Shell编写规范

前言

脚本对于开发和运维人员是很重要的，目前没什么统一的格式和管理方式，isd 运营部 itil 支持组、isd 运营部系统运维组，isd 运营部公共运维组以及 isddevwiki 门户同时有了这个需求，笔者有幸总结自己经验，撰写此文，当然是抛砖引玉，需要不断积累总结和完善，相关更新请查阅本文以后升级文档以及 ISDDevwiki 门户 shell 版 http://wiki.oa.com/ISDDevWiki

一、脚本分类：

目前在 isd 脚本主要分为 4 类：进程维护类，同步脚本，临时脚本，数据（日志）提取分析类

进程维护脚本：这类脚本特点是功能简单明确，附属某个服务进程，对这个进程进行启停监控操作；在 ISD 的打包规范中，这类脚本可以由打包工具自动生成。

同步脚本：主要解决不同机器的文件传送问题，在发布，定时同步的场景下大量用到；

数据提取分析脚本：主要通过对特定数据文件、db、日志等进行数据提取，然后经过相对复杂的逻辑得到明确输出结果。这类脚本一般包括多个脚本、配置文件，比较复杂、专用。

临时脚本：一般是运维或者开发为了方便某些临时操作，写好的一些批处理脚本命令，这类脚本生命周期一般不长。

二、脚本编写规范

脚本编写规范以可维护、可靠、安全为原则，这里针对的脚本对象是同步类脚本和数据提取分析类脚本。

可维护性：

物理部署：

在机器特定目录下部署公共库；/usr/local/commscript
公共库包含两个文件夹
bin:存放可执行脚本和脚本所调用工具了文件
conf:公共配置文件，公共库的 bin 所需要的配置文件；

在业务目录（/usr/local/services）下部署业务脚本
业务脚本分一个主目录和 3 个子目录部署文件：
主目录命名同核心脚本的文件名，三个子目录统一命名：
bin ： 部署可制定脚本文件
conf ： 部署配置文件，pid 文件等
log ： 部署日志文件（公共库产生的日志也写在这里面）

文件命名：

可执行脚本以.sh 作后缀
配置文件以.conf 或 .ini 作后缀；
日志文件已 log 作后缀；
数据文件以.dat 或.data 作后缀；
进程 pid 文件以.pid；
中间产生文件以.开头 .tmp 作后缀；
公共组建以.comm 作为后缀

可执行文件采用主谓或者动宾结构，表达意思明确 比如 ServerLogMonitor.sh；
日志建议按天产生，文件名中间最好能包括日期。比如 logserver.2007-11-23.log；
配置文件以名词来命名后接.conf 或 .ini 作后缀；

文件命名长度不要超过 15 个字符

公共库：

从一般脚本中抽象出公共通用的代码和函数，组成公共，部署在机器统一地方，统一管理和维护，公共库和应用脚本界限不是绝对的，可以不断提取和优化；

配置化：

可塑化信息都写在配置文件里面；运维人员不需要读.sh 文件就可以方便使用脚本。

文件开始：

#!/bin/bash
PATH=/usr/local/bin:/usr/bin:/bin:/sbin

现在公司 os 的/bin/sh 都是 /bin/bash 的一个符号链接；
这个 PATH 最小集合 可以根据需要添加，shell 会从做到右遍历 PATH，所以注意 PATH 的顺序，比如/usr/local/bin 要在/usr/bin 前面；
这样做有好处是：
第一行明确告诉系统本身是个 Shell 程序而非一般文本
Fock 新的进程，子进程的对 PATH 的定义不影响父进程（export 除外）

公共库 PATH 必须是
PATH=/usr/local/bin:/usr/bin:/bin:/sbin:$PATH
否则会引起库引入覆盖

绝对路径：

对脚本中使用倒的配置文件、日志文件读写都采用绝对路径，绝对路径由程序自动获取，如后拼凑出其它文件或目录的绝对路径，让脚本尽量收敛；

结构：

采用结构化程序结构；注意函数封装，主流简洁，程调用各个函数完成；

注释：

文件注释在写完开始两行写，写明文件主要功能，入口参数、结果等
以”# ”（#空格）作为行开始
如：
this is a test shell input is none
have fun
…

函数注释在函数名上面
以”### ”（###空格）作为行开始
this function is to get a record form db
fn_GetRecord ()
{
    …
}

函数体或主程序中间注释以”#”开始（后不接空格）与需要注释的片断缩进相同

函数变量的命名：

主要参照匈牙利命名法则，一个函数或者变量的命名分为三个域
作用域+属性域+描述域
1、作用域：
如果是全局函数或者变量名，前面加上 g_ 全局前缀，如果是在函数体或者主程序中间定义的临时变量，前面加上 t_ 前缀，否则这项为空；总长度不得超过 20 个字符
2、属性域：
a Array
b Boolean
fn Function
i Integer
s String
t Time
l long
3、描述域：
简单直接描述变量和函数需要表现的信息
变量采用名词或者形容词+名词形式，单词第一个字符大写
函数采用动词+名词方式，单词第一个字符大写
如果这个变量或者函数名属于公共库，此域所有单词大写
4、例子

```
#定义全局变量
g_i_ReadCount=0

#定义临时变量
t_s_TempFile=”/tmp/$$.tmp”

#定义全局函数
g_fn_GetCount()
{
	echo ${g_i_ReadCount}
	return 0
}
#定义全局函数，属于公共库
g_fn_GETVERSION()
{
	#...
	return 0
}
#定义全局变量，属于公共库
g_s_AUTHOR=”ISD”

#没有作用域的变量

for((i_Loop=0;i_Loop<100;i_Loop++))
do
#..
done
```

#没有作用域的函数
fn_GetLogData()
{
#...
Return 0
}

5、函数内部变量，增加 local 关键字；

6、初始化
如果是字符串，初始化为 “”，
如果是数字初始化为 0；
函数不允许必须有 renturn，

文件注释内容：

文件注释写在之后
#!/bin/bash
PATH=/usr/local/bin:/usr/bin:/bin:/sbin

必须包含的内容有：
文件作者，所属部门，创建时间，功能描述
参考以下：
Author: marshalliu
Dept: IOD
Create Time : 2007-12-18
Description: 本脚本封装一组发送 RTX 消息到制定用户的功能函数；

公共库函数注释：

公共库函数必须包括
1. 函数功能说明
2. 参数说明
3. 返回值说明
4. 调用实例
5. 函数紧跟在后面（不需要空行），用中文描述

比如：
Desc : 发送 RTX 消息到制定用户
param: 接收 3 个入口参数，第一个为用户列表（不能为空，多用户用逗号隔开）
第二个参数是发送信息（不能为空）
第三个参数是延迟发送的分钟数（空为立即发送）
returns: 0 发送成功， 1 用户参数错误，2 时间参数错误，3 RTX 服务器错误
Usage : 以下向 marshalliu,peterxu,robertluo 在 180 分钟后发送信息 hello
g_fn_ RTX2USERS “marshalliu,peterxu,robertluo” “hello” 180
if [$? –eq 0]

```
###         then
###             #......
###             #发送成功
###         fi
g_fn_RTX2USERS()
{
    #...
}
```

空行缩进和空格：

两段有明显语义区别的脚本信息中必须空行

1. 文件注释后
2. 函数和函数之间
3. 函数和正文之间
4. 重要语句结束（比如循环体，”.” 表达式 case）

在分支（if+else case）循环（for while）等结构体内的正文必须缩进，缩进的长度为 4 个空格

同一级的 if 和 else elif fi 列对齐（then 可以和 if 同行，换行必须与他们对齐）
同一级 for do done 列对齐
同一级 while do done 列对齐
等

空格必须满足 shell 语法，任何情况下（特定字符串除外）不宜两个空格同时写，如果一行包含多条语句，”;”号后加一个空格

防止公共库重复加载：

由于应用脚本和公共库可能都会调某些公共组件，或者公共脚本间可能出现交叉引用，所以公共组件的全局变量或者函数定义必须保证不被重复加载；

凡是公共组建，在文件注释后，必须加上一个宏保证保证加载唯一性

```
#!/bin/bash
PATH=/usr/local/bin:/usr/bin:/bin:/sbin:$PATH

# 文件注释…
# 文件注释…
# 文件注释…

if [ -z "${__DEFINE_DATEPROC_COMM__}" ]
then
    __DEFINE_DATEPROC_COMM__="DEFINED"
```

全局变量定义
函数注释
函数定义

fi #end define

文件结束
这个宏__DEFINE_DATEPROC_COMM__命名的形式是 双下滑线开始+DEFINE+下滑线+文件名全大写_文件后缀全大写+双下滑线结尾；

返回和退出：

函数执行结果如果对整个程序执行由影响，要有明确返回值，用 return value 否则返回函数最后一行执行结果；
主程序执行过程中的每个出口必须告诉系统你的执行情况 exit value 方式，value 为 0，则表示脚本执行成功，否则失败，如果没有 exit 那么将返回系统作后一行执行结果。

版本：

每个脚本需要处理-V 参数，让外部用户查看当前脚本版本

```
if [ "$1" = "-V" ]; then
        echo -e "\nCreate Time:2007-11-13,Author:isd,V1.0\n"
        exit 0
fi
```

公共组件，和应用脚本都需要版本概念，同时处理这个参数；

每次升级脚本同时修改版本号和增加描述信息：

```
if [ "$1" = "-V" ]; then
        echo -e "\nCreate Time:2007-11-13,Author:isd,V1.0\n"
        echo -e "\nModified Time:2007-11-27,V1.1,add server log \n "
        exit 0
fi
```

日志：

在脚本全局变量定义日志名称

```
g_s_LOGDATE=`date +"%F"`
g_s_LOGFILE="../log/logserver.${g_s_LOGDATE}.log"
%%
            增加写日志函数
%%
### LOG to file
g_fn_LOG()
{
        s_Ddate=`date +"%F %H:%M:%S"`
```

```
        echo "[$s_Ddate]""$*" >>$g_s_LOGFILE
}
```

主程序或其他函数 调用 LOG $msg 记录日志 [YYYY-mm-DD HH:MM:SS] $msg
日志大小和内容要严格控制：运行频率在分钟以下级别，只记录重要日志和错误日志，频率在小时以上或手工执行的可记录过程日志；

可靠性：

调试：

-x 可以调试，不过有点混乱，建议增加调试函数

```
G_IFDEBUG=$1
### debuging variables
g_fn_DEBUG()
{
      if [ "$G_IFDEBUG" = "-D" ]; then
            s_Ddate=`date +"%F %H:%M:%S"`
            echo "[$s_Ddate]""$*"
      fi
      return 0
}

g_fn_DEBUG $MSG
```

如果程序执行时候加上-D 那么 DEBUG 则会打印信息，如果没有-D ，那么此行将略过

公共库的引入

公共库的引入要注意次序，shell 是解释执行语言，变量和函数可以重复定义，规则是顺序解释，后定义覆盖以前定义的，所以在引入公共库时候一般在文件开始
引入语法统一使用
. path/file 形式 或者 source path/file

系统调用时用通用参数

有些系统命令版本不同，有些命令提供了简化参数，但是写脚本特别是公共库应该避免；
举个例子：
用 head 或者 tail 看制定行数,比如看 test.log 前 100 行
我们尽量使用 head –n 100 `test.log` 或者 head –n100 `test.log`
而不要使用简化版参数 head -100 `test.log`
某些 head 的版本是不支持这种写法的。

字符串判断

对任何字符串做逻辑判断都要加””把字符产包起来，
if [-z “$s_Msg”]
if [“$s_Msg” = “hello”]
防止脚本异常
文件目录判断也是如此。

执行结果判读

对于系统调用执行后需要判断执行结果。系统命令一般结果正常统一返回；
用
if [$? –eq 0]
判断，这些命令典型有 ls rm mv 等 这些命令经常由于权限问题出错，如果随意对待，脚本会异常退出，如果处理不当，还会引起死循环等；

异常日志

对于一些严重影响到脚本执行的错误（这类错误一般是系统命令执行失败引起），需要把系统产生的错误日志导入本地日志
比如使用
g_ErorLog="../errorlog.log"
rm –f /root/test.sh 2 >> $g_ErorLog

安全性：

收敛

脚本尽量收敛于本身；这里的收敛包括
1. 对系统其他环境的依赖
2. 产生的临时文件限于本目录下，并在时候后马上清除；

```
fn_CleanTmp()
{
	if [ -w    "$s_TmpHost" ]; then
		rm -f $s_TmpHost
			fi
	if [ -w    "$s_TmpFile" ]; then
		rm -f $s_TmpFile
	fi
	return 0
```

}
比较好的办法是些个函数罗列所有临时文件，程序出口调用它。
对临时文件产生可以参考 tempfile

使用 trap

trap 信号量，保证你脚本安全，保证程序执行时候不被人为 kill，或者在被 kill 时候保存记录

```
fn_ForceExit()
{
    if [ ! -z "$g_LogFile" ]; then
        g_fn_LOG   "脚本被强行退出"
    fi
        fn_CleanTmp
    exit 1
}

trap "fn_ForceExit" 2 9
```

这里 trap 了 kill -9 和 kill -2 （ctrl +c）两个信号量

后台执行：

通常我们会把脚本写在 crontab 中，让系统调度，crond 一般顺序执行命令，如果脚本阻塞住了，那么对 crontab 后面的程序产生直接影响，所以写在 crontab 里的定式脚本一定要后台执行

普通用户执行：

脚本不到不得已不用超级用户执行；

一致性检测

为了防止脚本被有意或者无意更改，ISD 在部分脚本发布时候增加了 md5 效验功能；并定期扫描脚本核对 md5 值保证脚本的准确性，这个可根据脚本重要程度采用；

评审

脚本收敛的还有一项重要标准是脚本本身不产生明显副作用，即对业务、DB、系统带来比较明显的影响；所以在写完脚本以后，需要组织一些有经验的人做评审，如果脚本带来的伤害在允许范围内，才允许上线使用；

比如：
用遍历海量文件操作
处理上 G 大小文件使用 sort 、mv 、cp 等等
用超级用户使用 rm 的操作

脚本执行

为了防止误操作，不允许脚本不带任何参数执行