Shell 语法总结

脚本声明（需要的解释器、作者信息等）

注释信息（步骤、思路、用途、变量含义等）

可执行语句（操作代码）

使用shell变量 可能会变化的值

环境变量 ：变量名通常是大写 系统本身的

位置变量 ： bash内置 执行脚本提供的参数

预订义变量 ：

自定义变量 ：由用户自主设置 修改及使用

1.定义/赋值/查看变量

2.环境/预定义/位置变量的应用

一 .自定义变量

Test=11 把test 赋值 echo $test 可以查看

[root@svr5 ~]# test=11

[root@svr5 ~]# echo $test

11

查看变量时，若变量名称与后面要输出的字符串连在一起，则应该以**{}**将变量名括起来以便区分：

1. [root@svr5 ~]# echo $testRMB             //无法识别变量名test
2. [root@svr5 ~]# echo ${test}RMB             //区分后可以识别
3. 11RMB

若要撤销已有的变量，可使用unset命令：

· [root@svr5 ~]# unset test                 //撤销变量test

环境变量PWD、USER、HOME、SHELL，

预定义变量$0、$$、$?、$#、$\*

位置变量$1、$2、$10

单引号的应用

界定一个完整的字符串，并且可以实现屏蔽特殊符号的功能

反撇号或$()的应用

使用反撇号或$()时，可以将命令执行的标准输出作为字符串存储，因此称为命令替换。

read基本用法

回显功能关闭（stty -echo），

将回显功能恢复（stty echo）。

执行后从会等待并接受用户输入（无任何提示的情况），并赋值给变量 \* 自定义变量值

· #!/bin/bash

· read -p "请输入用户名: " USERNAME             //读取用户名

· stty -echo                                 //关闭回显

· read -p "请输入密码: " PASSWORD             //读取密码

· stty echo                                 //恢复回显

· echo ""                                     //恢复回显后补一个空行

· echo "您的用户名是: $USERNAME"             //确认赋值结果

## Shell中的数值运算

+ - \* / %求模

有以下 几中 使用运算表达式

Expr echo $[]或$(()) let bc小数运算

i=10

Expr $i + 10 - \* / %

i=10

Echo $[i+78] - \* / %

Let pr或$[]、$(())方式只进行运算，并不会改变变量的值；而let命令可以直接对变量值做运算再保存新的值。因此变量X=1234，在执行let运算后的值会变更；另外，let运算操作并不显示结果，但是可以结合echo命令来查看

[root@svr5 ~]# X=1234

[root@svr5 ~]# let y=X+22

[root@svr5 ~]# echo $y

1256

bc非交互式运算

[root@svr5 ~]# echo 'scale=4;12.34+5.678' | bc

18.018

1.条件测试测试

条件测试操作本身不显示出任何信息。测试的条件是否成立主要体现在命令执行后的返回状态（即 $?），所以可以在测试后查看变量$?的值来做出判断，或者结合&&、||等逻辑操作显示出结果（或作其他操作）

== 比较两个字符串是否相同 //查看结果0为对，非0为错

!= 比较两个字符串是否不相同 //查看结果0为对， 1为错

1. # A && B                     //仅当A命令执行成功，才执行B命令

&&，逻辑与给定条件必须都成立，整个测试结果才为真。

检查变量X的值是否大于10，且小于30：

1. [root@svr5 ~]# X=20                     //设置X变量的值为20
2. [root@svr5 ~]# [ $X -gt 10 ] && [ $X -lt 30 ] && echo "YES"
3. YES
4. # A || B                        //仅当A命令执行失败，才执行B命令

||，逻辑或 只要其中一个条件成立，则整个测试结果为真。

只要/tmp/、/var/spool/目录中有一个可写，则条件成立：

1. [root@svr5 ~]# [ -w "/tmp/" ] || [ -w "/var/spool/" ] && echo "OK"
2. OK
3. # A ; B                        //执行A命令后执行B命令，两者没有逻辑关系

-z 字符串为空

[ -z "$var1" ] && echo "空值" || echo "非空值"

非空值

-n 字符串的值不为空 （相当于 ！-z）

[ -n "$1haha" ] && echo "空值" || echo "非空值"

空值

整数值比较

EQ 等于

LT 小于

LE 小于或等于

GE 大于或等于

NE 不等于

-e 判断对象是否存在 不管是目录还是文件

-d 判断对象是否为目录（存在且是目录）

-f 判断对象是否为文件（存在且是文件

-r 判断对象是否可读

-w 判断对象是否可写

-x 判断对象是否具有可执行权限

## 使用if选择结构

**单分支**

1. if 条件测试
2. then 命令序列
3. Fi

**双分支**

1. if 条件测试
2. then 命令序列1
3. else 命令序列2
4. Fi

**多分支**

1. if 条件测试1
2. then 命令序列1
3. elif 条件测试2
4. then 命令序列2
5. else 命令序列n
6. fi

步骤一：检测/media/cdrom目录，若不存在则创建

1）编写脚本如下：

1. [root@svr5 ~]# vim mountdir.sh
2. #!/bin/bash
3. dir="/media/cdrom/"
4. if [ ! -d $dir ]
5. then
6. mkdir -p $dir
7. fi

· [root@svr5 ~]# vim pinghost.sh

· #!/bin/bash

· ping -c 3 -i 0.2 -W 3 $1 &> /dev/null

· if [ $? -eq 0 ] ; then

· echo "Host $1 is up."

· else

· echo "Host $1 is down."

· fi

· · #!/bin/bash

· read -p "请输入积分（0-100）：" JF

· if [ $JF –ge 90 ] ; then

· echo "$JF 分，神功绝世"

· elif [ $JF –ge 80 ] ; then

· echo "$JF 分，登峰造极"

· elif [ $JF –ge 70 ] ; then

· echo "$JF 分，炉火纯青"

· elif [ $JF –ge 60 ] ; then

· echo "$JF 分，略有小成"

· else

· echo "$JF 分，初学乍练"

· fi

## 使用for循环结构

· for 变量名 in 值列表

· do

· 命令序列

· done

练习for循环基本用法

· #!/bin/bash

· for i in {1..10}

· do

·         echo "hello world"

· done

·

#!/bin/bash

· for i in {1..254}

· do

· ping -c 3 -i 0.2 -W 1 192.168.4.$i &> /dev/null

· if [ $? -eq 0 ] ; then

· echo "Host 192.168.4.$i is up."

· else

· echo "Host 192.168.4.$i is down."

· fi

· done

## 使用while循环结构

1. while 条件测试
2. do
3. 命令序列

练习while循环基本用法

· #!/bin/bash

· i=1

· while [ $i -le 5 ]

· do

·         echo "$i"

· done

· #!/bin/bash

· num=$[RANDOM%100+1]

· i=0

· while :

· do

· read -p "随机数1-100,你猜:" guess

· let i++                                    //猜一次，计数器加1，统计猜的次数

· if [ $guess -eq $num ];then

· echo "恭喜，猜对了"

· echo "你猜了$i次"

· exit

· elif [ $guess -gt $num ];then

· echo "猜大了"

· else

· echo "猜小了"

· fi

· #!/bin/bash

· i=1

· while [ $i -le 254 ]

· do

· IP="192.168.4.$i"

· ping -c 3 -i 0.2 -W 1 $IP &> /dev/null

· if [ $? -eq 0 ] ; then

· echo "Host $IP is up."

· else

· echo "Host $IP is down."

· fi

· let i++

· done

## 基于case分支编写脚本

1. case 变量值 in
2. 模式1)
3. 命令序列1 ;;
4. 模式2)
5. 命令序列2 ;;
6. .. ..
7. \*)
8. 默认命令序列
9. esac

·脚本编写参考如下：

#!/bin/bash

· case $1 in

· redhat)

·         echo "fedora";;

· fedora)

·         echo "redhat";;

·     \*)                                             //默认输出脚本用法

· echo "用法: $0 {redhat|fedora}"

· esac

## 使用Shell函数

1. 函数名() {
2. 命令序列
3. .. ..
4. }
5. [root@svr5 ~]# mycd(){                        //定义函数
6. > mkdir /test
7. > cd /test
8. > }
9. [root@svr5 ~]# mycd                            //调用函数
10. [root@svr5 ~]# mycd(){                        //定义函数
11. > mkdir $1
12. > cd $1
13. > }
14. [root@svr5 ~]# mycd /abc                            //调用函数
15. [root@svr5 ~]# mycd /360                            //调用函数

## 中断及退出

过break、continue、exit在Shell脚本中实现中断与退出的功能。

· #!/bin/bash

· for i in {1..5}

· do

·      [ $i -eq 3 ]&& break //这里将break替换为continue，exit分别测试脚本执行效果     echo $i

· done

· echo "Game Over"

· #!/bin/bash

· while read -p "请输入待累加的整数（0表示结束）：" x

· do

· [ $x -eq 0 ] && break

· SUM=$[SUM+x]

· done

· echo "总和是：$SUM"