Shell 脚本

shell简介

- shell的本质是一个命令解释器 位于操作系统和应用程序之间,是操作系统与应用程序之间信息交互的主要接口
- shell主要负责将应用程序输入的命令信息进行解释 然后传递给操作系统 并将操作系统对输入指令的处理结果解释给应用程序 简单来说 shell就是操作系统和应用程序之间命令翻译的工具

shell的常见指令

```
echo $SHELL    # 查看当前系统的使用的shell类型
```

shell入门

```

#!/bin/bash
# 注释信息
echo "测试脚本"
echo "查询测试结果"

# 第一行为指定shell类型 固定写法
# "#"在shell脚本中表示注释信息开头
# 正文内容写linux可执行的指令集合即可 本质上是将多条linux指令放在文件中进行统一的执行 文件后缀为.sh

```

shell的基础语法

```
# 单行注释

:<< !多行注释!  格式为 : << 字符 被注释内容 字符  字符可以为任意字符 为了与正常变量混淆通常使用! 作为多行注释的起始字符和结束字符

#!/bin/bash
# 注释信息
:<<! echo "测试脚本"    # 注意开始符号 与 注释文本之间需要有空格作为间隔 否则多个字符会被当做起始符;;
echo "查询测试结果"
!


```

shell变量

- 定义变量
- 普通变量

```
方式一
变量名=变量值    # "=" 右边不能有特殊符号 变量值必须为一个整体
例: num=10     # 10与"="之间不能有空格 10 之后也不能用
方式二
变量名='变量值' # ''内的内容都会作为变量的值 按照原本内容输出
例: virable= '这是 变量'
方式三
变量名="变量值" # 双引号内有其他变量的情况下 会把变量结果进行拼接 然后赋值
```

- 命令变量

```
方式一:
    变量名=`命令`
方式二
    变量名=$(命令)
以上两种操作 可以将linux命令获取到的结果作为值 赋值给变量
```

- 使用变量

```
非标准写法
$变量名    "$变量名"
${变量名}
标准写法
"${变量名}"

```

- 只读变量

```
在 变量名前添加 readonly 即可
```

- 删除变量

```
在变量前添加 unset 即可
```

shell数组

```
定义数组
数组名=(值1 值2 ...) # 值与值之间以空格隔开
例: arr=(1 2 3)

给数组元素赋值
```

```
数组名[索引] = 值
例: arr[0]=34

获取元素
${数组名[索引]}
例: ${arr[0]}

获取数组长度
${#数组名[*]} 或者 ${#数组名[@]}

```

- 脚本示例

```
#!/bin/bash

arr=(1 2 3 4 5)

# 输出数组
echo "数组arr的值为${arr}"                # 输出为:组arr的值为1 默认输出数组的第一个元素

# 修改索引0处的元素值
arr[0]=5                                  # 修改0号元素的值为5
echo "索引0处的元素值为${arr[0]}"         # 输出为:索引0处的元素值为5

# 获取数组的长度
echo "数组的长度为${#arr[@]}"             # 数组的长度为5
```

shell运算符

算数运算符

```
运算符                    说明                    举例

  +                       加法                  expr $a + $b

  -                       减法                  expr $a - $b

  *                       乘法                  expr $a \* $b     # *在shell中有特殊
                                                含义需要转义
  /                       除法                  expr $a / $b

  %                       取余                  expr $a % $b

  =                       赋值                  a=$b  # 将变量b的值赋值给变量a

 ++/--                    自增/自减              ((a++))/((a--))


```

脚本示例

```
#!/bin/bash

# 声明变量ab
a=10
b=20

# ab相加
c="`expr $a + $b`"
# 输出结果
echo "ab相加的值为 ${c}"


# ab 相减
d="`expr $a - $b`"
echo "ab相减的值为 ${d}"

# ab 相乘# ab 相乘
e="`expr $a \* $b`"
echo "ab相乘的值为 ${e}"

#除以b
f="`expr $a / $b`"
echo "a除以b的值为 ${f}"

# a以b取模
g="`expr $a % $b`"
echo "a以b取模的值为 ${g}"

# 将变量g的值赋值给变量h
h=$g
echo "变量h的值为${h}"

# 变量a 自增
((a++))
echo "变量a的值为10 自增后的值为${a}"

```

- **注意:**

1. 原生的bash 不支持直接使用 + ,- 来直接进行算数运算 需要借助 expr 命令来实现
2. 表达式与运算符之间需要有 空格
3. 完整的表达式 需要 被 ``所包含

字符串运算符

运算符	说明	举例
=	检测两个字符串的值是否相等 相等则返回true	[$a = $b]
!=	检测两个字符串的值是否不相等 不相等则返回true	[$a != $b]
-z	检测字符串长度是否为0 为0则返回true	[-z$a]
-n	检测字符串长度是否不为0 不为0则返回true	[-n"$a"]
$	检测字符字符串是否为ong 不为空则返回true	[$a]

关系运算符

- 关系运算符只支持数字 不支持字符串 除非字符串的值为数字

运算符	说明	举例
-eq	检测两个数是否相等 相等则返回true	[$a -eq $b]
-ne	检测两个数是否不相等 不相等则返回true	[$a -ne $b]
-gt	检测左边数是否大于右边数 是则返回true	[$a -gt $b]
-lt	检测左边数是否小于右边数 是则返回true	[$a -lt $b]
-ge	检测左边数是否大于等于右边数 是则返回true	[$a -ge $b]
-le	检测左边数是否小于等于右边的 是则返回true	[$a -le $b]

布尔运算符

运算符	说明	举例
!	取反运算	[!false] 返回true
-o	或运算,有一个表达式为true则返回true	[$a -lt 20 -o $b -gt 100]
-a	与运算,两个个表达式都为true则返回true	[$a -lt 20 -a $b -gt 100]

逻辑运算符

```
运算符                    说明                    举例
  &&                    逻辑的AND               [[true && true]] 返回true

  ||                    逻辑的OR                [[false || false]] 返回false

```

流程控制语句

if

```
if[条件]
then          # 条件为true则执行
 语句体
fi             # 结束标识


if[条件]
then
 语句体
else
 语句体
fi

if[条件1]
then
 语句体
elif[条件2]
 语句体
else
 语句体
fi


```

选则语句

```
case 值 in
模式1)
语句1
 ;;    # 相当于java中的break
模式2)
语句2
 ;;
esac  # 结束标识


例

```

```
#!/bin/bash
v="test"
case "${v}" in
"case")
    echo "v的值为case"
    ;;
"others")
    echo "v的值为others"
    ;;
```

循环语句

for循环

```
for 变量 in 范围
do
    循环体
done   # 虚幻结束标识

例子
for loop in a b c
do
    echo "${loop}"
done
```

while循环

```
while 条件
do
    循环体
done

例:
a =1
while ["${a}" -le 10]
do
    echo "${a}"
    ((a++))
done
```

函数

```
# 函数定义
# 无参无返回值函数
函数名 (){
    函数体
}
```

```
# 函数调用 写函数名即可
函数名

# 有参无返回值函数
函数名 (){
    echo "第一个参数 $1"
    echo "第一个参数 $2"    # 有多少个写多少个 1 2 表示对应的参数
}

#调用
函数名 参数1 参数2

# 有参有返回值函数
函数名 (){
    echo "第一个参数 $1 和第二个参数 $2"
    return $(($1+$2))
}

#调用
函数名 10 20
echo $?   # 获取返回值


```