PHPLint

```
#!/usr/bin/env php
<?php


  /**
   * Linting Command Class
   *
   * @package CodeRobot
   **/
  class Lint_Command {


    private $count   = 0;
    private $errors  = array();
    private $options = NULL;
    private $parse   = NULL;
    private $files   = FALSE;
    private $git     = NULL;


    /**
     * Command constructor
     *
     * @return void
     **/
    public function __construct() {
      $this->path  = $_SERVER['PWD'];
      $this->files = $this->get_piped_files();
      $this->run();
    }


    /**
     * Run the command
     *
     * @return void
     **/
    private function run() {
      echo 'Linting files against PHP ' . phpversion() . PHP_EOL;
      if ($this->files) {
        foreach ($this->files as $file) {
          $this->check_file($path . '/' . $file);
        }
      } else { // Use arguments
        if ($_SERVER['argc'] > 1) {
          $last = end($_SERVER['argv']);
          if (substr($last, 0, 1) != '-') {
```

```
            $this->path = $last; // snag last argument, if it wasn't an
option switch
          }
        }

        $this->parse_options();
        $this->set_options();

        if (is_dir($this->path)) {
          $this->check_directory_contents($this->path);
        } elseif (is_file($this->path)) {
          $this->check_file($this->path);
        } else {
          echo $this->path . 'is not a file or directory.' . PHP_EOL;
          $this->display_help();
        }
      }
      $this->run_complete();
    }


    /**
     * Show the help menu
     *
     * @return void
     **/
    public function display_help() {
      echo PHP_EOL . 'usage: lint [-qR] [path]' . PHP_EOL . PHP_EOL .
           'options:' . PHP_EOL .
            '  -q, --quiet:     disable verbose output' . PHP_EOL .
            '  -b, --blame:     display git blame along with error messages'
. PHP_EOL .
            '  --exclude:       comma separated list of folder patterns to
exclude' . PHP_EOL .
            '  -h, --help:      display this help screen' . PHP_EOL .
PHP_EOL;
      exit(1);
    }


    /**
     * Check the directory contents for PHP files
     *
     * @param string $dir Path to the directory to check
     *
     * @return void
     **/
    function check_directory_contents($dir) {
      $di = new DirectoryIterator($dir);
      foreach ($di as $file) {
        // Ignore common junk
```

```
        if ($file->isDot() || ($file->getFilename() == '.git') ||
($file->getFilename() == '.svn')) {
          continue;
        }

        if ($this->matches_exclude_list($dir . '/' . $file->getFilename()))
{
          continue;
        }

        if ($file->isDir()) {
          $this->check_directory_contents($dir . '/' .
$file->getFilename());
          continue;
        }

        if ($file->isFile()) {
          $this->check_file($dir . '/' . $file->getFilename());
        }
      }
    }


    /**
     * Check the file for syntax errors
     *
     * @param string $path Path to the file to check
     *
     * @return void
     **/
    private function check_file($path) {
      // Skip non-php files
      if (substr($path, -4) != '.php') {
        return;
      }

      if (($this->count % 60 == 0)) {
        echo PHP_EOL;
      }

      $error = `php -l $path 2>&1 1> /dev/null`;
      if ($error) {
        preg_match('/line ([0-9]+)/i', $error, $line);
        $line = $line[1];
        $this->errors[] = (object)array(
          'message' => $error,
          'path'    => $path,
          'line'    => $line
        );
        echo 'E';
      } else {
```

```
      echo '.';
    }

    $this->count++;
  }


  /**
   * Set the options for this run
   *
   * @return void
   **/
  private function set_options() {
    $args = array_keys(getopt('qRhb'));
    $this->options->quiet = FALSE;
    $this->options->blame = FALSE;
    foreach ($args as $arg) {
      switch ($arg) {
        case 'q':
        case 'quiet':
          $this->options->quiet = TRUE;
          ob_start();
        break;
        case 'b':
        case 'blame':
          if ($this->is_blame_possible()) {
            $this->options->blame = TRUE;
          }
        break;
        case 'h':
        case 'help':
        default:
          $this->display_help();
          exit(0);
        break;
      }
    }
    if (empty($this->options->exclude)) {
      $this->options->exclude = array();
    }
  }


  /**
   * Parse the command line arguments
   *
   * @return void
   **/
  private function parse_options() {
    $this->options = (object)getopt('qhb', array(
      'quiet::',
```

```
        'help::',
        'exclude::',
        'blame'
      ));
      if (!empty($this->options->exclude)) {
        $this->options->exclude = explode(',', $this->options->exclude);
        for ($i = 0; $i < count($this->options->exclude); $i++) {
          $this->options->exclude[$i] = str_replace('*', '.+',
$this->options->exclude[$i]);
        }
      }
    }


    /**
     * Get the piped files
     *
     * @return array
     **/
    private function get_piped_files() {
      $files = array();
      stream_set_blocking(STDIN, FALSE);
      while ($line = trim(fgets(STDIN))) {
        $files[] = $line;
      }
      return $files;
    }


    /**
     * Find if this path is excluded
     *
     * @param string $path Path to the file or directory
     *
     * @return boolean
     **/
    public function matches_exclude_list($path) {
      foreach ($this->options->exclude as $rule) {
        if (preg_match('%.' . $rule . '.+%i', $path)) {
          return TRUE;
        }
      }
      return FALSE;
    }


    /**
     * The run is complete, finish and clean up
     *
     * @return void
     **/
```

```
    private function run_complete() {
      echo PHP_EOL . $this->count . ' files checked, ' .
count($this->errors) . ' errors.' . PHP_EOL;

      foreach ($this->errors as $error) {
        echo $error->message;
        if ($this->options->blame) {
          $this->find_blame($error);
        }
      }

      if ($this->options->quiet) {
        ob_end_clean();
      }
      if (!empty($this->errors)) {
        exit(1);
      } else {
        exit(0);
      }
    }


    /**
     * Find who caused the error
     *
     * @param object $error The error path and message
     *
     * @return void
     **/
    private function find_blame($error) {
      $lines_in_file = (int)trim(`wc -l < $error->path`);
      if ($error->line > $lines_in_file) {
        $error->line = $lines_in_file;
      }
      $blame     = `git blame $error->path -L $error->line,$error->line --
porcelain`;
      $git_lines = explode(PHP_EOL, trim($blame));
      $blame     = array();
      foreach ($git_lines as $git_line) {
        @list($var, $value) = explode(' ', $git_line, 2);
        $blame[$var] = $value;
      }
      if ($blame['author-mail'] == '<not.committed.yet>') {
        echo '    Caused by you.' . PHP_EOL;
      } else {
        echo '    Caused by ' . $blame['author'] . ' ' . $blame['author-
mail'] . PHP_EOL;
      }
    }
```

```
    /**
     * Check if blame is possible, quit with error if not
     *
     * @return boolean
     **/
    private function is_blame_possible() {
      if ($this->is_git_installed()) {
        if ($this->is_git_repo()) {
          if (!$this->git_has_history()) {
            echo 'Cannot run blame, git repo does not have any commit
history.' . PHP_EOL;
            exit(1);
          }
        } else {
          echo 'Cannot run blame, path is not a git repo or you are ' .
               'trying to run lint from outside the repo.' . PHP_EOL;
          exit(1);
        }
      } else {
        echo 'Cannot run blame, git is not installed.' . PHP_EOL;
        exit(1);
      }
      return TRUE;
    }


    /**
     * Find if git is installed on this server
     *
     * @return boolean
     **/
    private function is_git_installed() {
      $git = trim(`which git`);
      return (!empty($git));
    }


    /**
     * Find if the path belongs to a git repo
     *
     * @return boolean
     **/
    private function is_git_repo() {
      $cmd = 'git status ' . $this->path . ' 2>&1';
      $response = trim(`$cmd`);
      if (strpos($response, 'fatal') !== FALSE) {
        return FALSE;
      } else {
        return TRUE;
      }
    }
```

```
    /**
     * Find if the git repo has commit history
     *
     * @return boolean
     **/
    private function git_has_history() {
      $cmd = 'git log ' . $this->path . ' 2>&1';
      $response = trim(`$cmd`);
      if (strpos($response, 'fatal') !== FALSE) {
        return FALSE;
      } else {
        return TRUE;
      }
    }


  }


  // Hold on to your butts...
  $lint = new Lint_Command();
```

```
chmod 755 /usr/local/bin/phplint
```

https://github.com/dprevite/lint