上章回顾

- 在httpd.conf中如何调整才能使得Apache支持PHP类型的网页?
- LAMP中的”P”可能是哪几种语言?
- LAMP中的四个组件分别是什么?

本章目标

- 掌握Shell脚本的基础知识
- 学会使用Shell变量
- 学会编写简单的Shell脚本

一、shell脚本基础

1、Shell的作用

1）Linux中的解释器，介于系统和用户之间，负责对用户输入的指令进行翻译，充当“翻译官”的角色

2）Shell也属于一种应用程序，连接了用户和Linux内核，让用户更高效、安全、低成本的使用Linux内核完成工作

3）Shell不属于内核一部分，是在内核的基础上的编写的一个应用程序，理论上和QQ、微信、迅雷等程序没有区别

4）与其他应用程序不同的是Shell会随着系统开机自动启动，用户必须通过Shell来使用Linux

5）目前最流行的Shell环境是bash shell

2、用户登录的Shell

1）登录到系统的用户一般使用的shell程序是/bin/bash

2）不同shell环境的内部指令、运行环境会有所区别

- Bourne shell (sh)：这是Unix系统中最早的Shell之一，语法相对简单，功能较为基础。它是其他Shell的基础。
- Bourne Again shell (bash)：因为Linux操作系统缺省的shell，所以也是最常见的一种shell
- C shell (csh)：这种shell的语法接近于C语言，由柏克莱大学的Bill Joy设计。
- nologin ： 表示非交互，不能登录操作系统

3、Shell脚本

1）可以将需要执行的指令存储到文本中，按照顺序依次执行

2）也可以通过Shell语句将文本中的命令编写为条件类型，满足条件后执行

3）简单来说就是可以依赖Shell脚本自动的完成一些维护工作，减轻管理员的工作压力

4）所有的Shell脚本文件添加上执行权限后方可执行，脚本文件的后缀名为 *.sh

5）执行脚本的命令

- **sh 脚本文件**
- **source 脚本文件**
- **./脚本文件（必须添加执行权限）**

二、编写简单Shell脚本

1、编写一个简单脚本文件

1）每行添加指令，执行脚本文件时会自动执行每行命令

2）由于没有添加权限则必须使用 sh、source命令执行

```
[root@localhost~] vim test.sh
cd /boot/
pwd
ls -lh vml*
[root@localhost ~] source test.sh
/boot
-rwxr-xr-x. 1 root root 6.5M 3月  30 2023 vmlinuz-0-rescue-
87964a24466343bdb578677319e20241
-rwxr-xr-x. 1 root root 6.5M 10月 20 2020 vmlinuz-3.10.0-1160.el7.x86_64
```

2、为脚本赋予权限

1）具有执行权限后可以直接“./”方式执行

```
[root@localhost ~] chmod +x test.sh
[root@localhost ~] ll


-rwxr-xr-x. 1 root root   26 1月   2 22:35 test.sh


[root@localhost ~] ./test.sh
/boot
-rwxr-xr-x. 1 root root 6.5M 3月  30 2023 vmlinuz-0-rescue-
87964a24466343bdb578677319e20241
-rwxr-xr-x. 1 root root 6.5M 10月 20 2020 vmlinuz-3.10.0-1160.el7.x86_64
```

3、完善Shell脚本

1）一个完整的Shell脚本需要具有脚本声明（解释器）、注释信息、可执行语句

```
[root@localhost ~] vim test.sh
#!/bin/bash
# This is my test Shell-Script.
cd /boot
echo "当前的目录位于:"
pwd
echo "其中以vml开头的文件包括: "
ls -lh vml*

```

```
[root@localhost ~] ./test.sh
当前的目录位于:
/boot
其中以vml开头的文件包括:
-rwxr-xr-x. 1 root root 6.5M 3月  30 2023 vmlinuz-0-rescue-
87964a24466343bdb578677319e20241
-rwxr-xr-x. 1 root root 6.5M 10月 20 2020 vmlinuz-3.10.0-1160.el7.x86_64
```

三、重定向与管道符操作

1、交互式硬件设备，显示输入输出的结果

1）简单来说可以将用户向系统输入的指令进行查看

2）可以使用命令：**echo $?** 来输出上一条命令的结果

3）其中返回0表示用户使用键盘向系统输入的正确指令

4）返回1表示系统向用户通过显示器输出一个命令或语法错误执行的结果

5）返回>1的任何数表示向用户通过显示器输出一个上一条命令错误的结果

- **标准输入（stdin）是从键盘或其他设备接收用户输入的数据。**
- **标准输出（stdout）是通过屏幕或其他设备向用户输出数据。**
- **标准错误（stderr）是通过屏幕或其他设备报告执行出错信息**

2、重定向

1）如果重定向输出时指定文件不存在这自动创建新文件并写入数据

类型	操作符	用途
重定向输入	<	从指定的文件输入数据
重定向输出	>	将输出结果保存到指定的文件（覆盖原有内容）
	>>	将输出结果追加到指定的文件
重定向错误输出	**2>**	**将错误信息保存到指定的文件（覆盖原有内容）**
	2>>	**将错误信息追加到指定的文件中**
重定向混合输出	**&>**	**将标准输出、标准错误的内容保存到同一个文件中（覆盖原有内容）**
重定向混合输出	**&>>**	**将标准输出、标准错误的内容追加到同一个文件中**

2）重定向实例

```
#重定向输出
[root@localhost ~] hostname > host.txt
```

```
[root@localhost ~] cat host.txt
localhost
[root@localhost ~] hostname >> host.txt

[root@localhost ~] cat host.txt
localhost
localhost

#重定向输入
[root@localhost ~] setenforce 0
[root@localhost ~] echo "123456" > pass.txt
[root@localhost ~] passwd --stdin zhangsan < pass.txt
更改用户 zhangsan 的密码 。
passwd：所有的身份验证令牌已经成功更新。

#标准重定向
[root@localhost ~] vim http_install.sh
#!/bin/bash
tar zxvf httpd-2.2.17.tar.gz -C /usr/src
cd /usr/src/httpd-2.2.17/
./configure --prefix=/usr/local/httpd --enable-so &> /dev/null
make &> /dev/null
make install &> /dev/null
[root@localhost ~] sh http_install.sh
```

3、管道符

1）将左侧的命令输出结果不直接输出到显示器，而是交给右侧的命令继续执行

2）同一行中可以有多个管道符

```
[root@localhost ~] cat /etc/passwd | head -1
root:x:0:0:root:/root:/bin/bash
[root@localhost ~] cat /etc/passwd | head -1 | awk -F ":" '{print $1,$NF}'
root /bin/bash

[root@localhost ~] w | awk '{print $1,$2,$3" --- "$5}'

USER TTY FROM --- IDLE
root tty1 02:27 --- 0.40s
root pts/0 192.168.112.1 --- 5.00s
```

四、变量

1、变量的作用

1）变量的概念来自于数学领域，目前在计算机中能够存储计算结果或者表示不同值得抽象的概念

2）通过给变量名赋值，这个变量中的值可以被计算机随意使用，并且每次使用时均可以表示不同的含义，

3）如：a=10，那么a既可以是年龄，也可以是月份、天数、时间等含义

2、变量的组成

1）变量名：使用固定的名称，可以有系统预设也可以用户手动设置

2）变量值：可以根据实际使用情况、系统环境变化而变化，不一定是固定的

3、变量的类型 *****

1）自定义变量：由用户自行设定，管理、修改和使用

2）环境变量：由系统自动设定，用于设定工作环境，Linux常见的环境变量为：

- $PATH：记录系统所有命令的路径
- $LANG：记录系统语言环境
- $HOME：记录当前用户宿主目录
- $USER：记录当前登录的用户名称

3）位置变量：通过命令行给脚本程序传递参数

4）预定义变量：Bash中内置的一类变量，不可以直接修改

五、变量的使用

1、自定义变量

1）当某个程序或者脚本需要重复性使用一个值时，那么就可以使用变量代替

2）这样在需要进行值修改时，仅需要修改变量中的值即可

3）变量定义的规则，变量名 = 变量值：a=10

- 可以赋予任何类型的值
- 等于号两边不允许出现空格，如果必须添加有空格的值时必须使用双引号

```
[root@localhost ~] name=Mr. Wu
bash: Wu: 未找到命令
[root@localhost ~] name="Mr. Wu"
[root@localhost ~] echo $name
Mr. Wu
```

4）赋值时使用引号 * 3

- 双引号" "：会把引号的内容当成整体来看待，允许通过$符号引用其他变量值
- 单引号' '：会把引号的内容当成整体来看待，禁止引用其他变量值，shell中特殊符号都被视为普通字符
- 反撇号` `： 反撇号和$()一样，引号或括号里的命令会优先执行，执行后的结果就是赋予给变量的值，如果存在嵌套，反撇号不能用

- **read -p：将用户通过键盘输入的数值加入到变量中**

```
[root@localhost ~] echo "Hello $name"
Hello  Mr. Wu
[root@localhost ~] echo 'Hello $name'
Hello  $name
[root@localhost ~] create=`which mkdir`
[root@localhost ~] $create -pv /ceshi/ceshi/ceshi
/usr/bin/mkdir: 已创建目录 "/ceshi"
/usr/bin/mkdir: 已创建目录 "/ceshi/ceshi"
/usr/bin/mkdir: 已创建目录 "/ceshi/ceshi/ceshi"
[root@localhost ~] read -p "请输入你的名字:" name
请输入你的名字:liudehua
[root@localhost ~] echo $name
liudehua
```

2、整数变量的运算

1）格式：expr 变量1 运算符 变量2 [运算符 变量3] ...

2）常用运算符：

- **加法运算：+**
- **减法运算：-**
- **乘法运算：***
- **除法运算：/**
- **求模（取余）运算：%**

```
[root@localhost ~] a=10
[root@localhost ~] b=3
[root@localhost ~] expr $a + $b
13
[root@localhost ~] expr $a - $b
7

[root@localhost ~] expr $a \* $b
30

[root@localhost ~] expr $a / $b
3
[root@localhost ~] expr $a % $b
1
```

```
[root@localhost ~] jieguo=`expr $a \* $a + $b / $b + 100`
[root@localhost ~] echo $jieguo
201
```

3、特殊的变量

1）环境变量

2）由系统提前创建，用于设置用户的工作环境，可以通过env命令查看当前系统下的所有环境变量

4、位置变量

1）表示方法为$n，其中 n 一般表示 1 - 9 之间的任意数字

```
[root@localhost ~] vim sum.sh
#!/bin/bash
SUM=`expr $1 + $2`
echo "$1 + $2 = $SUM"
[root@localhost ~] source sum.sh 10 20
10 + 20 = 30
[root@localhost ~] source sum.sh 35 89
35 + 89 = 124
```

5、预定义变量

1）是Bash环境中已经定义好的一种特殊变量，用户只能使用，不能创新，更不能直接赋值

- $# 命令行中位置参数的个数
- $* 所有位置参数的内容
- **$? 前一条命令执行后的返回状态，返回值为0表示执行正确，非0均表示执行出现异常(127表示command not found（未找到命令）)。**
- $0 当前执行的脚本或程序的名称

```
[root@localhost ~] vim test.sh
#!/bin/bash
ls $*
echo "执行返回结果：$?"
echo "已执行 $0 脚本"
echo "共查看了 $# 个对象"
echo "具体内容包括：$*"
[root@localhost local] sh ~/test.sh /etc/passwd /etc/shadow
/etc/passwd  /etc/shadow
```

```
执行返回结果：0
已执行 /root/test.sh 脚本
共查看了 2 个对象
具体内容包括：/etc/passwd /etc/shadow
[root@localhost local]#
```

六、shell脚本与计划任务

1、将命令行中可以执行正确的命令复制到脚本文件中，确保命令正确

2、添加执行权限，否则系统无法执行

3、添加定时任务，启动服务，测试结果

```
[root@localhost ~] df -Th | awk '{print $6}' > /root/df.txt
[root@localhost ~] cat /root/df.txt
已用%
0%
0%
2%
0%
8%
14%
0%

[root@localhost ~] vim df.sh
#!/bin/bash
df -Th | awk '{print $6}' > /root/df.txt
[root@localhost ~] crontab -e
0 23 * * * source /root/df.sh
[root@localhost ~] systemctl start crond
```

作业：

1. Shell的作用？什么是Shell脚本？
2. shell脚本的作用？
3. **规范脚本的组成**
4. **执行脚本的三种方式**
5. **重定向与管道**
    a. **重定向的分类**
    b. **管道的作用**
6. 变量

什么是变量?

变量的分类

自定义变量、环境变量、位置变量、预定义变量

数值变量的运算 expr

7. 双引号、单引号、反撇号

上机作业:

1、使用脚本每两分钟执行一次 df -Th 命令，仅查看 "\" 目录的使用情况，要求

1）仅显示数字

2）需要显示当前时间以及自己的名字

3）需要提供脚本内容截图、tail -f 三行内容截图

2、编辑一个 apache 安装脚本，运行后直接显示网站内容

1）不显示全部安装过程

2）脚本完成后输出两个内容

1. netstat -anpt | grep httpd 的结果
2. curl 127.0.0.1 的结果（内容改为自己的名字）