

Bash 函数

网道 (WangDoc.com) , 互联网文档计划

本章介绍 Bash 函数的用法。

目录 [隐藏]

1. 简介
2. 参数变量
3. return 命令
4. 全局变量和局部变量, local 命令
5. 参考链接

1. 简介

函数 (function) 是可以重复使用的代码片段, 有利于代码的复用。它与别名 (alias) 的区别是, 别名只适合封装简单的单个命令, 函数则可以封装复杂的多行命令。

函数总是在当前 Shell 执行, 这是跟脚本的一个重大区别, Bash 会新建一个子 Shell 执行脚本。如果函数与脚本同名, 函数会优先执行。但是, 函数的优先级不如别名, 即如果函数与别名同名, 那么别名优先执行。

Bash 函数定义的语法有两种。

```
# 第一种
fn() {
```

📖 Bash 脚本教程

- 📖 1. 简介
- 📖 2. 基本语法
- 📖 3. 模式扩展
- 📖 4. 引号和转义
- 📖 5. 变量
- 📖 6. 字符串操作
- 📖 7. 算术运算
- 📖 8. 操作历史
- 📖 9. 行操作
- 📖 10. 目录堆栈
- 📖 11. 脚本入门
- 📖 12. read 命令
- 📖 13. 条件判断
- 📖 14. 循环
- 📖 15. 函数
- 📖 16. 数组
- 📖 17. set 命令, shopt 命令

```
# codes
}

# 第二种
function fn() {
    # codes
}
```

上面代码中，`fn` 是自定义的函数名，函数代码就写在大括号之中。这两种写法是等价的。

下面是一个简单函数的例子。

```
hello() {
    echo "Hello $1"
}
```

上面代码中，函数体里面的 `$1` 表示函数调用时的第一个参数。

调用时，就直接写函数名，参数跟在函数名后面。

```
$ hello world
Hello world
```


下面是一个多行函数的例子，显示当前日期时间。


```
today() {
    echo -n "Today's date is: "
    date +"%A, %B %-d, %Y"
}
```


删除一个函数，可以使用 `unset` 命令。


```
unset -f functionName
```

查看当前 Shell 已经定义的所有函数，可以使用 `declare` 命令。

 **18.** 脚本除错

 **19.** mktemp 命令，trap 命令

 **20.** 启动环境

 **21.** 命令提示符

链接

 本文源码

 代码仓库

 反馈

```
$ declare -f
```

上面的 `declare` 命令不仅会输出函数名，还会输出所有定义。输出顺序是按照函数名的字母表顺序。由于会输出很多内容，最好通过管道命令配合 `more` 或 `less` 使用。

`declare` 命令还支持查看单个函数的定义。

```
$ declare -f functionName
```

`declare -F` 可以输出所有已经定义的函数名，不含函数体。

```
$ declare -F
```

2. 参数变量

函数体内可以使用参数变量，获取函数参数。函数的参数变量，与脚本参数变量是一致的。

- `$1 ~ $9`：函数的第一个到第9个的参数。
- `$0`：函数所在的脚本名。
- `$#`：函数的参数总数。
- `$@`：函数的全部参数，参数之间使用空格分隔。
- `$*`：函数的全部参数，参数之间使用变量 `$IFS` 值的第一个字符分隔，默认为空格，但是可以自定义。

如果函数的参数多于9个，那么第10个参数可以用 `${10}` 的形式引用，以此类推。

下面是一个示例脚本 `test.sh`。

```
#!/bin/bash
# test.sh

function alice {
    echo "alice: $@"
    echo "$0: $1 $2 $3 $4"
```

```
    echo "$# arguments"

}

alice in wonderland
```

运行该脚本，结果如下。

```
$ bash test.sh
alice: in wonderland
test.sh: in wonderland
2 arguments
```

上面例子中，由于函数 `alice` 只有第一个和第二个参数，所以第三个和第四个参数为空。

下面是一个日志函数的例子。

```
function log_msg {
    echo "[`date '+ %F %T'` ]: $@"
}
```

使用方法如下。

```
$ log_msg "This is sample log message"
[ 2018-08-16 19:56:34 ]: This is sample log message
```



3. return 命令

`return` 命令用于从函数返回一个值。函数执行到这条命令，就不再往下执行了，直接返回了。

```
function func_return_value {
    return 10
}
```

函数将返回值返回给调用者。如果命令行直接执行函数，下一个命令可以用 `$?` 拿到返回值。

```
$ func_return_value
$ echo "Value returned by function is: $?"
Value returned by function is: 10
```

`return` 后面不跟参数，只用于返回也是可以的。

```
function name {
    commands
    return
}
```

4. 全局变量和局部变量，local 命令

Bash 函数体内直接声明的变量，属于全局变量，整个脚本都可以读取。这一点需要特别小心。

```
# 脚本 test.sh
fn () {
    foo=1
    echo "fn: foo = $foo"
}

fn
echo "global: foo = $foo"
```

上面脚本的运行结果如下。

```
$ bash test.sh
fn: foo = 1
global: foo = 1
```

上面例子中，变量 `$foo` 是在函数 `fn` 内部声明的，函数体外也可以读取。

函数体内不仅可以声明全局变量，还可以修改全局变量。

```
#!/bin/bash
foo=1

fn () {
    foo=2
}

fn

echo $foo
```

上面代码执行后，输出的变量 `$foo` 值为2。

函数里面可以用 `local` 命令声明局部变量。

```
#!/bin/bash
# 脚本 test.sh
fn () {
    local foo
    foo=1
    echo "fn: foo = $foo"
}

fn
echo "global: foo = $foo"
```

上面脚本的运行结果如下。

```
$ bash test.sh
fn: foo = 1
global: foo =
```

上面例子中，`local` 命令声明的 `$foo` 变量，只在函数体内有效，函数体外没有定义。

5. 参考链接

- [How to define and use functions in Linux Shell Script](#), by Pradeep Kumar

◀ 循环

数组 ▶

本教程采用[知识共享 署名-相同方式共享 3.0协议](#)。

分享本文      

联系: contact@wangdoc.com