Shell脚本编写建议

by zpo

目标

Shell脚本面向内部技术人员，编写目标如下：

- 脚本编写简单小巧，结构清晰，便于维护

    1. 脚本内容需要分为若干步骤，同时打印步骤序号和步骤名称
    2. 一个脚本做的事情不超过5步，脚本内容过多则进行相应拆分,避免某一个脚本过于复杂
    3. 脚本的每个步骤以及函数实现遵循单一功能原则，代码长度控制在50行以内（50行大概是整个桌面窗口最大化后能全部展示的行数）
    4. 文件部署尽量清晰，及时清理废弃脚本和废弃文件夹，被注释掉的无用代码及时删除
    5. 入口脚本应该提供-help，-version功能

- 日志输出可以快速定位问题，打印格式规范统一

    1. 日志止于出错位置，并且能够明确指出错误位置
    2. 能够看到执行脚本的脚本名与执行时的具体参数
    3. 关键日志和非关键日志区分开来
    4. 不打印看起来没有意义或没有人会看的日志
    5. 调试脚本时打印的辅助日志，调试完成后请**务必**删掉

注释

注释使用准则：描述详细准确，注释管理得当

- 注释标准
    1. 重要文件、重要函数、重要步骤，添加必要的注释说明，并紧跟对应模块，注释内容包括模块用途，参数，返回值，作者，版本，修改内容日期等信息
    2. 其他具体实现细节，建议通过合理的命名、编程规范体现代码功能，需要用到注释时，描述要详细恰当，并像管理代码一样管理注释

- 注释管理

1. 代码功能如有修改，且存在对应的注释说明，注释也要跟着修改，无用的代码和注释，及时删除
2. 若存在未完成的内容、已知但未解决的问题、后期大概率用到的被注释代码，添加必要的注释说明，并添加***TODO***关键字以区分正常注释，避免他人阅读困扰

文件部署

- 为使各服务平台便于阅读，现规定引擎内部实现平台按以下文件目录部署

```
>script
        >config     配置文件
        >log        日志
        >autobuild     脚本
                >bin    主脚本
                >core   具体实现脚本
                >lib    引用的第三方库
                >tool   通用的工具脚本
                >...
        >temp       临时文件
        >...
```

- 脚本中将通用的变量处理为配置文件变量，配置文件与脚本实现放置于同一级，增加脚本可复用性
- lib存放非内部实现的第三方脚本，lib下按照第三方脚本功能命名二级文件夹，并于二级文件夹下建立文本文件，描述第三方脚本的功能，使用方法，引用者，版本，引用时间，如果是替换了别人原先使用的第三方库，还要注明替换原因，替换者，替换时间等信息
- 谨慎引用第三方库，如果没有对第三方库有一定了解，不要轻易使用
- tool存放内部实现的通用工具脚本，如日志输出脚本等等，tool下可按照脚本功能进行二级文件命名，工具脚本中添加注释说明工具脚本的实现者，版本号，使用方法等信息，如非第一版实现，应该添加修改内容，修改者，修改时间等信息

规约

命名风格统一

- 文件名、内部变量名使用 ***lowerCamelCase*** 小写驼峰形式，如: buildServerConsole.sh，resultNum
- 函数名使用 ***UpperCamelCase*** 大写驼峰形式，如: ReportFailure，AfterShell
- 常量名、路径名使用以下划线连接的全大写单词形式，如: QUIET_RUN，SCRIPTS_AUTOBUILD_DIR
- 命名时，除非某个单词有业内通用的缩写简写，否则不要使用单词缩写简写命名，**特别不要把自己平时的缩写简写写入公共脚本**

日志输出规范

编写了一套Shell的日志工具脚本，目前能够实现：统一格式打印，分步执行，时间戳，出错后停止输出日志，出错后打印Shell堆栈，区分关键日志与非关键日志。编写Shell脚本时请按照以下规范调用日志方法。 工具脚本svn路径：svn://192.168.21.208/engine_doc/编码规范/Shell编码规范/通用工具脚本/日志工具

1. 在调用任何日志方法之前，请先调用 *CleanFlagBeforeStart* 函数。该函数会清除上一次打包失败留下的标志。

2. 脚本开始时，调用 *BeforeShell* 函数，传入所有执行参数$@。该函数会打印执行脚本名称与所有参数：

```
BeforeShell $@

2018-09-05 [17:52:21]
2018-09-05 [17:52:21]    +   test.sh  BEGIN   +
2018-09-05 [17:52:21]    +   test.sh -n -r -android 90   +
2018-09-05 [17:52:21]
```

3. 脚本执行步骤时，打印步骤序号和步骤名称。调用 *Step* 函数，传入两个参数，第一个为步骤序号，第二个为步骤名称。该函数会打印步骤标志：

```
Step 1 "Copy Unity Resource"

2018-09-05 [17:52:21]
2018-09-05 [17:52:21] ==================== test.sh  STEP 1 : Copy Unity Resour
2018-09-05 [17:52:21]
```

4. 脚本结束时，调用 *AfterShell* 函数。该函数会打印脚本结束标志：

```
AfterShell

2018-09-05 [17:52:21]
2018-09-05 [17:52:21]    +   test.sh  FINISH   +
```

5. 需要打印关键步骤成功的标志时，调用 *ReportSuccess* 函数(默认为空)，可以传入关键步骤名称参数。该函数会打印成功结果标志：

```
ReportSuccess "Build Dynamic Framework Project"


****     Build Dynamic Framework Project SUCCESS      ****
```

6. 需要打印错误标志时，调用 *ReportFailure* 函数。该函数会打印调用者的行号、脚本名，并阻止之后的日志输出，并通过内建命令caller输出一个Shell的调用堆栈：

```
ReportFailure

****     line 10 test.sh REPORTED FAILURE     ****


    CALLER LIST
 - line 10 a test.sh
 - line 14 b test.sh
 - line 17 main test.sh
```

7. 非关键命令，如svn操作，cp命令执行前请调用 *BlankLine* 打印一个空行

8. 如果需要打印带TAG的Shell日志，可以在source logForShell.sh时传入TAG参数(默认为空)，则日志打印时均会带上该TAG输出：

```
source ./logForShell.sh [Shell]
Show "Using TAG to LOG"

2018-09-11 [10:08:26] [Shell] Using TAG to LOG
```

其他规范

1. 脚本设计编写时应考虑按功能模块拆分成独立脚本，执行时均使用sh或source执行，去掉分配脚本执行权限的过程

2. 变量使用时要加花括号

> $Ditch => ${Ditch}

3. 规范脚本的错误码，调用脚本后需要对脚本返回值$?进行相应处理，推荐每个调用其他脚本的脚本内部写一个处理返回值的函数。该函数最少应具备处理错误返回值的情况，如面对错误返回值不继续执行后续操作，需要清除相应标志位：

```
function HandleError(){
returnVal=$?
if [ $retVal -ne 0 ]; then
    Show " error exit status "$returnVal" last command"
    rm -f ${MARK_FILE}
    exit 0
fi
}
```

4. 引用命令输出时，使用$()而不用反引号``

> `pwd` => $(pwd)

5. 打印变量时，把变量名和变量值一起打印

> 如Show "$PLATFORM",
> 至少应该写成Show "PLATFORM : $PLATFORM"

6. 定义函数时在函数名前加上*function*:

```
function UpperCamelCase(){
        ...
}
```

7. if-else流程控制，then和条件判断请写在同一行:

```
if [ ... ];then
  ...
elif [ ... ]; then
  ...
else
  ...
fi
```

8. 区分$0与$BASH_SOURCE的区别，需要打印当前执行脚本名时使用$BASH_SOURCE

其他

如有其他大家觉得应该添加和修改的规范，请于**群里反馈**，或者直接跟**子博，文森，晓杰**讨论添加。