编写可靠shell脚本的八个建议

bash spacewander 2 天前发布

这八个建议，来源于键者几年来编写 shell 脚本的一些经验和教训。事实上开始写的时候还不止这几条，后来思索再三，去掉几条无关痛痒的，最后剩下八条。毫不夸张地说，每条都是精挑细选的，虽然有几点算是老生常谈了。

1. 指定bash

shell 脚本的第一行，`#!`之后应该是什么？如果拿这个问题去问别人，不同的人的回答可能各不相同。
我见过`/usr/bin/env bash`，也见过`/bin/bash`，还有`/usr/bin/bash`，还有`/bin/sh`，还有`/usr/bin/env sh`。这算是编程界的"'茴'字四种写法"了。
在多数情况下，以上五种写法都是等价的。但是，写过程序的人都知道："少数情况"里往往隐藏着意想不到的坑。

如果系统的默认 shell 不是 bash 怎么办？比如某 Linux 发行版的某个版本，默认的 sh 就不是 bash。
如果系统的 bash 不是在`/usr/bin/bash`怎么办？

我推荐使用`/usr/bin/env bash`和`/bin/bash`。前者通过`env`添加一个中间层，让`env`在`$PATH`中搜索`bash`；后者则是官方背书的，约定俗成的 bash 位置，`/usr/bin/bash`不过是指向它的一个符号链接。

2. set -e 和 set -x

OK，经过一番讨论，现在第一行定下来了。接下来该开始写第二行了吧？
且慢！在你开始构思并写下具体的代码逻辑之前，先插入一行`set -e`和一行`set -x`。

`set -x`会在执行每一行 shell 脚本时，把执行的内容输出来。它可以让你看到当前执行的情况，里面涉及的变量也会被替换成实际的值。
`set -e`会在执行出错时结束程序，就像其他语言中的"抛出异常"一样。（准确说，不是所有出错的时候都会结束程序，见下面的注）

注：`set -e`结束程序的条件比较复杂，在`man bash`里面，足足用了一段话描述各种情景。大多数执行都会在出错时退出，除非 shell 命令位于以下情况：

1. 一个 pipeline 的非结尾部分，比如`error | ok`
2. 一个组合语句的非结尾部分，比如`ok && error || other`
3. 一连串语句的非结尾部分，比如`error; ok`
4. 位于判断语句内，包括`test`、`if`、`while`等等。

这两个组合在一起用，可以在 debug 的时候替你节省许多时间。出于防御性编程的考虑，有必要在写第一行具体的代码之前就插入它们。扪心自问，写代码的时候能够一次写对的次数有多少？大多数代码，在提交之前，通常都经历过反复调试修改的过程。与其在焦头烂额之际才引入这两个配置，不如一开始就给 debug 留下余地。在代码终于可以提交之后，再考虑是否保留它们也不迟。

3. 带上shellcheck

好了，现在我已经有了三行（样板）代码，具体的业务逻辑一行都没写呢。是不是该开始写了？
且慢！工欲善其事，必先利其器。这次，我就介绍一个 shell 脚本编写神器：shellcheck

说来惭愧，虽然写了几年 shell 脚本，有些语法我还是记不清楚。这时候就要依仗 shellcheck 指点一下了。shellcheck 除了可以提醒语法问题以外，还能检查出 shell 脚本编写常见的 bad code。本来我的N条建议里面，还有几条是关于这些 bad code 的，不过考虑到 shellcheck 完全可以发掘出这些问题，于是忍痛把它们都剔除在外了。毫无疑问，使用 shellcheck 给我的 shell 编写技能带来了巨大的飞跃。

所谓"站在巨人的肩膀上"，虽然我们这些新兵蛋子，技能不如老兵们强，但是我们可以在装备上赶上对方啊！动动手安装一下，就能结识一个循循善诱的"老师"，何乐而不为？

顺便一提，shellcheck 居然是用 haskell 写的。谁说 haskell 只能用来装逼？

4. 变量展开

在 shell 脚本中，偶尔可以看到这样的做法：`echo $xxx | awk/sed/grep/cut...` 。看起来大张形势的样子，其实不过是想修改一个变量的值。杀鸡何必用牛刀？bash内建的变量展开机制已经足以满足你各种需求！还是老方法， read the f**k manaul! `man bash` 然后搜索 `Parameter Expansion`，下面就是你想要的技巧。键者也写过一篇相关的文章，希望能助上一臂之力：玩转Bash变量

5. 注意local

随着代码越写越多，你开始把重复的逻辑提炼成函数。有可能你会掉到bash的一个坑里。在bash，如果不加 local 限定词，变量默认都是全局的。变量默认全局——这跟 js 和 lua 相似；但相较而言，很少有 bash 教程一开始就告知你这个事实。在顶级作用域里，是否是全局变量并不重要。但是在函数里面，声明一个全局变量可能会污染到其他作用域（尤其在你根本没有注意到这一点的情况下）。所以，对于在函数内声明的变量，请务必记得加上 local 限定词。

6. trap信号

如果你写过稍微复杂点的在后台运行的程序，应该知道 posix 标准里面“信号”是什么一回事。如果不知道，直接看下一段。像其他语言一样，shell 也支持处理信号。`trap sighandler INT` 可以在接收到 SIGINT 时调用 sighandler 函数。捕获其他信号的方式以此类推。

不过 trap 的主要应用场景可不是捕获哪个信号。`trap` 命令支持“捕获”许多不同的流程——准确来说，允许用户给特定的流程注入函数调用。其中最为常用的是 `trap func EXIT` 和 `trap func ERR`。

`trap func EXIT` 允许在脚本结束时调用函数。由于无论正常退出抑或异常退出，所注册的函数都能得以调用，在需要调用一个清理函数的场景下，我都是用它注册清理函数，而不是简单地在脚本结尾调用清理函数。

`trap func ERR` 允许在运行出错时调用函数。一个常用的技法是，使用全局变量ERROR存储错误信息，然后在注册的函数中根据存储的值完成对应的错误报告。把原本四分五裂的错误处理逻辑集中到一处，有时候会起奇效。不过要记住，程序异常退出时，既会调用EXIT注册的函数，也会调用ERR注册的函数。

7. 三思后行

以上几条都是具体的建议，剩下两条比较务虚。

这条建议的名字叫“三思而行”。其实无论写什么代码，哪怕只是一个辅助脚本，都要三思而行，切忌粗心大意。不，写脚本的时候更要记住这点。毕竟许多时候，一个复杂的脚本发端于几行小小的命令。一开始写这个脚本的人，也许以为它只是一次性任务。代码里难免对一些外部条件有些假定，在当时也许是正常的，但是随着外部环境的变化，这些就成了隐藏的暗礁。雪上加霜的是，几乎没有人会给脚本做测试。除非你去运行它，否则不知道它是否还能正常使用。

要想减缓脚本代码的腐烂速度，需要在编写的时候辨清哪些是会变的依赖、哪些是脚本正常运行所不可或缺的。要有适当的抽象，编写可变更的代码；同时要有防御性编程的意识，给自己的代码一道护城河。

8. 扬长避短

有些时候，使用 shell 写脚本就意味着难以移植、难以统一地进行错误处理、难以利索地处理数据。
虽然使用外部的命令可以方便快捷地实现各种复杂的功能，但作为硬币的反面，不得不依靠 `grep`、`sed`、`awk` 等各种工具把它们粘合在一起。
如果有兼容多平台的需求，还得小心规避诸如BSD和GNU coreutils，bash版本差异之类奇奇怪怪的陷阱。
由于缺乏完善的数据结构以及一致的API，shell 脚本在处理复杂的逻辑上力不从心。

解决特定的问题要用合适的工具。知道什么时候用 shell，什么时候切换到另外一门更通用的脚本语言（比如ruby/python/perl），这也是编写可靠 shell 脚本的诀窍。如果你的任务可以组合常见的命令来完成，而且只涉及简单的数据，那么 shell 脚本就是适合的锤子。如果你的任务包含较为复杂的逻辑，而且数据结构复杂，那么你需要用ruby/python之类的语言编写脚本。

3 推荐 收藏

你可能感兴趣的文章

BASH入门 3 收藏，188 浏览

Linux启动脚本（设置环境变量） 1 收藏，343 浏览

跟我一起写shell补全脚本（开篇） 10 收藏，908 浏览

讨论区

很好很好，看得出笔者经验丰富

P_Chou · 2 天前

回复 P_Chou:
我可是有丰富的失败经验的

spacewander · 1 天前

求个shell脚本的入门以及深入的教程书籍都可以

jichaowu · 32 分钟前

使用评论询问更多信息或提出修改意见，请不要在评论里回答问题

提交评论

评论支持部分 Markdown 语法：`**bold**` `_italic_` `[link](http://example.com)` `> 引用` `` `code` `` `- 列表`。
同时，被你 @ 的用户也会收到通知

本文隶属于专栏

spacewander

这个专栏什么都有，大部分都是关于Linux或后端开发的

spacewander
作者

关注专栏

相关收藏夹

Bash command 速查手册
20 个条目 | 81 人关注

linux
5 个条目 | 0 人关注

分享扩散: