2_Engineer01Shell 脚本基础

一 脚本

一个可以执行的文件,运行可以实现某种功能

Linux 内文件颜色为绿色: 表示可执行程序

案例: 编写 hello.sh 脚本

```
~]# vim /root/hello.sh
echo Hello World
~]# chmod +x /root/hello.sh  #赋予执行权限
~]# /root/hello.sh  #绝对路径运行脚本
```

二 规范 Shell 脚本的一般组成

#! 环境声明,以下代码由谁进行翻译执行

注释文本

可执行代码

案例: 输出当前红帽系统的版本信息\当前使用的内核版本\当前系统的主机名

```
~]# vim  /root/hello.sh
#!/bin/bash
echo  Hello  World
cat  /etc/redhat-release    #输出版本信息
uname  -r                   #输出内核版本
hostname                    #输出主机名
```

```
ifconfig  |  head -2
~]# /root/hello.sh    #赋予执行权限后运行
```

三 重定向输出

> : 只收集前面命令的正确输出

2>: 只收集前面命令的错误输出

&>: 正确与错误都收集

单引号 **' ' :** 取消所有特殊字符的意义,原样输出

```
~]# echo '* | & ^ % ***'
~]# echo ', !'
```

反撇号 **` `或$( ) :**将命令的输出,直接参与下一次命令的运行

```
~]# date +%F
2019-07-06
~]# cd /opt/
opt]# mkdir  nsd-`date +%F`
opt]# mkdir  abc-$(date +%F)
opt]# mkdir  $(hostname)-$(date +%F)
```

四 变量

变量:会变化的量, 以不变的名称,存储可以变化的值, 类似于容器

格式: 变量名=存储的值

```
/]# vim  /root/user.sh
```

```
#!/bin/bash
a=tc
useradd  $a  &> /dev/null
echo  用户$a 创建成功
echo  123  |  passwd --stdin  $a  &> /dev/null
echo  用户$a 密码设置成功
```

五 交互式在命令行传递参数给脚本的代码

read -p '屏幕提示信息'

直接产生交互 记录用户在键盘上所有输入 将记录的内容交由变量储存

```
/]# vim /root/user.sh
#!/bin/bash
read  -p  '请输入您要创建的用户名:'    a
 useradd $a  &> /dev/null
 echo “用户$a 创建成功”
 echo 123  |  passwd --stdin $a &> /dev/null
 echo 用户$a 密码设置成功
```

vim 错误解决:产生交换文件(缓存文件)

```
]# vim /opt/4.txt
]# rm -rf /opt/.4.txt.swp  #删除交换文件
```

六 定义/赋值变量

6.1 设置变量时的注意事项

等号两边不要有空格

变量名只能由字母/数字/下划线组成,区分大小写

变量名不能以数字开头,不要使用关键字和特殊字符

若指定的变量名已存在,相当于为此变量重新赋值

6.2 基本格式

引用变量值:$变量名

查看变量值:echo $变量名、echo ${变量名}

```
~]# a=rhel
~]# echo $a
rhel
~]# echo ${a}
rhel
~]# echo $a7
~]# echo ${a}7
rhel7
~]# b=7
~]# echo ${a}${b}
```

6.3 环境变量: 变量名大写　由系统定义并且赋值完成

USER: 当前登录的用户身份

6.4 位置变量: 由系统定义并且赋值完成

$1,$2,...,$n 非交互式 在命令行传递参数给脚本的代码

6.5 预定义变量: 由系统定义并且赋值完成

$# 已加载的位置变量的个数,判断用户是否输入命令行参数

$? 程序退出后的状态值,0 表示正常,其他值异常

6.6 条件测试及选择

格式: [　测试表达式　]　　#每一部分之间都要有空格

常用的测试选项

检查文档状态

-e: 存在即为真

-d: 存在并且必须为目录　才为真

-f: 存在并且必须为文件　才为真

6.7 比较整数大小

-gt: 大于　　-ge: 大于等于

-eq: 等于　　-ne: 不等于

-lt: 小于　　-le: 小于等于

6.8 字符串比对

==: 一致为真　　!=: 不一致为真

6.9 if 双分支处理

```
if  [条件测试];then
   命令序列 1
else
   命令序列 2
fi
```

6.10 判断用户是否输入参数(位置变量)

```
~]# vim  /root/if02.sh
      #!/bin/bash
      if   [  $#  -eq  0  ];then
            echo 您没有输入参数
      else
            echo 您输入了参数
      fi
```

案例: 利用 read 获取用户名进行判断

如果用户存在,则输出 用户已存在

如果用户不存在,则输出 用户不存在

```
~]# vim  /root/if03.sh
      #!/bin/bash
      read  -p  '请输入您要测试的用户名:'    a
```

```
id  $a  &> /dev/null
if  [  $?  -eq  0  ];then
  echo  用户$a 已存在
else
  echo  用户$a 不存在
fi
```

案例:测试用户输入的 IP 地址能否 ping 通,并给出提示信息

```
~]# vim   /root/if04.sh
#!/bin/bash
read -p '请输入您要测试的 IP 地址:' ip
ping -c 2 $ip &> /dev/null       #-c 指定 ping 的次数
if  [  $?  -eq  0  ];then
      echo $ip 可以通信
else
      echo $ip 不可以通信
fi
```

6.11 if 多分支处理

```
if [条件测试 1];then
      命令序列 1
elif [条件测试 2];then
```

```
        命令序列 2
    ...
    else
        命令序列 n
    fi
```

案例: 利用 read 获取用户的成绩

如果大于等于 90,则输出 优秀

如果大于等于 80,则输出 良好

如果大于等于 70,则输出 一般

如果大于等于 60,则输出 合格

以上均不满足,则输出 再牛的肖邦也弹不出哥的悲伤

```
~]# vim  /root/if05.sh
     #!/bin/bash
     read  -p  '请输入您的成绩:'      num
     if   [  $num  -ge  90  ];then
          echo  优秀
     elif [  $num  -ge  80  ];then
          echo  良好
     elif [  $num  -ge  70  ];then
          echo  一般
```

```
elif [  $num  -ge  60  ];then
      echo  合格
else
      echo 再牛的肖邦也弹不出哥的悲伤
fi
```

6.12 列表式循环,解决重复性的操作

```
for  变量   in  数组\集合\...
        do
             命令序列
        done
```

6.13 列表值可以不参与,循环代码的执行

```
~]# vim /root/for01.sh
      #!/bin/bash
      for  a  in zhangsan lisi  wangwu haha xixi
      do
       echo I Love DC
      done
```

6.14 造数工具:制造连续范围的数字 {起始值..结束值}

```
~]# vim  /root/for01.sh
      #!/bin/bash
```

```
for  a  in  {1..99}
do
 sleep 0.5          #暂停 0.5 秒
 echo I Love DC $a
done
~]# /root/for01.sh
```