数组 脚本除错

set 命令，shopt 命令

网道（WangDoc.com），互联网文档计划

`set` 命令是 Bash 脚本的重要环节，却常常被忽视，导致脚本的安全性和可维护性出问题。本章介绍 `set` 的基本用法，帮助你写出更安全的 Bash 脚本。

目录 [隐藏]

1. 简介

我们知道，Bash 执行脚本时，会创建一个子 Shell。

```
$ bash script.sh
```

上面代码中， `script.sh` 是在一个子 Shell 里面执行。这个子 Shell 就是脚本的执行环境，Bash 默认给定了这个环境的各种参数。

`set` 命令用来修改子 Shell 环境的运行参数，即定制环境。一共有十几个参数可以定制，官方手册有完整清单，本章介绍其中最常用的几个。

顺便提一下，如果命令行下不带任何参数，直接运行 `set` ，会显示所有的环境变量和 Shell 函数。

```
$ set
```

2. set -u

执行脚本时，如果遇到不存在的变量，Bash 默认忽略它。

```
#!/usr/bin/env bash

echo $a
echo bar
```

上面代码中， `$a` 是一个不存在的变量。执行结果如下。

```
$ bash script.sh

bar
```

可以看到， `echo $a` 输出了一个空行，Bash 忽略了不存在的 `$a` ，然后继续执行 `echo bar` 。大多数情况下，这不是开发者想要的行为，遇到变量不存在，脚本应该报错，而不是一声不响地往下执行。

`set -u` 就用来改变这种行为。脚本在头部加上它，遇到不存在的变量就会报错，并停止执行。

链接

本文源码

代码仓库

反馈

```
#!/usr/bin/env bash
set -u

echo $a
echo bar
```

运行结果如下。

```
$ bash script.sh
bash: script.sh:行4: a: 未绑定的变量
```

可以看到，脚本报错了，并且不再执行后面的语句。

`-u` 还有另一种写法 `-o nounset` ，两者是等价的。

```
set -o nounset
```

3. set -x

默认情况下，脚本执行后，只输出运行结果，没有其他内容。如果多个命令连续执行，它们的运行结果就会连续输出。有时会分不清，某一段内容是什么命令产生的。

`set -x` 用来在运行结果之前，先输出执行的那一行命令。

```
#!/usr/bin/env bash
set -x

echo bar
```

执行上面的脚本，结果如下。

```
$ bash script.sh
+ echo bar
bar
```

可以看到，执行 `echo bar` 之前，该命令会先打印出来，行首以 `+` 表示。这对于调试复杂的脚本是很有用的。

`-x` 还有另一种写法 `-o xtrace` 。

```
set -o xtrace
```

脚本当中如果要关闭命令输出，可以使用 `set +x` 。

```
#!/bin/bash

number=1

set -x
if [ $number = "1" ]; then
  echo "Number equals 1"
else
  echo "Number does not equal 1"
fi
set +x
```

上面的例子中，只对特定的代码段打开命令输出。

4. Bash 的错误处理

如果脚本里面有运行失败的命令（返回值非 `0` ），Bash 默认会继续执行后面的命令。

```
#!/usr/bin/env bash

foo
echo bar
```

上面脚本中， `foo` 是一个不存在的命令，执行时会报错。但是，Bash 会忽略这个错误，继续往下执行。

```
$ bash script.sh
script.sh:行3: foo: 未找到命令
bar
```

可以看到，Bash 只是显示有错误，并没有终止执行。

这种行为很不利于脚本安全和除错。实际开发中，如果某个命令失败，往往需要脚本停止执行，防止错误累积。这时，一般采用下面的写法。

```
command || exit 1
```

上面的写法表示只要 `command` 有非零返回值，脚本就会停止执行。

如果停止执行之前需要完成多个操作，就要采用下面三种写法。

```
# 写法一
command || { echo "command failed"; exit 1; }

# 写法二
if ! command; then echo "command failed"; exit 1;

# 写法三
command
if [ "$?" -ne 0 ]; then echo "command failed"; exi
```

另外，除了停止执行，还有一种情况。如果两个命令有继承关系，只有第一个命令成功了，才能继续执行第二个命令，那么就要采用下面的写法。

```
command1 && command2
```

5. set -e

上面这些写法多少有些麻烦，容易疏忽。 `set -e` 从根本上解决了这个问题，它使得脚本只要发生错误，就终止执行。

```
#!/usr/bin/env bash
set -e
```

```
foo
echo bar
```

执行结果如下。

```
$ bash script.sh
script.sh:行4: foo: 未找到命令
```

可以看到，第4行执行失败以后，脚本就终止执行了。

`set -e` 根据返回值来判断，一个命令是否运行失败。但是，某些命令的非零返回值可能不表示失败，或者开发者希望在命令失败的情况下，脚本继续执行下去。这时可以暂时关闭 `set -e` ，该命令执行结束后，再重新打开 `set -e` 。

```
set +e
command1
command2
set -e
```

上面代码中， `set +e` 表示关闭 `-e` 选项， `set -e` 表示重新打开 `-e` 选项。

还有一种方法是使用 `command || true` ，使得该命令即使执行失败，脚本也不会终止执行。

```
#!/bin/bash
set -e

foo || true
echo bar
```

上面代码中， `true` 使得这一行语句总是会执行成功，后面的 `echo bar` 会执行。

`-e` 还有另一种写法 `-o errexit` 。

```
set -o errexit
```

6. set -o pipefail

`set -e` 有一个例外情况，就是不适用于管道命令。

所谓管道命令，就是多个子命令通过管道运算符（`|`）组合成为一个大的命令。Bash 会把最后一个子命令的返回值，作为整个命令的返回值。也就是说，只要最后一个子命令不失败，管道命令总是会执行成功，因此它后面命令依然会执行，`set -e` 就失效了。

请看下面这个例子。

```
#!/usr/bin/env bash
set -e

foo | echo a
echo bar
```

执行结果如下。

```
$ bash script.sh
a
script.sh:行4: foo: 未找到命令
bar
```

上面代码中，`foo` 是一个不存在的命令，但是 `foo | echo a` 这个管道命令会执行成功，导致后面的 `echo bar` 会继续执行。

`set -o pipefail` 用来解决这种情况，只要一个子命令失败，整个管道命令就失败，脚本就会终止执行。

```
#!/usr/bin/env bash
set -eo pipefail

foo | echo a
echo bar
```

运行后，结果如下。

```
$ bash script.sh
a
script.sh:行4: foo: 未找到命令
```

可以看到， `echo bar` 没有执行。

7. set -E

一旦设置了 `-e` 参数，会导致函数内的错误不会被 `trap` 命令捕获（参考《trap 命令》一章）。 `-E` 参数可以纠正这个行为，使得函数也能继承 `trap` 命令。

```
#!/bin/bash
set -e

trap "echo ERR trap fired!" ERR

myfunc()
{
  # 'foo' 是一个不存在的命令
  foo
}

myfunc
```

上面示例中， `myfunc` 函数内部调用了一个不存在的命令 `foo` ，导致执行这个函数会报错。

```
$ bash test.sh
test.sh:行9: foo: 未找到命令
```

但是，由于设置了 `set -e` ，函数内部的报错并没有被 `trap` 命令捕获，需要加上 `-E` 参数才可以。

```
#!/bin/bash
set -Eeuo pipefail

trap "echo ERR trap fired!" ERR
```

```
myfunc()
{
  # 'foo' 是一个不存在的命令
  foo
}

myfunc
```

执行上面这个脚本，就可以看到 `trap` 命令生效了。

```
$ bash test.sh
test.sh:行9: foo: 未找到命令
ERR trap fired!
```

8. 其他参数

`set` 命令还有一些其他参数。

- `set -n` ：等同于 `set -o noexec` ，不运行命令，只检查语法是否正确。
- `set -f` ：等同于 `set -o noglob` ，表示不对通配符进行文件名扩展。
- `set -v` ：等同于 `set -o verbose` ，表示打印 Shell 接收到的每一行输入。
- `set -o noclobber` ：防止使用重定向运算符 `>` 覆盖已经存在的文件。

上面的 `-f` 和 `-v` 参数，可以分别使用 `set +f` 、 `set +v` 关闭。

9. set 命令总结

上面重点介绍的 `set` 命令的几个参数，一般都放在一起使用。

```
# 写法一
set -Eeuxo pipefail
```

```
# 写法二
set -Eeux
set -o pipefail
```

这两种写法建议放在所有 Bash 脚本的头部。

另一种办法是在执行 Bash 脚本的时候，从命令行传入这些参数。

```
$ bash -euxo pipefail script.sh
```

10. shopt 命令

`shopt` 命令用来调整 Shell 的参数，跟 `set` 命令的作用很类似。之所以会有这两个类似命令的主要原因是， `set` 是从 Ksh 继承的，属于 POSIX 规范的一部分，而 `shopt` 是 Bash 特有的。

直接输入 `shopt` 可以查看所有参数，以及它们各自打开和关闭的状态。

```
$ shopt
```

`shopt` 命令后面跟着参数名，可以查询该参数是否打开。

```
$ shopt globstar
globstar  off
```

上面例子表示 `globstar` 参数默认是关闭的。

（1）-s

`-s` 用来打开某个参数。

```
$ shopt -s optionNameHere
```

（2）-u

`-u` 用来关闭某个参数。

```
$ shopt -u optionNameHere
```

举例来说， `histappend` 这个参数表示退出当前 Shell 时，将操作历史追加到历史文件中。这个参数默认是打开的，如果使用下面的命令将其关闭，那么当前 Shell 的操作历史将替换掉整个历史文件。

```
$ shopt -u histappend
```

（3）-q

`-q` 的作用也是查询某个参数是否打开，但不是直接输出查询结果，而是通过命令的执行状态（ `$?` ）表示查询结果。如果状态为 `0` ，表示该参数打开；如果为 `1` ，表示该参数关闭。

```
$ shopt -q globstar
$ echo $?
1
```

上面命令查询 `globstar` 参数是否打开。返回状态为 `1` ，表示该参数是关闭的。

这个用法主要用于脚本，供 `if` 条件结构使用。下面例子是如果打开了这个参数，就执行 `if` 结构内部的语句。

```
if (shopt -q globstar); then
  ...
fi
```

11. 参考链接

- The Set Builtin
- Safer bash scripts with 'set -euxo pipefail'
- Writing Robust Bash Shell Scripts

数组 脚本除错

分享本文 

联系：contact@wangdoc.com