**Shell风格指南**

这本书开头和《Advanced Bash-Scripting Guide》基本一致，写了shell与bash的相关特点，

## 使用哪一个shell

一般我们选择linux/unix系统上功能相对完善，认可度强的bash。

可执行文件必须以＃！/ bin / bash和最少数量的标志开始。

## 何时使用shell

shell只应该用于小工具或简单的包装脚本。复杂的函数或方法间的调用不适合完全用shell。

这里提到了几点shell编程的准则：

1.如果需要调用其他实用程序，并且执行的操作相对较少，那么对于此任务而言，shell是可接受的选择。

2.如果性能很重要，可以使用shell以外的东西。（shell并不擅长对数据、复杂结构等的处理）

3.如果发现需要使用数组来完成$ {PIPESTATUS}的赋值，那么您应该使用Python。

4.如果您正在编写超过100行的脚本，则应该用Python代替。 请记住，脚本的增长。 尽早用另一种语言重写脚本，以避免在以后的日子里耗费时间重写。（所以我们应该花时间看看python）

## Shell文件和解释器调用

文件扩展

可执行文件应该没有扩展名（强烈偏好）或.sh扩展名。 库必须具有.sh扩展名，并且不应该有可执行权限。

在执行程序时不需要知道程序是用什么语言编写的，而且shell不需要扩展名，所以我们不希望使用可执行文件。

但是，对于库来说，知道它是什么语言是很重要的，有时候需要用不同语言的类似的库。除了语言特定的后缀外，这允许具有相同目的但不同语言的库文件被命名为相同。

SUID/SGID

SUID和SGID在shell脚本上是被禁止的，为了安全起见，我们明确禁止它，因为某些平台上仍然有可能突破suid、sgid的限制，在需要的时候，建议使用sudo代替。

## 环境

标准输出和错误输出

所有错误信息都应该发送到STDERR，这使得更容易将正常状态与实际问题分开，可以打印出错误信息来提高问题定位能力。

注释

用 # 开始的行，给出脚本的介绍、注释、实施意见、todo信息等，

注释是提高代码可读性的关键，在公司代码不仅是个人的财产，更多时候需要结合实际情况和大家共同解决，杂乱无章、没有注释的垃圾是不能阅读的。

## 格式化

这里提到了一些大家需要注意的约定俗成的良好习惯。

缩进：两个空格符。同一级代码同缩进，学会使用缩进而不是一味添加标签。（这也是python的好习惯）

字符串长度：一行最大不能超过80个字符。如果一行必须超过80字符，可以使用 \ 在末尾然后换行。

循环：shell中的循环有点不同，但是在声明函数时我们遵循与花括号相同的原则，do/then 和if/for/while在同一行，同一级的if/fi和if/else应该竖直对齐。

几个例子：

for dir in ${dirs\_to\_cleanup}; do

if [[ -d "${dir}/${ORACLE\_SID}" ]]; then

log\_date "Cleaning up old files in ${dir}/${ORACLE\_SID}"

rm "${dir}/${ORACLE\_SID}/"\*

if [[ "$?" -ne 0 ]]; then

error\_message

fi

else

mkdir -p "${dir}/${ORACLE\_SID}"

if [[ "$?" -ne 0 ]]; then

error\_message

fi

fi

done

扩展变量时注意：

首先应该和源代码格式一致，如：$var 或 ${var}

一般不要使用单引号去引特殊和位置参数，除非真正需要这么去做。

使用大括号去引用所有变量是个不错的选择，很少出错。

## 引用

引用需要遵循以下几个原则：

1. 变量、命令的替换、空格或者元字符串都需要引用。
2. 将选项或路径作为一个变量来引用。
3. 不要引用整数。
4. 一定注意[[中的引用规则，
5. 一般情况下不要使用$\* 而去使用$@

## 功能和BUG

命令替换

一般我们习惯将要执行后将结果传参的命令用 ` ` （反引号）去引用，但是这是不安全的，更推荐使用 $(command)去做同样的事。因为嵌套的反引号需要\来转义，但是$()不会在嵌套时更改，便于阅读。

例：

# This is preferred:

var="$(command "$(command1)")"

# This is not:

var="`command \`command1\``"

测试，[ 和 [[

一般来说，测试代码时[ 和 [[的用法相当，一个是内建函数，一个是关键字。

但是实际使用时，[[...]]优于[，test和/ usr / bin / [

因为[[...]]减少了错误，在[[ and ]]和[[...]]之间允许正则表达式匹配的地方没有路径名扩展或分词发生。

测试字符串

测试字符串时尽量使用引号而不是填充字符。在这一点上，bash做的很好，自动处理空字符串，在测试字符串时，灵活使用-z –n参数，来避免字符串的错误。

具体方法在另一篇笔记中写的详细。

# Use this

if [[ -n "${my\_var}" ]]; then

do\_something

fi

# Instead of this as errors can occur if ${my\_var} expands to a test

# flag

if [[ "${my\_var}" ]]; then

do\_something

fi

文件名的通配符

在删除或寻找某文件时，我们可以选择使用\*去匹配所有文件，但是，由于文件名可以是 – 开头，所以某些情况下会出现啼笑皆非的BUG，此时使用./\*要安全的多，如一下例子：

# Here's the contents of the directory:

# -f -r somedir somefile

# This deletes almost everything in the directory by force

psa@bilby$ rm -v \*

removed directory: `somedir'

removed `somefile'

# As opposed to:

psa@bilby$ rm -v ./\*

removed `./-f'

removed `./-r'

rm: cannot remove `./somedir': Is a directory

removed `./somefile'

eval

[**shell 中的 eval**](http://www.cnblogs.com/huzhiwei/archive/2012/03/14/2395956.html)

**功能说明**：重新运算求出参数的内容。  
  
**语　　法**：[eval](http://www.linuxso.com/command/.html) [参数]  
  
**补充说明**：eval可读取一连串的参数，然后再依参数本身的特性来执行。  
  
**参　　数**：参数不限数目，彼此之间用分号分开。

1.eval[命令](http://www.linuxso.com/command/)将会首先扫描命令行进行所有的替换，憨厚再执行命令。该命令使用于那些一次扫描无法实现其功能的变量。该命令对变量进行两次扫描。这些需要进行两次扫描的变量有时候被称为复杂变量。

2.eval也可以用于回显简单变量，不一定时复杂变量。

NAME=ZONE

eval echo $NAME等价于echo $NAME

3.两次扫描

test.txt内容：hello shell world!

my[file](http://www.linuxso.com/command/file.html)="[cat](http://www.linuxso.com/command/cat.html) test.txt"

(1)echo $myfile　　#re[su](http://www.linuxso.com/command/su.html)lt:cat test.txt

(2)eval echo $myfile　　#result:hello shell world!

从（2）可以知道第一次扫描进行了变量替换，第二次扫描执行了该字符串中所包含的命令

4.获得最后一个参数

echo "Last argument is $(eval echo \$$#)"

echo "Last argument is $(eval echo $#)"

shell 也提供了 eval 命令，如同熟悉的其他脚本语言，会将它的参数做为命令执行，初看会疑惑为什么shell要提供两种动态执行命令字串的机制，但是经过仔细分析，才发现shell的eval同其他语言有很大区别。  
  
1.shell 中的函数虽然可以通过return 返回，但是这里的return 相当于 [ex](http://www.linuxso.com/command/ex.html)it，只能是个状态值用于测试，而不能像其它语言一样返回复杂的结果，其处理结果只能通过输出到标准输出经过 `` ,$()取得。  
  
2.shell 中的 eval  
  
    2.1 不能获得函数处理结果 ，如1所说，所有命令，函数的处理结果只能通过 ``来获得，那么其它语言中利用eval来获得动态生成代码执行后的输出变得不可能。  
  
    2.2 eval 嵌套无意义 ，在其他语言中可以通过 eval(eval("code"))，来执行（执行动态生成的code的返回），而由于shell 中 eval 将后面的eval命令简单当作命令字符串执行，失去了嵌套作用，嵌套被命令替换取代。

将管道连接循环

最好不要将管道数据连上while循环，在while循环中修改的变量不会传播到父项，因为循环的命令在子shell中运行。一个管道中的隐式子shell可以使追踪错误变得困难。

例子：

last\_line='NULL'

your\_command | while read line; do

last\_line="${line}"

done

# This will output 'NULL'

echo "${last\_line}"

当确定输入不包括空格或特殊字符时，可以使用for循环。

使用进程替换可以重定向输出，但是将命令放在一个明确的子shell中，而不是bash为while循环创建的隐式子shell。

## 命名约定

函数名称

一个shell函数建立时，需要按照如下规则命名：

1. 是小写
2. 用下划线分隔单词
3. 用::分开库
4. 函数名称后需要括号
5. 关键字需要在项目中格式一致使用

如果您正在编写单个函数，请使用小写字母和单独的下划线。 如果你正在编写一个软件包，使用::来分开软件包名称。 大括号必须与函数名、函数名和括号之间没有空格。

例子：

# Single function

my\_func() {

...

}

# Part of a package

mypackage::my\_func() {

...

}

变量名称

一个循环中的变量名称应该和你的正在循环的变量类似（单数或复数等等），便于区别和阅读。

例：

for zone in ${zones}; do

something\_with "${zone}"

done

常量和环境变量

基本规则：所有大写，用下划线分隔，在文件顶部声明。常量和任何输出到环境中的数据都应该大写。

例：

# Constant

readonly PATH\_TO\_FILES='/some/path'

# Both constant and environment

declare -xr ORACLE\_SID='PROD'

有些变量会在第一次设置后不会再变，（比如通过getopts设置的）。所以可以在getopts中设置常量或基于条件来设置常量，但是应该在之后立即进行读取。

注：declare不会对函数内的全局变量进行操作，因此建议有全局变量时使用readonly或export。

源文件名

全程小写，如果需要，用下划线分隔单词。

只读变量

使用readonly或声明-r以确保它们是只读的。

由于全局变量在shell中被广泛使用，所以在处理它们的时候发现错误是非常重要的。 当你声明一个只读的变量时，要明确这一点。

比如说：

zip\_version="$(dpkg --status zip | grep Version: | cut -d ' ' -f 2)"

if [[ -z "${zip\_version}" ]]; then

error\_message

else

readonly zip\_version

fi

使用局部变量

只在本地使用的此函数中的特定变量。

我们要确保局部变量只能在声明它们的时候通过局部变量才能看到。这样就避免了全局名称污染。

例：

my\_func2() {

local name="$1"

# Separate lines for declaration and assignment:

local my\_var

my\_var="$(my\_func)" || return

# DO NOT do this: $? contains the exit code of 'local', not my\_func

local my\_var="$(my\_func)"

[[ $? -eq 0 ]] || return

...

}

函数位置

将所有函数放在常量下的文件中，不要在函数间隐藏可执行代码。

如果你有一些函数，将他们放在文件的最上面。声明函数之前，只能包含ser语句和设置常亮。

不要在函数之间隐藏可执行代码，这样会让代码难以调试，还会出现未知BUG

Main函数

在一个长脚本中，如果有一个以上的函数，那么这时候就需要有一个main函数作为入口。

这是因为，为了方便查找程序的开始，将主程序放在一个名为main的函数中作为最底层的函数。这与代码库的其余部分保持一致，并允许您将更多变量定义为局部变量，文件中的最后一个非注释行应该是对main的调用。

main "$@"

## 调用命令

检查返回值

我们需要对返回值经常检查，来保证代码如预期的运行，

对于非管道类的命令，最常用的检查返回值的方法就是 使用$? 或者直接用if语句

当然，对于管道类的命令。Bash也有PIPESTATUS变量，它允许从管道的所有部分检查返回代码。举个例子：

tar -cf - ./\* | ( cd "${dir}" && tar -xf - )

if [[ "${PIPESTATUS[0]}" -ne 0 || "${PIPESTATUS[1]}" -ne 0 ]]; then

echo "Unable to tar files to ${dir}" >&2

fi

要注意的是，当我们执行了其他命令后，PIPESTATUS将被覆盖，如果您需要根据管道中发生的错误采取不同的行为，则需要在运行命令后立即将PIPESTATUS分配给另一个变量。

这里提示到：不要忘记 [ 是一个命令，也会重置PIPESTATUS。

内建命令与外部命令

内建命令的优先级是最高的，因为他是系统选择的最可靠的命令。

以上是shell编程时的一些tips，他不会教我们如何编程，但是会让我们的程序更加规范、简单、可读。