Shell 脚本基础

脚本: 一个可以执行的文件,实现某种功能

- 需要提前设计、智能化难度大
- 批量执行、效率高
- 方便在后台静悄悄地运行

规范 Shell 脚本的一般组成

- **#! 环境声明(用什么程序去翻译下面命令)**
- **# 注释文本**
- **可执行代码**

1. 编写一个面世问候 /root/hello.sh 脚本
- 显示出一段话“Hello World!!”

```
[root@server0 ~]# vim /root/hello.sh
  #!/bin/bash
  echo hello world
  hostname
  cat /etc/redhat-release
  uname -r
  ifconfig | head -2

[root@server0 ~]# chmod +x /root/hello.sh
[root@server0 ~]# /root/hello.sh                    #绝对路径运行

##############################################
# grep -v '^#' /etc/login.defs      |   grep -v '^$'
```

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
&> : 收集前面命令的正确输出和错误输出

```
[root@server0 ~]# echo 123 > /opt/1.txt

[root@server0 ~]# cat /opt/1.txt   /etc/
[root@server0 ~]# cat /opt/1.txt   /etc/ > /mnt/a.txt
[root@server0 ~]# cat /mnt/a.txt

[root@server0 ~]# cat /opt/1.txt   /etc/ 2> /mnt/a.txt
```

```
[root@server0 ~]# cat /mnt/a.txt

[root@server0 ~]# cat /opt/1.txt   /etc/ &> /mnt/a.txt
[root@server0 ~]# cat /mnt/a.txt
```

##

书写创建用户并设置密码为 123456 的脚本

为了脚本适应多变的环境,方便使用,所以我们要用变量

变量: 以不变应万变 ， 以不变名称存储可以变化的值 容器

为了降低脚本的使用难度,所以我们可以产生交互,用问答的形式

read ：可以读入键盘的输入,赋值给变量

-p：屏幕输出信息

/dev/null: Linux 中黑洞设备,专用于收集不要的输出信息

```
[root@server0 ~]# vim   /root/user.sh
#!/bin/bash
 read   -p    '请输入您要创建的用户名:'          abc
 read   -p    '请输入您要设置的密码:'          pass
 useradd   $abc   &>   /dev/null
 echo   $abc 用户创建成功
 echo   $pass   |   passwd   --stdin    $abc   &>   /dev/null
 echo   $abc 密码成功

[root@server0 ~]# chmod +x /root/user.sh
[root@server0 ~]# /root/user.sh
```

##

变量的使用

- 变量名=变量值
- 方便以固定名称重复使用某个值
- 提高对任务需求、运行环境变化的适应能力

- **设置变量时的注意事项**
- **若指定的变量名已存在,相当于为此变量重新赋值**
- **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**

##

- **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}**

```
[root@server0 ~]# echo $a

[root@server0 ~]# echo rhel$a

[root@server0 ~]# echo $a00

[root@server0 ~]# echo ${a}00
```

##

环境变量　　变量名一般都大写,用来设置用户/系统环境
位置变量　　bash 内置,存储执行脚本时提供的命令行参数
预定义变量　bash 内置,可直接调用的特殊值,不能直接修改
自定义变量　用户自主设置、修改及使用

环境变量　　USER:永远储存当前登陆的用户名
位置变量:　　非交互式为脚本提供参数

执行脚本时　脚本后面的位置,参数分别用变量 1 2 3 4 5 ..... 来储存

```
 [root@server0 ~]# vim   /root/1.sh
    #!/bin/bash
     echo $1
     echo $2
     echo $3
     echo $4
     echo ${10}
     echo ${11}

[root@server0 ~]# /root/1.sh   haha   xixi   hehe   lele          hengheng   dc
tc   dz tz   100 200 300

[root@server0 ~]# vim /root/2.sh
    #!/bin/bash
     cat -n $1

[root@server0 ~]#   /root/2.sh    /etc/redhat-release
```

单引号: 取消特殊字符的意义
$( )和反撇号 ``` : 将命令的输出结果做为参数

```
[root@server0 opt]# rm -rf /opt/*
[root@server0 opt]# cd /opt
[root@server0 opt]# date
[root@server0 opt]# date   +%F
```

```
[root@server0 opt]# mkdir MySQL-$(date +%F)
[root@server0 opt]# ls

[root@server0 opt]# mkdir `hostname`-$(date +%F)
[root@server0 opt]# ls
```

##

预定义变量

- 用来保存脚本程序的执行信息
  - 直接使用这些变量
  - 不能直接为这些变量赋值

$?　程序退出后的状态值,0 表示正常,其他值异常

$#　已加载的 位置变量 的个数

$*　所有位置变量的值

```
[root@server0 opt]# vim  /root/4.sh
 #!/bin/bash
 echo $1
 echo $2
 echo $3
 echo ${10}
 echo $#
 echo $*

[root@server0 opt]# /root/4.sh  1 2 3 4 5 6 7 8 9 10
```

##

条件测试

- 检查文件状态

-e:存在为真

-d:存在并且必须是目录　才为真

-f:存在并且必须是文件　才为真

-r:存在并且必须对其具备读权限 才为真

-w:存在并且必须对其具备写权限 才为真

-x:存在并且必须对其具备执行权限 才为真

- 比较整数大小(带 e 字母的都有 '等于' 二字)

-gt:大于

-ge:大于等于

-eq:等于

-ne:不等于

-lt:小于

-le:小于等于

- 字符串比对
  == ：相等为真
  != ： 不相等为真

##

if 双分支处理

```
if [条件测试];then
   执行命令 xx
 else
   执行命令 yy
fi
```

判断一个用户是否存在,如果存在则输出用户基本信息,如果不存在则创建该用户

```
[root@server0 ~]# vim  /root/4.sh
#!/bin/bash
  read  -p   '请输入您要判断的用户名:'      user   #将用户输入赋值给 user
  id  $user &> /dev/null                    #查看用户基本信息
  if  [ $? -eq 0 ];then   #判断上面命令是否成功,从而判断用户是否存在
   echo  $user 用户已存在
   id  $user
  else
   useradd  $user
   echo  $user 用户已经创建
  fi
```

请用 read 命令实现,读取用户输入的 ip 地址
判断如果本机能够 ping 通该 ip 地址,则输出 "ip 可以通信"
如果本机不能 ping 通该 ip 地址,则输出 "ip 不可以通信"

```
[root@server0 ~]# cat  /root/6.sh
#!/bin/bash
 read  -p  '请输入您要测试的 IP 地址:'     ip
 ping  -c  2  $ip  &>  /dev/null          #ping 测试 2 包结束

 if  [  $?  -eq  0  ];then
  echo  $ip 可以通信

 else
  echo  $ip 不可以通信

 fi
```

##

if 多分支处理

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

##

书写脚本请实现

利用 read 读入学员成绩

大于等于 90 输出优秀

大于等于 80 输出良好

大于等于 60 输出合格

以上条件均不满足

输出 仍需努力

```
[root@server0 ~]# vim  /root/7.sh
#!/bin/bash
 read -p  '请输入您的成绩:'     num

 if [  $num   -ge  90   ];then      #判断是否大于等于 90
      echo 优秀
  elif  [  $num  -ge  80  ];then    #判断是否大于等于 80
    echo 良好
  elif  [  $num  -ge  60  ];then     #判断是否大于等于 60
    echo 合格
 else
  echo 仍需努力
 fi
```

##

在 server0 上创建 /root/foo.sh 脚本

1)当运行/root/foo.sh redhat,输出为 fedora

2)当运行/root/foo.sh fedora,输出为 redhat

3)当没有任何参数或者参数不是 redhat 或者 fedora 时,其错误输出产生以下信息:

/root/foo.sh redhat|fedora

```
[root@server0 ~]# vim  /root/foo.sh
#!/bin/bash
```

```
if  [  $#  -eq  0  ];then                          #判断是否有位置变量
   echo '/root/foo.sh  redhat|fedora'  >&2  #将正确的输出变成错误
   exit 1                                          #脚本运行后返回值
 elif  [  $1  ==  redhat  ];then
   echo fedora
 elif  [  $1  ==  fedora  ];then
   echo redhat
else
   echo '/root/foo.sh  redhat|fedora'  >&2  #将正确的输出变成错误
   exit 1                                          #脚本运行后返回值
fi
```

```
[root@server0 ~]#    /root/foo.sh

[root@server0 ~]#    echo $?
```

##

for 循环结构

```
for  haha  in zhangsan  lisi  wangwu  tianqi
do
      useradd $haha
done
```

```
[root@server0 opt]# vim  /root/for.sh

#!/bin/bash
for  i  in  stu01  stu02  stu03  stu04
do
    useradd $i
    echo $i 创建成功
done
```

##