with # are comments. Each lines syntax is:

<regular_expression|ALL|DEFAULT><whitespace><command><format strings>

regular_expression is an emacs GNU compatible expression for CVS and a PCRE compatible expression for CVSNT. The first regular expression that matches the current path within the repository executes the associated command (CVSNT can execute more than one match). If there is no match, the first line beginning with DEFAULT is used. All lines beginning with ALL are executed. <command> can be a shell script, an executable, etc. <format strings> are special substitution parameters available to pass data to the script. They consist in % followed by a character, i.e. %s The commit support files are:

- *commitinfo*: invoked *before* a commit takes place. If the script returns with a non-zero value the commits are aborted. AFAIK, CVS will not pass the log message to the script, while CVSNT can. Anyway, use this script only if you want to abort the commit if there is something to fix. Otherwise a non properly written script can block commits.

- *verifymsg*: invoked to check the log message entered by the user. The log message is passed as a path to a file. The ALL keyword is not supported, making it less useful for a Mantis integration. If the invoked script/executable returns with a non-zero code, the commit is aborted. Again, I wouldn't use this for Mantis. It is useful to check if the log message contains all required information, and modify/reject it if not.
- *loginfo*: used to control where cvs commit log information is sent. IMHO this is the best script to integrate CVS with Mantis. It can pass several information and can't abort a commit if the script contains an error. CVS will pass log information in STDIN, while CVSNT can use STDIN or command line parameters. Because with a remote CVS server updates take place in the remote repository, CVS sends some unuseful information in STDIN, i.e. the directory where the update of files takes place. CVSNT allows to pass only the log message and/or any required information.

Also, remember to read all data from STDIN, or CVS can return with a "broken pipe" error. **Using loginfo for Mantis integration** To check *any* commit for a Mantis issue number, we can add a line like: ALL c:/php/php.exe C:/mantis/core/checkin.php The standard checkin.php will read from STDIN, and will try to match the "commit regexp". IMHO, that script contains a bug because it reads chunks of 1024 bytes and try to match within the chunk. If the string happens to be split between two reads, it won't be matched. I modified code as follows - I am not a PHP developer, hope it is correct:

```
[php]// Build the whole comment line
while ( ( $t_line = fgets( STDIN, 1024 ) ) ) {
$t_comment .= $t_line;
  }

// match the regexp
if ( preg_match_all( $t_commit_regexp, $t_comment, $t_matches ) ) {
  for ( $i = 0; $i < count( $t_matches[0] ); ++$i ) {
  $t_issues[] = $t_matches[1][$i];
  }
}[/php]
```

This is could be a basic setup. We can use some of the format strings to pass more informations, or change how they are passed. For example in my CVSNT setup I use: ALL c:/php/php.exe C:/mantis/core/checkin.php %m %s %u that passes the log message (%m), modified module and file (%s) and the user who commited (%u) on the script command line. Therefore I modified the checkin.php file as follows:

```
[php]
/* removed all the stdin stuff */

$t_comment = $_SERVER['argv'][1];

if ( preg_match_all( $t_commit_regexp, $t_comment, $t_matches ) ) {
for ( $i = 0; $i < count( $t_matches[0] ); ++$i ) {
$t_issues[] = $t_matches[1][$i];
  }
}
```

```
/* ... */

/* if there has been a match */

$t_comment .= "\nModified module and files: " . $_SERVER['argv'][2] .
  "\nBy: " $_SERVER['argv'][3];

# Call the custom function to register the checkin on each issue.
[/php]
```

Of course many other combinations are possible. Recent CVSNT releases have a native "bugid" that can be assigned to files while committing or editing (cvs commit -B and cvs edit -b) that would allow for a "native" integration: ALL c:/php/php.exe C:/mantis/core/checkin.php %m %b %s %u where %b passes the bugid to the script, without having to match it with a regular epression.