Shell编程与运维

> **Organization**: 千锋教育 Python 教学部
>
> **Date** : 2019-08-28
>
> **Author**: 张旭

一、Shell 脚本概述

通过 Shell 中的各种命令, 开发者和运维人员可以对服务器进行维护工作。

但每次都手动输入命令, 工作效率太低, 而且很容易出错, 尤其是需要维护大量服务器时。

为了能够对服务器批量执行操作, 我们可以将需要执行的命令写入文件, 批量执行, 这种文件便是 Shell 脚本。

Shell 脚本一般是以 `.sh` 结尾的文本文件, 当然, 也可以省略扩展名。

二、Shell 脚本首行

脚本文件第一行通过注释的方式指明执行脚本的程序。

在 Shell 脚本中, `#` 开头的文本是注释, 但第一句 `#!` 开头的这句话比较特殊, 他会告诉 Shell 应该使用哪个程序来执行当前脚本。

常见方式有:

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

Python 脚本的第一句一般是 `#!/usr/bin/env python`

三、第一个脚本

1. 创建 cpu-count.sh 文件
2. 将下面文本写入 cpu-count.sh 中

```
#!/bin/bash

echo "Hello"
echo "I am `whoami`"
echo "I love Linux"
echo "The CPU in my PC has `cat /proc/cpuinfo |grep -c processor` cores"
exit 0
```

3. 执行 `chmod a+x cpu-count.sh` 对脚本授予可执行权限
4. 输入 `./cpu-count.sh` 执行脚本
5. 查看脚本的退出状态: `echo $?`
    - Linux 中的所有程序执行结束后都有状态码
    - 状态码为 **零** 表示正常, 状态码为 **正整数** 代表异常退出

四、变量

1. 定义

   变量的定义与其他语言差距不大, 需要注意的是赋值前后没有空格。

   ```
   # 变量定义：等号前后没有空格
   a=12345
   b=xyz
   ```

2. 使用

   使用变量时, 变量名前面加上 `$` 符

   ```
   echo "---$a---\n===$b==="
   printf "---$a---\n===$b===\n"
   ```

3. 注意 引号 的差别

   ```
   echo "---$a---\n===$b==="
   echo '---$a---\n===$b==='
   ```

4. 定义当前Shell下的全局变量
    1. 定义: `export ABC=9876543210123456789`
    2. 定义完后, 在终端里用 source 加载脚本: `source ./test.sh`
5. 常用的系统环境变量
    - `$PATH`: 可执行文件目录
    - `$PWD`: 当前目录
    - `$HOME`: 家目录
    - `$USER`: 当前用户名
    - `$UID`: 当前用户的 uid

五、分支语句: `if`

分支控制语句完整格式为:

```
if command
then
    commands
elif command
    commands
else
    commands
fi
```

1. `if` 语句检查判断的依据实际上是, 后面所跟的命令的状态码: 0 为 true, 其他值 为 false

    ```
    if ls /xxx
    then
        echo 'exist xxx'
    else
        echo 'not exist xxx'
    fi
    ```

2. 条件测试命令: `[ ... ]`
    - shell 提供了一种专用做条件测试的语句 `[ ... ]`
    - 这一对方括号本质上是一个命令, 里面的条件是其参数, 所以 `[` 的后面和 `]` 的前面必须有空格, 否则会报错。
    - 他可以进行三种比较
        - 数值比较
        - 字符串比较
        - 文件比较
    - 用法:

        ```
        if [ condition ]
        then
            commands
        fi
        ```

3. 条件列表
    - 数值比较

Condition	说明
n1 -eq n2	检查n1是否与n2相等
n1 -ge n2	检查n1是否大于或等于n2
n1 -gt n2	检查n1是否大于n2
n1 -le n2	检查n1是否小于或等于n2
n1 -lt n2	检查n1是否小于n2
n1 -ne n2	检查n1是否不等于n2

- 字符串比较

Condition	说明
str1 = str2	检查str1是否和str2相同
str1 != str2	检查str1是否和str2不同
str1 < str2	检查str1是否比str2小
str1 > str2	检查str1是否比str2大
-n str1	检查str1的长度是否非0
-z str1	检查str1的长度是否为0

- 文件比较

Condition	说明
-d file	检查file是否存在并是一个目录
-e file	检查file是否存在
-f file	检查file是否存在并是一个文件
-r file	检查file是否存在并可读
-w file	检查file是否存在并可写
-x file	检查file是否存在并可执行
-s file	检查file是否存在并非空
-O file	检查file是否存在并属当前用户所有
-G file	检查file是否存在并且默认组与当前用户相同
file1 -nt file2	检查file1是否比file2新
file1 -ot file2	检查file1是否比file2旧

六、循环语句: for

Shell 中的循环结构有三种: `for`、`while` 和 `until`, 这里重点介绍 `for` 循环。

1. for 循环的基本格式:

```
for 变量 in 序列 do
    要执行的命令
done
```

2. 练习: 打印 1 到 10 中的偶数

```
for i in `seq 1 10`
do
    if [[ $[ $i % 2] == 0 ]]
    then
        echo "偶数: $i"
    else
        echo "奇数: $i"
    fi
done
```

    1. `seq START END` 语句用来产生一个数字序列
    2. `$[ NUM1 + NUM2 ]` 语句用来进行基本的数学运算
    3. `[[ ... ]]` 语句用来更方便的进行比较判断
3. C语言风格的 for 循环

```
for ((i=0; i<10; i++))
do
    echo "num is $i"
done
```

七、函数

1. 函数定义

   定义时 `function` 不是必须的，可以省略

```
function foo() {
    echo "----------------------------"
    echo "Hello $1, nice to meet you!"
    echo "----------------------------"
}
```

2. 函数的使用

   - 在终端或脚本中直接输入函数名即可，不需要小括号
   - 传参也只需将参数加到函数名后面，以空格做间隔，像正常使用命令那样

```
foo seamile
```

3. 函数的参数

```
bar() {
    echo "执行者是 $0"
    echo "参数数量是 $#"
    echo "全部的参数 $@"
    echo "全部的参数 $*"

    if [ -d $1 ]; then  # 检查传入的第一个参数是否是文件夹
        for f in `ls $1`
        do
            echo $f
        done
    elif [ -f $1 ]; then
        echo 'This is a file: $1'  # 单引号内的变量不会被识别
        echo "This is a file: $1"  # 如果不是文件夹，直接显示文件名
    else
        echo 'not valid'  # 前面都不匹配显示默认语句
    fi
}
```

八、获取用户输入

```
read -p "请输入一个数字：" num
echo "您输入的是：$num"
```