# 代码上库前的自动review实现

mnstory.net

perlcritic是一款PERL的静态代码检测工具，官网为：<http://www.perlcritic.org>

源码下载：<https://github.com/Perl-Critic/Perl-Critic>，可以参考 <http://search.cpan.org/dist/Perl-Critic/> 使用，可以使用CPAN安装：

perl -MCPAN -e shell

> install Perl::Critic

本日志自动扫描工具建立在Critic基础上，为我们的日志检测规则写了一插件，名为RequireLogStatment，安装文件覆盖/usr/local/share/perl/5.14.2/Perl/Critic/Policy/CodeLayout/RequireLogStatment.pm 即可使用。

## 检测规则

针对部门使用大量PERL代码做日志规范，当前主要检查该打日志的地方未打日志的行为，该打日志的检查点为:

EXITWORD：return, exit, \_exit, die

打日志的函数为：

LOGWORD：ldie lemerg lalert lcrit lerror lwarn lnotice linfo ldebug dbg1 dbg2 dbg3 dbgv

不是所有EXITWORD前都要打日志，且日志级别不一定是lerror，例如一个函数运行最后一条return语句作为合法出口不要求打日志，为了避免日志过多，建议大家按照“先处理错误，最后处理正确原则”书写代码，例如：

*sub* isUserAllowed {

    my ($name, $age) = @\_;

    if(!$name) {

        lerror("Invalid param ".($name?$name:'undef')); #错误处理1

        return 0;

    }

    if($age < 18) {

        lerror("Adults Only"); #错误处理2

        return 0;

    }

    #正确情况

    print $name;

    return 1;

}

如果某些语句真不需要日志，可以添加annotation，规则是在语句前一条注释处加：

@logcheck no

例如：

foreach my $net (@$vm\_nets) {

    #@logcheck no

    return 1 if ($net->{type} eq 'bvs');

}

## 日志检测插件与效果

perlcritic使用PERL编写，分析其原理，然后实现如下代码：

package Perl::Critic::Policy::CodeLayout::RequireLogStatment;

use 5.006001;

use strict;

use warnings;

use Readonly;

use Data::Dumper;

use Carp;

use Perl::Critic::Utils qw{ :characters :severities };

use base 'Perl::Critic::Policy';

our $VERSION = '1.0';

my $transLocal = 1;

my $DESC=$transLocal?'日志规范检查：':'Log missing ';

my $EXPL=29219;

*sub* supported\_parameters { return ()                     }

*sub* default\_severity     { return $SEVERITY\_HIGH       }

*sub* default\_themes       { return qw(core bugs pbp cosmetic)  }

*sub* applies\_to           { return 'PPI::Token::Word' }

# base functions

*sub* ltrim { my $s = shift; $s =~ s/^\s+//;       return $s };

*sub* rtrim { my $s = shift; $s =~ s/\s+$//;       return $s };

*sub*  trim { my $s = shift; $s =~ s/^\s+|\s+$//g; return $s };

# check stat functions

use constant CheckRight  => 1;

use constant CheckWrong  => 2;

use constant CheckPass => 3;

my $checkRight = 0;

my $checkWrong = 0;

my $checkPass = 0;

*sub* addCheckRecord {

    my ($chktype, $elem) = @\_;

    if($chktype == CheckRight) {

        ++$checkRight;

        #printf(ele2str($elem)."\n");

    } elsif($chktype == CheckWrong) {

        ++$checkWrong;

    } elsif($chktype == CheckPass) {

        ++$checkPass;

    }

}

*sub* volationEntry {

    my ($self, $elem, $desc) = @\_;

    addCheckRecord(CheckWrong, $elem);

    return $self->violation($DESC.$desc, $EXPL, $elem);

}

# helper functions

*sub* existsIn {

    my ($word, $words) = @\_;

    for my $i (@$words) {

        if($word eq $i) {

            return 1;

        }

    }

    return 0;

}

my @exitWords = qw(return die exit \_exit);

*sub* isExitWord {

    my ($word) = @\_;

    return existsIn($word, \@exitWords);

}

my @logWords = qw(ldie lemerg lalert lcrit lerror lwarn lnotice linfo ldebug dbg1 dbg2 dbg3 dbgv);

*sub* isLogWord {

    my ($word) = @\_;

    return existsIn($word, \@logWords);

}

my @conditionJoinWords = qw(|| && or and);

*sub* isConditionJoinWord {

    my ($word) = @\_;

    return existsIn($word, \@conditionJoinWords);

}

my @conditionWords = qw(unless if);

*sub* isConditionWord {

    my ($word) = @\_;

    return existsIn($word, \@conditionWords);

}

*sub* ele2str {

    my ($elem) = @\_;

    return $elem->location()->[4].":".$elem->location()->[0]." ".$elem->content()."\n";

}

*sub* sfindNextType {

    my ($nsib, $type) = @\_;

    while ( $nsib && ! $nsib->isa($type) ) {

        $nsib = $nsib->snext\_sibling();

    }

    return $nsib;

}

*sub* sfindNextTypeAll {

    my ($nsib, $type) = @\_;

    my @res = ();

    while ($nsib) {

        if ($nsib->isa($type)) {

            push @res, $nsib;

        }

        $nsib = $nsib->snext\_sibling();

    }

    return @res;

}

*sub* findIfBlock {

    my ($if) = @\_;

    if (!$if || !$if->isa('PPI::Statement::Compound')) {

        printf(ele2str($if)." is not if statement!\n");

        return;

    }

    return sfindNextTypeAll($if->schild(0), 'PPI::Structure::Block');

}

*sub* slast {

    my $next\_sibling = shift;

    my $last\_following\_sibling = $next\_sibling;

    if (!$next\_sibling) {

        Carp::cluck("null next sibling");

        return undef;

    }

    while ($next\_sibling = $next\_sibling->snext\_sibling()) {

        $last\_following\_sibling = $next\_sibling;

    }

    # if the last is ;, use previous

    if ($last\_following\_sibling

        && $last\_following\_sibling->isa('PPI::Token::Structure')

        && ';' eq $last\_following\_sibling->content()) {

        return $last\_following\_sibling->sprevious\_sibling();

    }

    return $last\_following\_sibling;

}

*sub* \_commentExtra {

    my ($comment) = @\_;

    if (!$comment) {

        return "";

    }

    my $offset = index($comment, '#');

    if (-1 == $offset) {

        printf("not a comment: ".$comment."\n");

        return "";

    }

    $comment = substr($comment, $offset+1);

    return trim($comment);

}

*sub* annotationParse {

    my ($statement) = @\_;

    if (!$statement || 0 != index($statement, '@logcheck ')) {

        return ();

    }

    my @annos = split(/[\t ]+/, $statement);

    shift @annos;

    return @annos;

}

*sub* processStatement {

    my ($self, $elem, $desc, $location) = @\_;

    my $ftoken = $elem->schild(0);

    if (!$ftoken) {

        printf(Dumper($elem)." no first child!\n");

        return;

    }

    if (isLogWord($ftoken->content())) {

        return; #condition right

    }

    if ($ftoken->content() eq "if") {

        return processIf($self, $elem, $desc, $location);

    }

    return volationEntry($self, $location?$location:$ftoken, $desc?$desc:$transLocal?"PRE LOG MISSING 退出请打日志":"PRE LOG MISSING");

}

*sub* processIf {

    my ($self, $elem, $desc, $location) = @\_;

    my @ifblks = findIfBlock($elem);

    for(my $i = 0; $i < @ifblks; $i++){

        my $ifblk = $ifblks[$i];

        if(!$ifblk) {

            printf("can't found ele's ifblk:".Dumper($elem)."\n");

            next;

        }

        my $firstblock = $ifblk->schild(0);

        if(!$firstblock) {

            #Carp::cluck("null schild ".ele2str($ifblk));

            # e.g: if($node ne 'cluster') {}

            # with empty block!

            next;

        }

        my $last = slast($firstblock);

        if (!$last) {

            printf("can't found ifblk's last:".Dumper($ifblk)."\n");

            next;

        }

        my $violate = process($self, $last, $desc, $location);

        if ($violate) {

            return $violate;

        }

    }

    addCheckRecord(CheckRight, $elem);

    return;

}

*sub* process {

    my ($self, $elem, $desc, $location) = @\_;

    if (!$elem) {

        printf("statement is null\n");

        return;

    }

    if ($elem->isa('PPI::Statement::Compound')) {

        return processIf($self, $elem, $desc, $location);

    } elsif ($elem->isa('PPI::Statement')) {

        return processStatement($self, $elem, $desc, $location);

    } else {

        printf("unknown statement type: ".Dumper($elem)."\n");

    }

    return;

}

*sub* \_isLastStatOfSub {

    my ($stat) = @\_;

    my $sp = $stat->parent();

    if(!$sp) {

        #printf("stat no parent: ".Dumper($stat)."\n");

        return 0;

    }

    my $findLast = slast($sp->schild(0));

    if ((refaddr $findLast) == (refaddr $stat)) {

        return 1;

    }

    return 0;

}

*sub* isLastStatOfSub {

    my ($stat) = @\_;

    while ($stat && \_isLastStatOfSub($stat)) {

        my $sp  = $stat ? $stat->parent() : undef;

        my $spp = $sp ? $sp->parent() : undef;

        if ($spp && $spp->isa('PPI::Statement::Sub') && $sp->isa('PPI::Structure::Block')) {

            return 1;

        }

        $stat = $sp;

    }

    return 0;

}

*sub* violates {

    my ( $self, $elem, undef ) = @\_;

    if(!isExitWord($elem->content()))  {

        return;

    }

    my $statement = $elem->statement();

    if (!$statement) {

        printf(Dumper($elem)." no statement!\n");

        return;

    }

    #CASE sub except

    if (isLastStatOfSub($statement)) {

        addCheckRecord(CheckRight, $statement);

        return;

    }

    #CASE annotation

    my $presib = $statement->previous\_sibling();

    while ($presib && $presib->isa('PPI::Token::Whitespace')) {

        $presib = $presib->previous\_sibling();

    }

    if($presib && $presib->isa('PPI::Token::Comment')) {

        my @annos = annotationParse(\_commentExtra($presib->content()));

        if (@annos && ("no" eq $annos[0])) {

            addCheckRecord(CheckPass, $presib);

            return;

        }

    }

    #CASE or return

    $presib = $elem->sprevious\_sibling();

    if ($presib && $presib->isa('PPI::Token::Operator') && isConditionJoinWord($presib->content())) {

        return volationEntry($self, $elem, $transLocal?"OR RETURN 退出前请打日志":"OR RETURN need log");

    }

    #CASE return if

    my $lastsib = slast($elem);

    if($lastsib && $lastsib->isa('PPI::Structure::Condition')) {

        my $if = $lastsib->sprevious\_sibling();

        if ($if && isConditionWord($if->content())) {

            return volationEntry($self, $elem, $transLocal?"RETURN IF 退出前请打日志":"RETURN IF need log");

        }

        printf("invalid condition found: ".Dumper($lastsib)."\n");

    }

    #CASE no pre-statement

    my $prestatement = $statement->sprevious\_sibling();

    if(!$prestatement) {

        return volationEntry($self, $elem, $transLocal?"NO PRE-STATEMENT 退出前请打日志":"NO PRE-STATEMENT");

    }

    #CASE if-else return

    if ($prestatement->isa('PPI::Statement::Compound')) {

        #printf("case 1, ".ele2str($prestatement));

        return processIf($self, $prestatement, $transLocal?"IF-ELSE RETURN 退出前请打日志":"IF-ELSE RETURN need log", $elem);

    }

    #CASE normal check

    if ($prestatement->isa('PPI::Statement')) {

        my $ftoken = $prestatement->schild(0);

        if ($ftoken) {

            if(isLogWord($ftoken->content())) {

                #pre statement is a logstat

                addCheckRecord(CheckRight, $ftoken);

                return;

            } elsif(isConditionWord($ftoken->content())) {

                return processIf($self, $prestatement, $transLocal?"IF-ELSE RETURN 退出前请打日志":"IF-ELSE RETURN need log", $elem);

            }

        }

        return volationEntry($self, $elem, $transLocal?"PRE LOG MISSING 退出前请打日志":"PRE LOG MISSING need log");

    }

    printf("unknown pre statement type: ".Dumper($prestatement)."\n");

    return;

}

END {

    printf("LogCheck Right $checkRight Wrong $checkWrong Pass $checkPass\n");

}

1;

目前可检测如下语法unit.pm：

#[W] no pre-statement

return 4;

#[P] annotation

if (-e "path/to/file") {

    i++;

    #@logcheck no

    exit (0); #PPI::Statement

}

#[R] if return

if (-e "path/to/file") {

    i++;

    ldebug("file found!");

    exit (0); #PPI::Statement

}

#[R] if if return

if (-e "path/to/file") {

    if (true){

        ldebug("file found!");

    }

    exit 1; #PPI::Statement

}

#[W] if-else return

if (-e "path/to/file") {

    if ($i == 1){

        lerror("file found 1!");

    } elsif ($i == 2){

        ldebug("file found 2!");

    } else {

        i++;

    }

    exit 1; #PPI::Statement

}

#[R] if-else return

if (-e "path/to/file") {

    if ($i == 1){

        lerror("file found 1!");

    } elsif ($i == 2){

        ldebug("file found 2!");

    } else {

        i++;

        linfo("file found 3!");

    }

    exit 1; #PPI::Statement

}

#[W] if-else return

if (-e "path/to/file") {

    if ($i == 1){

        lerror("file found 1!");

    } elsif ($i == 2){

        i++;

    } else {

        ldebug("file found 2!");

    }

    exit 1; #PPI::Statement

}

#[W] if-else return

if (-e "path/to/file") {

    if ($i == 1){

        i++;

    } elsif ($i == 2){

        lerror("file found 1!");

    } else {

        ldebug("file found 2!");

    }

    exit 1; #PPI::Statement

}

#[W] pre log missing

while (true) {

    i++;

    exit 2;

}

#[W] return if

return (-1, "file not found") if(! -e "path/to/file"); #statement=PPI::Structure::Condition,PPI::Token::Structure=';'

#[W] return unless

return $vmlist unless (-e $powerfile);

#[W] or return

open STDOUT, '>/dev/null' or die "$msg $!\n";

#[W] or return

! -e "path/to/file" || die "file not exist!"; #psib=PPI::Token::Operator='||'

#[W] sub except

*sub* foo {

    my ($arg) = @\_;

    i++;

    if (i > 0) {

        return i;

    }

    i--;

}

#[R] sub except

*sub* foo {

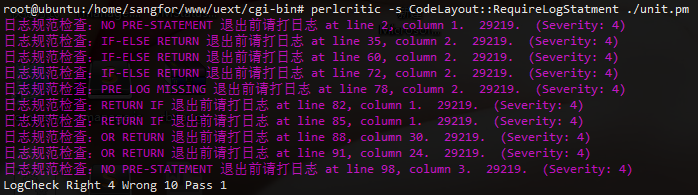
    my ($arg) = @\_;

    i++;

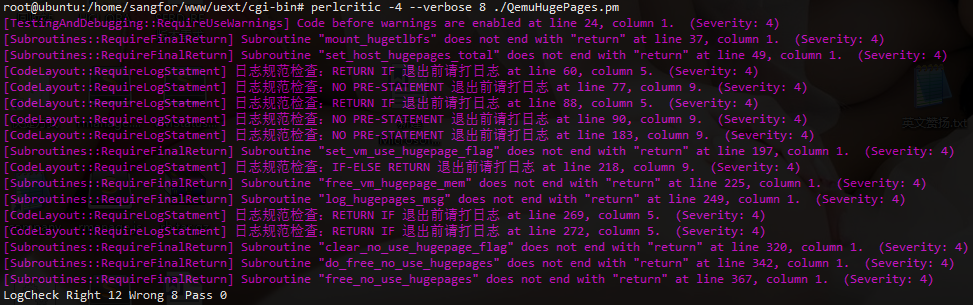
    return i;

}

此工具可单独使用：perlcritic -s CodeLayout::RequireLogStatment ./unit.pm



也可配合其他检测选项使用：perlcritic -4 --verbose 8 ./QemuHugePages.pm



## 根据reviewID扫描并提交reviewboard

上面命令实现后，想要完成自动化，需要实现根据reviewboard的reviewID自动获取代码进行检测，并将检测结果提交成comments形式给reviewboard，由于这部分使用大量Restful API请求，使用js处理request和JSON非常方便，但是node.js最大的问题是默认异步，导致代码并不是很优雅：

#!/bin/node

/\*\*

\* syntex check and submit comments to reviewboard

\*     all api reference https://www.reviewboard.org/docs/manual/2.0/webapi/

\*     su -c "./rbjob -r 21371 retrive --check log --flog 2>&1" www-data

\*     su -c "./rbjob -r 21371 retrive --check all" www-data

\* @author mnstory.net

\* @version 20160910 - check syntex and post uniq comments

\*/

//public defines

*var* suspend = require('suspend'),

    resume = suspend.resume;

const fs = require('fs')

*var* path = require('path')

*var* request = require('request')

*var* p = require('child\_process')

*var* moment = require('moment')

*function* mkdirsSync(*targets*) {

    if (fs.existsSync(targets)) {

        return true

    }

*var* joinTarget

    targets.split(path.sep).forEach(*function*(dirname) {

        if(!joinTarget && !dirname) {

            joinTarget='/'

            return true

        }

        joinTarget = joinTarget ? path.join(joinTarget, dirname) : dirname

        if (!fs.existsSync(joinTarget)) {

            fs.mkdirSync(joinTarget)

        }

    })

    return fs.existsSync(targets)

}

*function* strArrBeginWith(*strArr, strMatch*) {

    for (i in strArr) {

        if(strArr[i].indexOf(strMatch) == 0) {

            return true

        }

    }

    return false

}

*function* extractLastURLToken(*url*) {

    if(!url || url.length <= 2) {

        l.error('invalid url ' + (url?url:'<null>'))

        return null

    }

*var* lastOffset = ('/' == url[url.length - 1]) ? 1 : 0

*var* offset = url.lastIndexOf('/', url.length-1-lastOffset)

    if(-1 == offset) {

        l.error('url ' + url + ' not expected format')

        return null

    }

    return url.substring(offset + 1, url.length-lastOffset)

}

mkdirsSync('/var/log/rb')

*var* log4js = require('log4js')

log4js.configure({

    appenders: [

        {

            type: 'console',

            category: "consoleRBJob"

        },

        {

            type: 'dateFile',

            filename: '/var/log/rb/job',

            pattern: ".yyyy-MM-dd.log",

            maxLogSize: 4096,

            alwaysIncludePattern: true,

            backups: 1,

            category: 'RBJob',

            layout: {

                type: 'pattern',

                pattern: "%d{ABSOLUTE} %-5p " + process.pid +" %c %m"

            }

        },

    ],

    replaceConsole: false,

    levels: {

        RBJob: 'trace',

        consoleRBJob: 'trace'

    }

})

*var* l = log4js.getLogger('consoleRBJob')

//module defines

*var* reviewURI

*var* reviewRequestOption

*var* outDir='/tmp/rb/job'

*var* sessionDir=null

*function* optParse(*options*) {

    if(options.flog) {

        l = log4js.getLogger('RBJob')

    }

    p.execSync('rm -rf ' + outDir)

    reviewURI='http://'+options.rserver+'/'

    reviewRequestOption = {

        'auth': {

            'user': options.ruser,

            'pass': options.rpass,

            'sendImmediately': false

        }

    }

    l.debug('reviewURI = ' + reviewURI)

    l.debug('reviewRequestOption = ' + JSON.stringify(reviewRequestOption))

    return true

}

*function* commentsParse(*strComments*) {

    if(!strComments) {

        l.error('strComments is empty when parse comments')

        return null

    }

*var* res = null

*var* comments = strComments.split("\n")

*var* regex = /at line ([0-9]+),/

    for (*var* idx = 0; idx < comments.length; ++idx) {

        //Log missing NO PRE-STATEMENT at line 99, column 9.  29219.  (Severity: 4)

*var* matched = comments[idx].match(regex)

        if (!matched) {

            continue

        }

        if(null == res) {

            res = {}

        }

        res[matched[1]] = comments[idx]

    }

    return res;

}

*function* commentsCommit(*reviewID, commentID*) {

*var* url = reviewURI+'api/review-requests/'+reviewID+'/reviews/'+commentID+'/'

    l.debug('post comment commit: ' + url)

*var* comment = {}

    comment.api\_format='json'

    comment.ship\_it=false

    comment.public=1

    request.put(url, reviewRequestOption).form(comment).on('data', *function*(d) {

*var* r = JSON.parse(d.toString())

        if (!r || 'ok' != r.stat) {

            l.error("fail to post comment commit " + JSON.stringify(comment) + " : " + d.toString())

            return

        }

        l.debug("success to post comment commit " + JSON.stringify(comment))

    })

}

*function* commentIDParse(*text*) {

*var* r = JSON.parse(text)

    if (!r || 'ok' != r.stat) {

        l.error("invalid response when parse commentID: " + text)

        return -1

    }

    return r.review.id

}

*function* diffFilesParse(*url, r*) {

    if(null == r || 0 == r.total\_results) {

        l.error('<'+url+'> invalid total results: ' + (r?r.total\_results:-1))

        return null

    }

*var* files = []

    for (*var* idx in r.files) {

*var* name = path.basename(r.files[idx].dest\_file)

*var* ext = path.extname(name)

        if(!ext || (ext != '.pm' && ext != '.pl')) {

            l.debug('<'+url+'> not check: ' + name)

            continue

        }

*var* fileID = r.files[idx].dest\_file

*var* newInsert = false

        if(0 != r.files[idx].extra\_data.total\_line\_count

            && (0 == r.files[idx].extra\_data.equal\_count

                || (r.files[idx].extra\_data.equal\_count / r.files[idx].extra\_data.total\_line\_count < 0.2))) {

            newInsert = true

        }

        /\*if('/src/app/vtp-common/StartOrder.pm' == fileID) {

            l.debug("set start order to true")

            newInsert = true

        }\*/

        files.push({'fileDiffID':r.files[idx].id, 'name': name, 'fileID': fileID, 'newInsert': newInsert, 'patchedFileURL': r.files[idx].links.patched\_file.href})

    }

    return 0 == files.length ? null : files

}

*function* responseBodyParse(*text*) {

*var* r

    try {

        r = JSON.parse(text);

    } catch(e) {

        //not a json object, just common text

        return text

    }

    //if codereview return a json, filter invalid stat

    if(r.stat && 'ok' != r.stat) {

        l.error('<'+url+'> invalid response: ' + text)

        return null

    }

    return r

}

*function* responseParse(*url, error, response*) {

    //error occur

    if(error) {

        l.error('<'+url+'> error: ' + error)

        return null

    }

    //status code not right

    if(response.statusCode < 200 || response.statusCode >= 300) {

        l.debug('<'+url+'> invalid status: ' + response.statusCode + ' body: ' + response.body)

        return null

    }

    return responseBodyParse(response.body)

}

*function* diffStat(*reviewID, diffIdx, cb*) {

*var* url = reviewURI+'api/review-requests/'+reviewID+'/diffs/'+diffIdx + '/'

    request.get({url:url, auth:reviewRequestOption.auth}, *function*(error, response){

*var* r = responseParse(url, error, response)

        return cb(0, r ? {'modifyTime': r.diff.timestamp, 'filesURL': r.diff.links.files.href} : null)

    })

}

*function* diffPatchedFiles(*url, cb*) {

    request.get({url:url, auth:reviewRequestOption.auth}, *function*(error, response){

        return cb(0, diffFilesParse(url, responseParse(url, error, response)))

    })

}

*function* syntexCheck(*path, check*) {

*var* checkCmd = null

    if ('log' == check) {

        checkCmd = 'perlcritic -s CodeLayout::RequireLogStatment ' + path

    } else if ('l4' == check) {

        checkCmd = 'perlcritic -4 --verbose 8 ' + path

    } else if ('all' == check) {

        checkCmd = 'perlcritic --verbose 8 ' + path

    } else {

        checkCmd = check + path

    }

    l.debug(checkCmd)

*var* checkResp

    try {

        checkResp = p.execSync(checkCmd).toString()

    } catch(e) {

        checkResp = e.stdout.toString()

    }

    fs.unlinkSync(path)

    return commentsParse(checkResp)

}

*function* writeContent(*content, filePath*) {

    mkdirsSync(path.dirname(filePath))

    fs.writeFileSync(filePath, content)

}

*function* commentPost(*url, fileDiffID, line, text, cb*) {

    // post file's ERROR check information to the server

    // @see https://www.reviewboard.org/docs/manual/dev/webapi/2.0/resources/review-diff-comment-list/

*var* comment = {}

    comment.api\_format   = 'json'

    comment.num\_lines    = 1

    comment.filediff\_id  = fileDiffID

    comment.issue\_opened = true

    comment.text\_type    = 'plain'

    comment.first\_line   = line

    comment.text         = text

    request.post({url:url, form: comment, auth:reviewRequestOption.auth}, *function*(error, response){

*var* r = responseParse(url, error, response)

        if(r) {

            l.info('comment success: ' + text)

        } else {

            l.error('comment failed: ' + text)

        }

        cb(0, r)

    })

}

*function* diffPatchedFileCheck(*diff, check, cb*) {

    request.get({url:diff.patchedFileURL, auth:reviewRequestOption.auth}, *function*(error, response){

        //download file

*var* fileContent = responseParse(diff.patchedFileURL, error, response)

        if(null == fileContent) {

            l.error('<'+diff.patchedFileURL+'> download file content failed')

            return cb(0, null)

        }

        //write download file to filesystem

*var* patchedFileDir = outDir + '/' + diff.fileDiffID

*var* patchedFilePath = patchedFileDir + '/' + diff.name

        writeContent(fileContent, patchedFilePath)

        l.debug('<'+diff.patchedFileURL+'> download to "' + patchedFilePath + '"')

*var* comments = syntexCheck(patchedFilePath, check)

        //clean download resource

        fs.rmdirSync(patchedFileDir)

        return cb(0, comments)

    })

}

*function* culminate(*lastStat*) {

*var* desc = ''

    for (*var* fileID in lastStat) {

        if(lastStat[fileID].IsNewInsert && lastStat[fileID].ErrorCount > 0) {

            console.error('@' + fileID + ' ' + lastStat[fileID].ErrorCount + ' errors need check')

            process.exitCode = 2

        } else {

            console.log('@' + fileID + ' ' + lastStat[fileID].ErrorCount + ' errors')

        }

    }

}

*function* workLog(*type, s*) {

*var* split = ' '

*var* record = null

    if ('diff' == type) {

        if(s.PostErrorCount > 0) {

            record = s.CheckTime + split + s.DiffTime + split + s.ReviewID + split + s.DiffIndex + split + s.FileDiffID + split + (s.IsNewInsert?1:0) + split + s.ErrorCount + split + s.PostErrorCount + split + s.Name + '\n'

        }

        //l.debug('CheckTime DiffTime ReviewID DiffIndex FileDiffID IsNewInsert ErrorCount PostErrorCount Name')

    }

    l.debug(JSON.stringify(s))

    if(record && sessionDir) {

        fs.appendFileSync(sessionDir+'/'+type, record)

    }

}

*function* onDeleteComments(*reviewID*) {

    suspend(*function*\*() {

*var* url = reviewURI+'api/review-requests/'+reviewID+'/reviews/'

*var* postsResponse = yield request.get(url+'?start=0&max-results=500', reviewRequestOption, resume())

*var* posts = responseBodyParse(postsResponse.body)

*var* regex = /at line ([0-9]+),/

        l.debug('review ' + reviewID + ' already has ' + posts.total\_results + ' comment posts, need to iterator delete')

        for (*var* idxPost = 0; idxPost < posts.reviews.length; ++idxPost) {

*var* urlc = posts.reviews[idxPost].links.diff\_comments.href

*var* commentsResponse = yield request.get(urlc+'?start=0&max-results=5000', reviewRequestOption, resume())

*var* rc = responseBodyParse(commentsResponse.body)

            for (*var* idxComment = 0; idxComment < rc.diff\_comments.length; ++idxComment) {

*var* comment=rc.diff\_comments[idxComment]

                if(comment.text.match(regex)) {

*var* commentURL=comment.links.delete.href

                    l.debug('<' + commentURL + '>' + ' delete ' + comment.text)

                    request.delete({url:commentURL, auth:reviewRequestOption.auth}, *function*(error, response){

                        l.debug(' delete ' + response.body + (error?("failed: " + error): ""))

                    })

                }

            }

        }

    })()

}

*function* commentsFill(*url, comments, cb*) {

    request.get({url:url, auth:reviewRequestOption.auth}, *function*(error, response){

*var* r = responseParse(url, error, response)

        if(null == r) {

            l.error('<'+url+'> comments parse failed')

            return cb(0, null)

        }

        l.debug('<'+url+'> ' + r.diff\_comments.length + ' comments')

        for (*var* idxComment = 0; idxComment < r.diff\_comments.length; ++idxComment) {

*var* fileDiffID = extractLastURLToken(r.diff\_comments[idxComment].links.filediff.href)

            if(fileDiffID) {

                comments[fileDiffID + '.' + r.diff\_comments[idxComment].first\_line] = r.diff\_comments[idxComment].text

            }

        }

        return cb(0, r.links.next?r.links.next.href:null)

    })

}

*function* onRetrive(*reviewID, check*) {

    suspend(*function*\*() {

*var* url = reviewURI+'api/review-requests/'+reviewID+'/reviews/'

*var* lastStat = {}

*var* commentID = null

*var* comments = null

        // iterator all diffs

*var* diffIdx = 1

        for (; diffIdx < 20; ++diffIdx) {

*var* stat = yield diffStat(reviewID, diffIdx, resume())

            if(null == stat) {

                l.debug('diff ' + reviewID + '/' + diffIdx + ' stat not ok, reach the end')

                break

            }

*var* patchedFiles = yield diffPatchedFiles(stat.filesURL, resume())

            if(null == patchedFiles) {

                l.debug('<'+stat.filesURL+'> no patched files should be check')

                continue

            }

            for (*var* idx in patchedFiles) {

*var* patched = patchedFiles[idx]

*var* eCount = 0

*var* posteCount = 0

*var* lines = yield diffPatchedFileCheck(patched, check, resume())

                for(*var* line in lines) {

                    //lazy find old comments

                    if(null == comments) {

                        comments = {}

*var* postsResponse = yield request.get(url+'?start=0&max-results=500', reviewRequestOption, resume())

*var* posts = responseBodyParse(postsResponse.body)

                        l.debug('review ' + reviewID + ' already has ' + posts.total\_results + ' comment posts')

                        for (*var* idxPost = 0; idxPost < posts.reviews.length; ++idxPost) {

*var* nextURL = posts.reviews[idxPost].links.diff\_comments.href+'?start=0&max-results=5000'

                            while(nextURL) {

                                nextURL = yield commentsFill(nextURL, comments, resume())

                            }

                        }

                        //l.debug(JSON.stringify(comments))

                    }

*var* commentIdx = patched.fileDiffID+'.'+line

                    if(comments[commentIdx]) {

                        l.debug('already exists comments ' + commentIdx + ': ' + comments[commentIdx]

                            + ((lines[line] == comments[commentIdx])?'':(' => ' + lines[line])))

                    } else {

                        //lazy request commentID

                        if(null == commentID) {

*var* data = yield request.post(url, reviewRequestOption, resume())

                            commentID = commentIDParse(data.body)

                        }

                        yield commentPost(url+commentID+'/diff-comments/', patched.fileDiffID, line, lines[line], resume())

                        ++posteCount

                    }

                    ++eCount

                }

                lastStat[patched.fileID] = {

                    'ReviewID': reviewID,

                    'DiffIndex': diffIdx,

                    'FileDiffID': patched.fileDiffID,

                    'IsNewInsert': ((lastStat[patched.fileID]?lastStat[patched.fileID].IsNewInsert:false) || patched.newInsert),

                    'ErrorCount': eCount,

                    'PostErrorCount': posteCount,

                    'DiffTime': moment(stat.modifyTime).local().format('YYYY-MM-DD HH:mm:ss'),

                    'CheckTime': moment().format('YYYY-MM-DD HH:mm:ss'),

                    'Name': patched.fileID,

                }

                workLog('diff', lastStat[patched.fileID])

            }

        }

        if(null != commentID) {

            commentsCommit(reviewID, commentID)

        }

        workLog('review', lastStat)

        culminate(lastStat)

    })()

}

*function* main() {

*var* param = require('commander')

    param

        .version('2016.08.06')

        .description('Review Board manage tool')

        .option('-s, --rserver <addr>', 'Review Board server address, default is [127.0.0.1]', '127.0.0.1')

        .option('-u, --ruser <username>', 'Review Board server login username, default is [root]', 'root')

        .option('-p, --rpass <password>', 'Review Board server login password, default is []', '')

        .option('--flog', 'log to file')

        .option('-r, --reviewID <id>', '@retrive Review ID')

        .option('--check <cmd>', '@retrive check command, can be "log" "l4" "all" "\$userdef", default is [log]', 'log')

        .option('--session [dir]', '@retrive session record dir, default no session and session dir is [/etc/rb]', '/etc/rb')

    param

        .command('retrive [env]')

        .description('Retrive all original/patched files special by reviewid')

        .action(*function*(env, options){

            if(!optParse(options.parent)) {

                console.log("opt parse failed")

                process.exit(1)

            }

            if(!options.parent.reviewID) {

                console.log("reviewid is required")

                process.exit(1)

            }

            if(strArrBeginWith(options.parent.rawArgs, '--session')) {

                sessionDir = options.parent.session

                mkdirsSync(sessionDir)

            }

*var* check=options.parent.check

            onRetrive(parseInt(options.parent.reviewID), check)

        })

    param

        .command('delcomment [env]')

        .description('Delete all comments submited by this tool')

        .action(*function*(env, options){

            if(!optParse(options.parent)) {

                console.log("opt parse failed")

                process.exit(1)

            }

            if(!options.parent.reviewID) {

                console.log("reviewid is required")

                process.exit(1)

            }

            onDeleteComments(parseInt(options.parent.reviewID))

        })

    param.parse(process.argv)

}

main()

此rbjob工具支持如下参数：

# ./rbjob --help

Usage: rbjob [options] [command]

Commands:

retrive [env] Retrive all original/patched files special by reviewid

delcomment [env] Delete all comments submited by this tool

Review Board manage tool

Options:

-h, --help output usage information

-V, --version output the version number

-s, --rserver <addr> Review Board server address, default is [127.0.0.1]

-u, --ruser <username> Review Board server login username, default is [root]

-p, --rpass <password> Review Board server login password, default is []

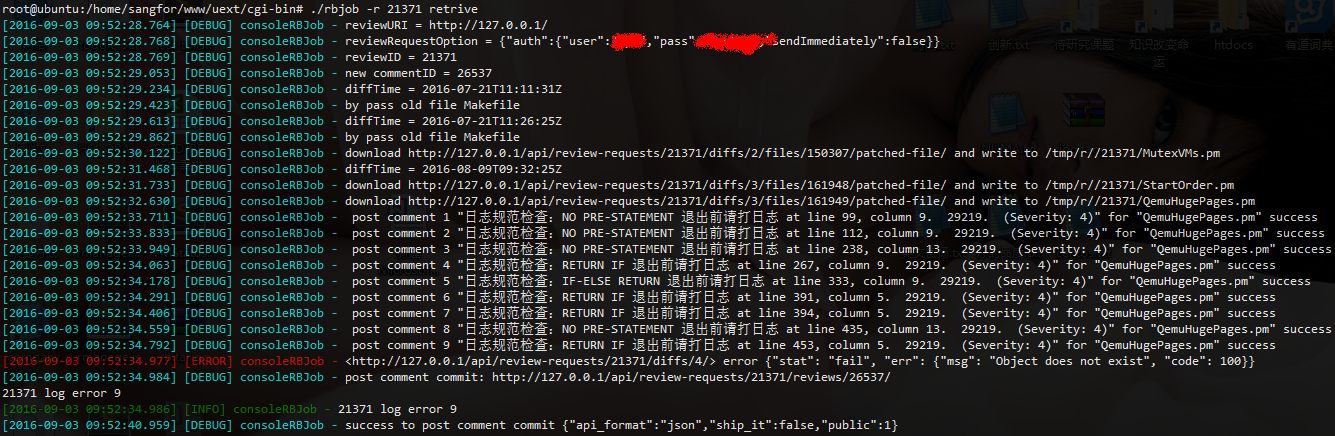
--flog log to file

-r, --reviewID <id> @retrive Review ID

--check <cmd> @retrive check command, can be "log" "l4" "all" "$userdef", default is [log]

--session [dir] @retrive session record dir, default no session and session dir is [/etc/rb]

rbjob运行效果：



剥去日志，输出类似：

@/src/app/vtp-common/MutexVMs.pm 0 errors

@/src/app/vtp-common/StartOrder.pm 1 errors

@/src/app/vtp-qemu-server/VTP/QemuHugePages.pm 9 errors

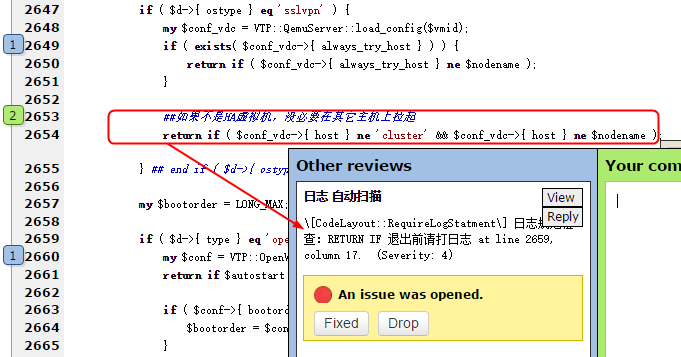
@/Test.pm 80 errors need check

检测到错误会打印xx errors，如果是xx errors need check表示检测到错误且是一个新增文件或者改动超过80%的文件，之所以如此，是因为遗留代码不宜强制程序员修改，而新增代码强制要求符合规则比较容易推动。

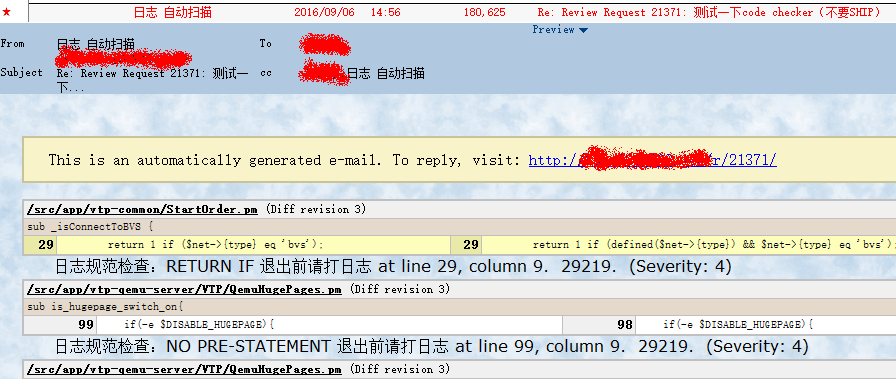
上述具体的错误信息自动提交到reviewboard上，comments列表效果如：



具体错误信息如：



以上错误信息还会发送到内部邮件：



## 自动扫描

rbjob实现了根据reviewID自动下载代码，然后运行工具检测告警，然后将告警自动提交成review comments，但是需要人工来调用rbjob才能执行，如何做到自动在后台监控？

找了一下reviewboard没有提供最进改动review列表API，但是从页面上看有此功能，于是用curl定时将最近改动列表抓下来，然后自动调用rbjob，此脚本取名rbeye:

#!/bin/bash

# monitor code submit and auto check

#     su -c "./rbeye >>/var/log/rb/eye.$(date +%F).log 2>&1" www-data

# @author mnstory.net

# @version 20160910 - auto check by page list

if [ -f "$0" ]; then

    SCRIPTDIR=$(dirname $(realpath $0 2>/dev/null) 2>/dev/null)

fi

if [ -z "$SCRIPTDIR" ]; then

    SCRIPTDIR=.

fi

rb\_username="root"

rb\_password=""

rb\_baseurl="http://127.0.0.1"

rb\_tmpdir="/tmp/rb/"

ldebug()

{

    echo -e $(date +"%F %T") "$$ $@" 1>&2

}

coolywork()

{

    local timeafter=$1

    local cookies="${rb\_tmpdir}cookies.txt"

    local home="${rb\_tmpdir}home.txt"

    local rid\_records="${rb\_tmpdir}rid\_records.txt"

    local curlcmd="curl -v -c $cookies -b $cookies"

    mkdir -p "$rb\_tmpdir" 2>/dev/null

    rm -f "$cookies" 2>/dev/null

    $curlcmd $rb\_baseurl/account/login/ >/dev/null 2>&1

    local csrf=$(cat $cookies | grep csrftoken | awk '{print $7}')

    $curlcmd -d "username=${rb\_username}&password=${rb\_password}&csrfmiddlewaretoken=${csrf}" $rb\_baseurl/account/login/ >/dev/null 2>&1

    $curlcmd -H 'Pragma:no-cache; Cache-Control:no-cache' $rb\_baseurl/r/ 2>/dev/null > "$home"

    cat "$home" | grep -oP '<a href="/r/\d+/">.\*</div>|datetime=".\*"' | awk -F/ '{id=$3; getline T; t=substr(T,11,22); print id,t}' > "$rid\_records"

    while read line; do

        local rid=${line% \*}

        local uptime=${line#\* }

        local uptime\_sec=$(date --date="$uptime" +%s)

        if [[ -n "$timeafter" && $uptime\_sec -lt $timeafter ]]; then

            #ldebug "by pass $rid $uptime, $uptime\_sec < $timeafter"

            continue

        fi

        local rbjobcmd="$SCRIPTDIR/rbjob --flog -r $rid retrive --session --check l4"

        ldebug "$rbjobcmd (at $uptime, start)"

        $rbjobcmd

        lasterr=$?

        ldebug "$rbjobcmd (exit $lasterr)"

    done < "$rid\_records"

    return 0

}

main()

{

    if [ "?" = "$1" ]; then

        echo "$0 [timeafter]"

        return 1

    fi

    local timeafter=""

    if [ -n "$1" ]; then

        timeafter=$(date --date="$timeafter" +%s)

    #else

    #    timeafter=$(date +%s)

    fi

    while : ; do

        local timecheckpoint=$(date +%s)

        coolywork $timeafter

        sleep 15

        timeafter=$timecheckpoint

    done

    return 0

}

main "$@"

运行rbeye，由于rbjob我们还会在cgi里面调用，cgi owner是www-data，www-data读写文件权限受限较多，需要特别注意不同用户操作导致权限不够情况。SSH链接进程会在SSH CLIENT退出时候SIG HUB给子进程，需要考虑，例如：

nohup command ...

screen -dmS newscreen; screen -list; screen -r newscreen; command ...

pkill rbeye; pkill rbjob; ps auxf | grep -P 'rbeye|rbjob'

rm -rf /var/log/rb /tmp/rb; mkdir -p /var/log/rb /etc/rb /tmp/rb; chmod 777 /var/log/rb /etc/rb /tmp/rb; chown www-data -R /var/log/rb /etc/rb /tmp/rb

nohup su -c "./rbeye >>/var/log/rb/eye.$(date +%F).log 2>&1" www-data &

tail -f ../log/review-debug.log /var/log/rb/\*.log

## SVN PRE-COMMIT

为了在SVN上库流程卡主代码，需要对svn提交做HOOK，将svn pre-commit通过网络请求转发到自动检测代码，这段流程已经有，在已有的reviewboard HOOK脚本处添加检测调用：

import subprocess

syntexcmd = "rbjob -r "+*str*(rid)+" retrive --check log --flog 2>&1"

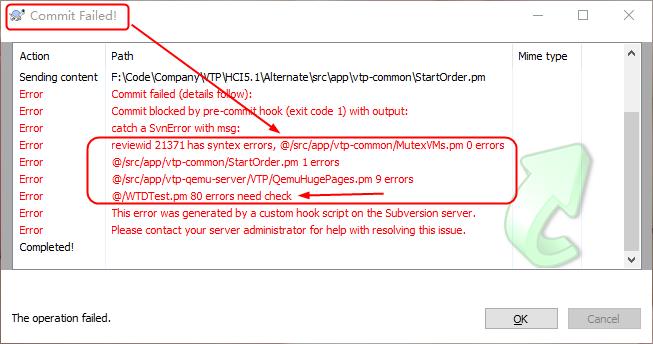
checkResult = subprocess.Popen(syntexcmd, *shell*=True, *stdout*=subprocess.PIPE).stdout.read()

if "errors need check" in checkResult:

    raise SvnError, "reviewid %s has syntex errors, %s" % (rid, checkResult)

debug(syntexcmd + " => " + checkResult)

这里检测是否有错是检测输出里面是否有"errors need check"，且检测标准和rbeye里面的有所不同，这里检测的是log，rbeye里面检测的是l4，如前面所述，只对判定为新增文件的PERL代码做强制要求，不符合规定就无法上库，所以，这里控制了报错，只要求新增代码做日志规范。阻止上库效果：



此时，你需要打开reviewboard查看并处理相应的错误，然后重新提交改动后的代码到同一review,如果没有错误了，下次提交会顺利通过。

2016/9/12