# PERL单元测试简易框架

[mnstory.net](http://mnstory.net)

本来是写个测试脚本来测试函数是否正确，写着写着就把输出写的漂亮了些，于是将代码抽出来，做成一个公共框架。

说是框架，非常王婆卖瓜，因为就一百五十行SHELL脚本，PERL本是有单元测试的，但是我认为几行代码用那单元测试有点小题大做，最关键是，我的需求非常少，就是要跑几句代码，检测一下返回值，并且匹配一下预期数据，然后列出测试结果而已。

我之前喜欢这样简单测试PERL函数代码：

perl -T -w -e 'use VTP::MutexVMs; use Data::Dumper; print Dumper(VTP::MutexVMs::mutexVMIDs())'

这样跑一遍，或者加-d调试一下，函数是否有问题，一下就能看出来，唯一的问题是，测试多组输入参数和输出的时候，每次都要靠双眼来对比结果是否符合预期。

有了此简易框架，两个函数的四条完整单元测试案例，你只需要类似这样写：

#!/bin/bash

. unitest.sh

#自定参数，随意！

MUTEXPATH='/cfs/mutexvms.xml'

mutexVMIDs()

{

    #设置一个单元测试案例名字

    target="$FUNCNAME.readNormal"

    #准备输入参数，例如：准备文件

    \cp -f "$SCRIPTDIR/in/$target" $MUTEXPATH

    #开始执行PERL代码，重定向输出到 "$SCRIPTDIR/out/$target"

    perl -T -w -e 'use VTP::MutexVMs; use Data::Dumper; print Dumper(VTP::MutexVMs::mutexVMIDs())' > "$SCRIPTDIR/out/$target"

    #对比结果，如果不比较返回值，只比较输入输出，用alwaysSuccess，其他的还有 zeroSuccess equalSuccess nonZeroSuccess

    alwaysSuccess "$target" "read vmids normal"

}

addMutexGroup()

{

    target="$FUNCNAME.emptyAdd"

    rm -f $MUTEXPATH 2>/dev/null

    errno=`perl -T -w -e 'use VTP::MutexVMs; print VTP::MutexVMs::addMutexGroup("6105658195070", "6460910776608")'`

    \cp -f $MUTEXPATH "$SCRIPTDIR/out/$target"

    #对比返回值，如果返回值为0表示成功，并且，会对比输出

    zeroSuccess "$errno" "$target" "remove $MUTEXPATH first, then add"

    target="$FUNCNAME.emptyArg"

    errno=`perl -T -w -e 'use VTP::MutexVMs; print VTP::MutexVMs::addMutexGroup()'`

    \cp -f $MUTEXPATH "$SCRIPTDIR/out/$target"

    equalSuccess "-1" "$errno" "$target" "empty argments"

    target="$FUNCNAME.invalidArg"

    errno=`perl -T -w -e 'use VTP::MutexVMs; print VTP::MutexVMs::addMutexGroup(undef, undef, undef)'`

    \cp -f $MUTEXPATH "$SCRIPTDIR/out/$target"

    nonZeroSuccess "$errno" "$target" "invalid argments"

}

entry()

{

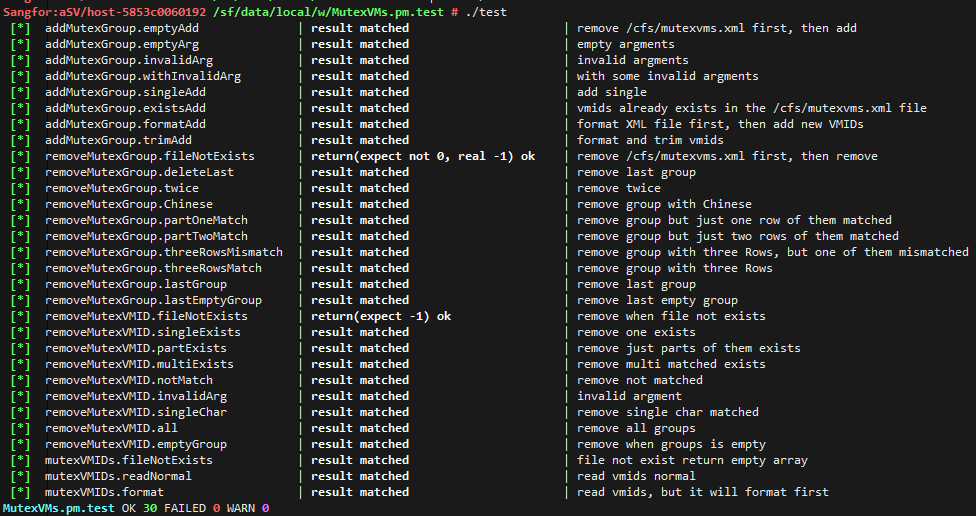
    mutexVMIDs

    addMutexGroup

}

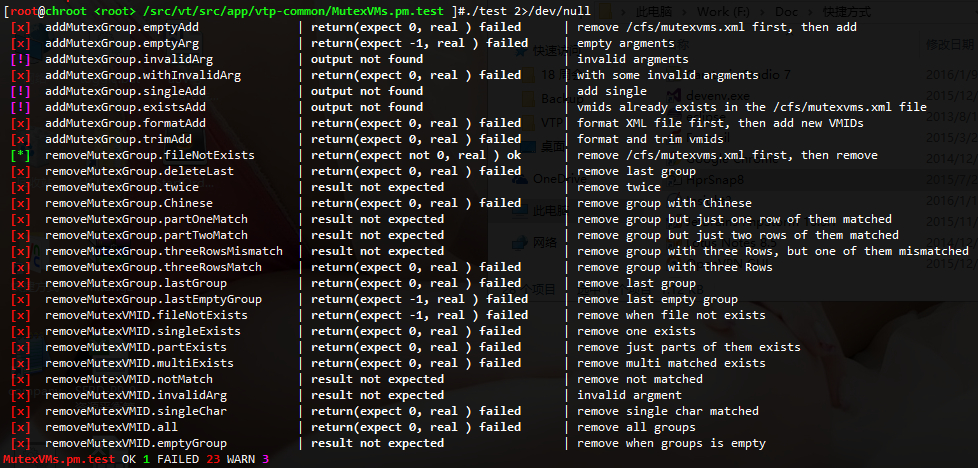
main "$@"

然后在脚本目录下的expected目录准备对应测试案例名称的输出以便对比结果。执行输出类似：



这是全成功的输出，最后一行可以看出，30个案例，全部成功。

也有失败的效果：



只有一个成功，23个失败，3个WARN表示内部错误（找不到输出文件）。

现在看一下所谓的框架unitest.sh：

#!/bin/bash

# common function for unit test

# @author mnstory.net

# @version 1.0 - 20160723

shopt -s expand\_aliases

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

SCRIPT\_NAME=$(basename $(realpath $0 2>/dev/null) 2>/dev/null)

STATUS\_OK="\033[01;32m [\*] \033[00m"

STATUS\_FAILED="\033[01;31m [x] \033[00m"

STATUS\_WARN="\033[01;35m [!] \033[00m"

LOGLEVEL=0

status\_ok\_cnt=0

status\_failed\_cnt=0

status\_warn\_cnt=0

status\_info()

{

    local module="$SCRIPT\_NAME"

    if [ "$SCRIPT\_NAME" == "test" ]; then

        module=$(basename "$SCRIPTDIR")

    fi

    local lasterr=0

    if [ 0 -ne $status\_failed\_cnt -o 0 -ne $status\_warn\_cnt ]; then

        echo -en "\033[5m\033[1m\033[01;31m$module\033[00m"

        lasterr=2

    else

        echo -en "\033[01;36m\033[1m$module\033[00m"

    fi

    echo -e " OK \033[01;32m$status\_ok\_cnt\033[00m FAILED \033[01;31m$status\_failed\_cnt\033[00m WARN \033[01;35m$status\_warn\_cnt\033[00m"

    return $lasterr

}

compareSuccess()

{

    local ret\_expected="$1"

    local ret\_compare="$2"

    local ret\_real="$3"

    local target\_name="$4"

    shift; shift; shift; shift

    local msg="$@"

    if [ -n "$msg" ]; then

        msg="| $msg"

    fi

    local statusmsg=""

    local checkmsg=""

    local ret\_success=""

    if [ "eq" == "$ret\_compare" ]; then

        if [ "$ret\_expected" == "$ret\_real" ]; then

            ret\_success="true"

            statusmsg=$STATUS\_OK

            checkmsg="return(expect $ret\_expected) ok"

        else

            statusmsg=$STATUS\_FAILED

            checkmsg="return(expect $ret\_expected, real $ret\_real) failed"

        fi

    else

        if [ "$ret\_expected" != "$ret\_real" ]; then

            ret\_success="true"

            statusmsg=$STATUS\_OK

            checkmsg="return(expect not $ret\_expected, real $ret\_real) ok"

        else

            statusmsg=$STATUS\_FAILED

            checkmsg="return(expect not $ret\_expected) failed"

        fi

    fi

    if [ -n "$ret\_success" ]; then

        if [ -f "expected/$target\_name" ]; then

            if [ -f "out/$target\_name" ]; then

                if ! diff "expected/$target\_name" "out/$target\_name" >/dev/null 2>&1 ; then

                    checkmsg="result not expected"

                    statusmsg=$STATUS\_FAILED

                    if [ $LOGLEVEL -ge 1 ]; then

                        diff "expected/$target\_name" "out/$target\_name" 1>&2

                    fi

                else

                    checkmsg="result matched"

                fi

            else

                checkmsg="output not found"

                statusmsg=$STATUS\_WARN

            fi

        fi

    fi

    if [ "$statusmsg" == "$STATUS\_FAILED" ]; then

        status\_failed\_cnt=$(( status\_failed\_cnt + 1 ))

    elif [ "$statusmsg" == "$STATUS\_OK" ]; then

        status\_ok\_cnt=$(( status\_ok\_cnt + 1 ))

    elif [ "$statusmsg" == "$STATUS\_WARN" ]; then

        status\_warn\_cnt=$(( status\_warn\_cnt + 1 ))

    fi

    printf "${statusmsg} %-35s | " "$target\_name"

    checkmsg=`printf "%-35s" "$checkmsg"`

    echo -e "\033[1m${checkmsg}\033[00m ${msg}"

}

equalSuccess()

{

    local ret\_expected="$1"

    shift

    compareSuccess "$ret\_expected" "eq" "$@"

}

notEqualSuccess()

{

    local ret\_expected="$1"

    shift

    compareSuccess "$ret\_expected" "ne" "$@"

}

alias zeroSuccess='equalSuccess 0'

alias nonZeroSuccess='notEqualSuccess 0'

alias alwaysSuccess='equalSuccess "<empty>" "<empty>"'

main()

{

    if [ "?" == "$1" -o "-h" == "$1" -o "--help" == "$1" ]; then

        echo "$0, unit test for perl module"

        return 1

    fi

    while [ "" != "$1" ]; do

        case "$1" in

            -v\* )

                if [ "$1" == "-v" ]; then

                    LOGLEVEL=1

                elif [[ "$1" == "-vv" ]]; then

                    LOGLEVEL=2

                elif [[ "$1" == "-vvv" ]]; then

                    LOGLEVEL=3

                fi

                shift

                ;;

            \*)

                break

                ;;

        esac

    done

    entry "$@"

}

export -f main

童叟无欺，加了一些无相关的代码，整个简易框架就一百五十行代码。不论什么框架，殊途同归，单元测试过程中，发现如下意义：

1. 构造单元测试案例的同时，就是思考异常的过程，过程会发现一些没考虑的地方，例如我在测试过程发现了VMID项为中文调用写入函数出错；技术支持可能手动修改脚本，可能残留空格在VMID两侧，导致匹配失败。
2. 可复现性较好，例如，我在代码里面添加了对中文字符串的转换后，再运行一遍案例，是否对已有功能有影响一目了然，而不需要再次人工确认。
3. SHELL脚本做单元测试，相对用原生语言，例如PERL或C，更加灵活，且测试框架本身无外部依赖。

2016/7/26