

# bash shell 脚本简介

## shell 运行环境

如果你运行的是 Unix 或 Linux 系统 ,例如 Ubuntu ,Red Hat ,SUSE Linux ,还有 macOS ,都是内置了 bash shell 的 ,所以你不需要额外配置所谓的开发环境。

我的 shell 环境是 macOS Sierra 版本 ,如果你用的是其他 Linux 系统 ,后面的例子基本上都是可以运行的。

首先 ,打开 Terminal 命令行 ,先检查下你的系统的 shell 版本 :

```
echo $BASH_VERSION
```



```
[...]$ echo $BASH_VERSION
3.2.57(1)-release
[...]$
```

## bash 命令层次结构

### 命令类型

bash shell 内置了一个 type 命令会根据你输入的单词来显示此命令的类型 ,主要有以下五种类型 :

别名

方法

shell 内置命令

关键字

文件

例如，我们经常使用的 `cd` 命令，我们来执行下面的命令来看下它属于哪种类型。

```
type cd
```

```
[root@localhost ~]$ type cd  
cd is a shell builtin
```

同时，为了查看更加详细的信息，可以使用

```
type -a cd
```

```
[root@localhost ~]$ type -a cd  
cd is a shell builtin  
cd is /usr/bin/cd
```

如果想查看的信息更加简洁和适合人们理解，可以使用如下命令和参数：

```
type -t ls
```

```
[root@localhost ~]$ type -t cd  
builtin
```

## **PATH 命令**

Linux 会检查配置在 `PATH` 环境中的指定路径的程序是否可以执行。通常情况下，当前目录是不会被查找，除非你把它配置到 `PATH` 中，我们可以执行如下命令，把当前目录加到 `PATH` 环境中。

```
export PATH=$PATH:.
```

接下来，我们创建一个存放 `shell` 脚本的目录 `bin`，可以用如下命令来执行：

```
$ test -d $HOME/bin || mkdir $HOME/bin
```

当然你可以在你的主目录下手动创建目录 `bin`。上面的意思是检查主目录是否有 `bin` 目录，没有则创建。

## 创建脚本

没啥说的，学习每一种语言的第一个程序就是“Hello，World”，文件名叫 hello1.sh。

```
#!/bin/bashecho "Hello World"exit 0
```

解释一下：

#!/bin/bash：通常情况下，脚本的默认第一行代码就是它。“#!”又被成为 shebang。它用来告诉系统的解释器来执行脚本。除了 bash，我们还可以 PHP，Perl 等其他脚本。

echo "Hello World"：echo 是一个内置的命令，用来表示标准输出，类似于 Java 中的 System.out.println()。

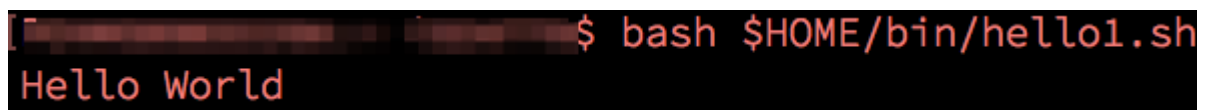
exit 0：表示脚本结束退出，exit 有一个整型参数，0 表示正常退出，非 0 表示脚本执行中有错误。

## 执行脚本

现在，我们来执行上面的脚本，你可以在命令行下，进入到脚本文件存在的目录，也可以是在任意目录下，但是，在执行的时候，文件的路径就应该为绝对路径：

```
bash $HOME/bin/hello1.sh
```

执行结果为，打印出“Hello world”。

A terminal window with a black background and red text. The prompt is a red dollar sign '\$'. The command entered is 'bash \$HOME/bin/hello1.sh'. The output is 'Hello World' on the next line.

```
$ bash $HOME/bin/hello1.sh
Hello World
```

Tip

在执行上面的命令中，可能会报错，提示权限不足或访问拒绝的错误。这是因为 `hello1.sh` 没有执行的权限。所以我们使用如下命令给文件加上对应的权限。

```
chmod +x $HOME/bin/hello1.sh
```

脚本中的一些特殊参数

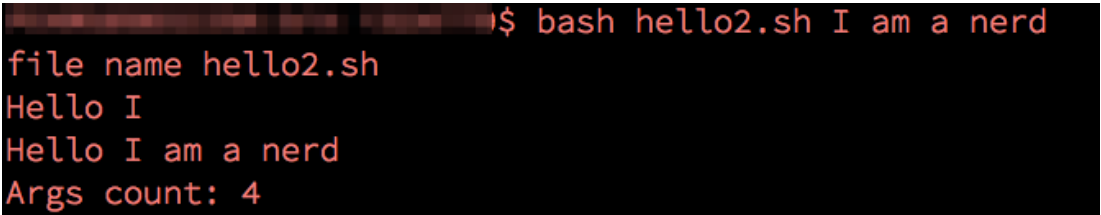
在脚本中，有些表示特殊含义的参数，下面列出常见的几个：

参数标识符	含义
<code>\$0</code>	文件本身的名字
<code>\$1</code>	表示位置的参数，第一个参数传递给脚本
<code>\${10}</code>	在超过两位数的参数时，使用大括号限定起来
<code>\$#</code>	参数的个数
<code>\$*</code>	表示所有的参数

如下所示：

```
#!/bin/bashecho "file name $(basename $0)"echo "Hello $1"echo "Hello $*"echo "Args count: $#"\exit 0
```

输出的结果为：

A terminal window with a black background and red text. The prompt is '\$ bash hello2.sh I am a nerd'. The output consists of four lines: 'file name hello2.sh', 'Hello I', 'Hello I am a nerd', and 'Args count: 4'.

```
$ bash hello2.sh I am a nerd
file name hello2.sh
Hello I
Hello I am a nerd
Args count: 4
```

重视引号的正确使用

到现在，我们使用了双引号来包围字符串用于 `echo` 命令的输出。

在第一个 `Hello1.sh` 中，使用单引号还是双引号，效果是一样的。下面的两行代码是等效的。

```
echo "Hello World"echo 'Hello World'
```

但是，在包含有变量的引用时，单引号和双引号的效果是不一样的。

```
echo "Hello $1" // 打印传递的值，例如 Tim。echo 'Hello $1' // 把$1 原样打印出来
```

所以，在有变量的字符串里，推荐使用双引号。这时，\$1 就会被变量的值所替代，而不是作为字符串显示出来。

## 打印脚本名字

前面提到过，特殊参数\$0 用来表示脚本的名字，这里的名字会带有完整的路径，如果我们只想要文件名的话，可以使用以下代码：

```
echo "You are using $(basename $0)"
```

这里\$(....)语法的作用是我们先执行括号里面的命令，然后然后把结果赋给外面不知名的变量。

\$(....)语法还有一种相等的写法，注意，是键盘上数字 1 左边的按键，不是单引号。

```
echo "You are using `basename $0`"
```

个人不推荐这种写法，太容易混淆，不容易差错。

## 调试你的脚本

如果想调试你的脚本，bash 给我们提供了两个选项：-v 和-x。

如果我们想逐行详细地查看脚本的内容，可以使用-v 选项。

```
#!/bin/bashecho "file name $(basename $0)"echo "Hello $1"echo "Hello $*"echo "Args count: $#"
```

```
[PEKM50903229A:bin i324779$ bash -v hello2.sh Tim
#!/bin/bash
echo "file name $(basename $0)"
basename $0
file name hello2.sh
echo "Hello $1"
Hello Tim
echo "Hello $*"
Hello Tim
echo "Args count: $#"
```

更常用的是-x 选项，它们在执行时显示命令。当我们决定选择分支的时候，更加使用。

```
[PEKM50903229A:bin i324779$ bash -x hello2.sh Tim
++ basename hello2.sh
+ echo 'file name hello2.sh'
file name hello2.sh
+ echo 'Hello Tim'
Hello Tim
+ echo 'Hello Tim'
Hello Tim
+ echo 'Args count: 1'
Args count: 1
+ exit 0
```

可以看到，basename 最先执行了，使用此选项不会看到代码的详情。