

Shell 编程(了解)

1 初识 Shell 脚本

如果我们有一系列经常使用的 Linux 命令，我们可以把它们存储在一个文件中。Shell 可以读取这个文件并执行其中的命令。这样的文件被称为脚本文件。

执行 shell 脚本

要创建一个 shell 脚本，我们要使用任何编辑器比如 vi 在文本文件中编写它，保存的文件最好是.sh 后缀的。

如：vi aa.sh

chmod +x aa.sh 然后 ./aa.sh 或 bash aa.sh 或 sh aa.sh

2 shell 脚本的编写语法

2.1 变量

2.1.1 程序开始与注释

1.程序往往以下面的行开始#!/bin/bash (redhat/suse 下，所以系统默认的 shell 是 bash shell。)

2.注释用#

2.1.2 shell 变量

shell 变量没有数据类型，都是字符串，即使数值也是字符串

创建变量：变量名称=值。如果值有空格则必须用""或者''引用起来

Eg: a= "hello" (=号两边不能有空格)

引用变量：echo \$a 或 echo \${a} 或 echo "\${a}" 注意""""的区别（单引号：消除所有字符的特殊意义；双引号：消除除\$、""、''三种以外其它字符的特殊意义）

1>: #echo

→hello 等同于#echo \${a}
#echo "\${a}"

2>: #echo "hello b\$aa"

→hello b,因为此时把 aa 作为一个整体变量，而且没有定义，所以输出前面的字符串

3>: #echo "hello b\${a}a"

→hello bhelloa

4>: #echo "\${a}a"

→helloa

5>: #echo "\${a}a"

→\${a}a,因为''会消除特殊字符的意义。

6>: #echo ‘\${a}a’ →\\${a}a.
 删除变量: unset 变量名 eg: unset a
 还可以设置变量为只读变量 readonly a=3
 也可以允许用户从键盘输入, 实现程序交互: read a
 echo \$? 用于显示上一条命令的执行结果(0 表示成功, 1 表示失败),
 或者函数返回值。
 转义符 a=What's\ your \\'topic\'\'? (→#a="What's your \'topic\'"?)

#echo \$a
 命令代换 echo `date` (小飘号) 或 echo \$(date) 显示当前系统时间,
 即用系统变量时, 用 echo \$(命令)的形式等价于 echo `命令`
 eg:echo `pwd` →echo \$(pwd)
 表达式计算:
 expr 4 + 5 expr \$a + \$b
 echo `expr 4 + 5` echo `expr \$a + \$b`
 echo \$(expr 4 + 5) echo \$(expr \$a + \$b)
 echo \$((4 + 5)) echo \$((\$a + \$b))
 echo \${4 + 5} echo \${ \$a + \$b }

举例: 写 1.sh 要求读入 1 个目录名, 在当前目录下创建该目录, 并复制 etc 下的 conf 文件到该目录, 统计 etc 下所有目录的数目到 etcdire.txt 中

```
==>#!/bin/bash
#this is my first shell project
read dir
mkdir ${dir}
cp -rf /etc/*.conf ${dir}
ls -l /etc/* | grep ^d | wc -l > etcdire.txt
```

2.1.3 标准变量或环境变量

系统预定义的变量, 一般在/etc/profile 中进行定义

HOME 用户主目录 PATH 文件搜索路径

PWD 用户当前工作目录 PS1、PS2 提示符

UNAME HOSTNAME LOGNAME echo \$PWD

用 echo \$PATH 显示, 用 env 看环境所有变量, 用 env | grep "name" 查找

用 "export" 进行设定或更改为全局变量,

用 unset 变量名 →取消全局变量

的定义

例: 定义本地变量 name="Red Hat Linux" export name 把 name 变为全局变量

sh 进入子 shell echo \${name} 全局变量可以作用于子进程, 而本地变量不

可以。

或直接输出 `export name="Red Hat Linux"`

`bash` 退出子 shell,进入父 shell

设置环境变量: 比如把 `/etc/apache/bin` 目录添加到 `PATH` 中:

1. `#PATH=$PATH:/etc/apache/bin`

2. `vi /etc/profile` 在里面添加 `PATH=$PATH:/etc/apache/bin`

3. `vi ~/.bash_profile` 在里面修改 `PATH` 行,把 `/etc/apache/bin` 加进去,此种方法针对当前用户有效。

2.1.4 特殊变量

`$1,$2...$n` 传入的参数 `$0` 表示 shell 程序名称 → 每一项相当于 `main` 函数中 `argv[i]`

`$#` 传递到脚本的参数列表,或表示参数个数 → 等价于 `main` 函数中的 `argc-1`

`$@` 传入脚本的全部参数 → `argv[1] ---- argv[n-1]`

`$*` 显示脚本全部参数

`$?` 前个命令执行情况 0 成功 1 失败

`$$` 脚本运行的当前进程号

`#!` 运行脚本最后一个命令

举例:

```
vi 1.sh
#!/bin/bash
echo $1
echo $2
echo $3
echo $#
echo $@
echo $*
echo $$
exit 3
./1.sh 1 2 hello "hello world"
echo $?
```

2.2 运算符与表达式

算术运算符(+、-、*、/、%)

逻辑运算符(&&、||、>、==、<、!=)

赋值运算符(=、+=、-=、*=、/=、%=、&=、^=、|=、<<=、>>=)

计算表达式有四种: 1、`$(())` 2、`$[]` 3、`let var=` 4、`expr 4 + 5`

`echo $[$v1 < $v2]` 计算逻辑表达式(用 1 表示 true,用 0 表示 false)

```
echo [$v1<$v2]&&[$v1>$v2] 计算逻辑表达式
v3=2
let v3*=(($v1+$v2))
echo $v3 或 echo ${v3}
```

举例：写 2.sh 要求输入 2 个数 计算 2 个数的和

```
#!/bin/bash
#this is my second shell project
echo "please input the first number:"
read a
echo "please input the second number:"
read b
c=$((a + b))
echo "The result of $a + $b is $c"
```

2.3 Test 命令的用法

```
VAR=2    test $VAR -gt 1    echo $?
```

1) 判断表达式 and or

```
test 表达式 1 -a 表达式 2    两个表达式都为真
test 表达式 1 -o 表达式 2    两个表达式有一个为真
测试是否是闰年：test $((($iYear % 400)) -eq 0 -o $((($iYear % 4)) -eq 0
-a $((($iYear % 100)) -ne 0
```

2) 判断字符串

```
test -n 字符串    字符串的长度非零
-z 字符串长度为零 ==字符串相等 != 字符串不等
a="abc" test $a == "abc" echo $? (0) test $a == "afd" echo $? (1)
```

3) 判断整数

```
test 整数 1 -eq 整数 2    整数相等
-ge 大于等于 -gt 大于 -le 小于等于 -lt 小于 -ne 不等于
```

4) 判断文件

```
test File1 -ef File2    两个文件具有同样的设备号和 i 结点号
test File1 -nt File2    文件 1 比文件 2 新
test File1 -ot File2    文件 1 比文件 2 旧
test -d File    文件存在并且是目录
test -e File    文件存在
test -f File    文件存在并且是正规文件
test -r File    文件存在并且可读
test -w File    文件存在并且可写
test -x File    文件存在并且可执行
```

举例:

```
a=2
test $a -ge 3
echo $?
```

8. 数组

定义 1: `a=(1 2 3 4 5)` 下标从 0 开始 各个数据之间用空格隔开

定义 2: `a[0]=1;a[1]=2;a[2]=3`

定义 3: `a=([1]=1 [2]=2)`

引用 `${a[1]}`

`${#a[@]}` 数组长度 \rightarrow `${#a[*]}`

`${a[@]:1:2}` 从下标 1 开始后面显示 2 个

`${a[@]}` 或 `${a[*]}` 输出数组的所有元素

例子

```
#a=(2 5 7 10)
```

```
#echo ${a[2]} //输出下标为 2 的数据
```

```
#echo ${#a[*]} //输出数组的长度
```

```
#echo ${a[@]:2} 截取下标从 2 到最后
```

```
#echo ${a[@]:1:2} //截取从下标 1 开始后面连续 2
```

```
#!/bin/bash
```

```
a=(3 10 6 5 9 2 8 1 4 7)
```

```
x=0
```

```
while [ $x -lt ${#a[*]} ]
```

```
do
```

```
    echo ${a[$x]} //或者 echo ${a[x]}
```

```
    x=$((x + 1))
```

```
done
```

```
#!/bin/bash
```

```
a=(3 10 6 5 9 2 8 1 4 7)
```

```
i=0
```

```
while (( i<10 )) //类似 C 语言的写法
```

```
do
```

```
    echo ${a[i]}
```

```
    i=$((i+1))
```

```
done
```

2.4 If 语句

`if [condition] then action fi` 只有当 `condition` 为真时, 该语句才执行操作, 否则不执行操作, 并继续执行 “fi” 之后的任何行。

```
if [ condition ] then action elif [ condition2 ] then action2 ... elif
[ condition3 ] then else actionx fi
```

在使用时，将“if”和“then”放在不同行，如同行放置，则 if 语句必须要；结束

举例：用参数传 1 个文件名，该文件如果是文件并且可读可写就显示该文件，如果是目录就进入该目录，并判断 ls.sh 存在否，如果不存在就建立 1 个 ls.sh 的文件并运行该文件。

该文件的内容是 ls -li /etc > etc.list

```
#!/bin/bash
```

```
if [ -f $1 -a -r $1 -a -w $1 ]    //判断是普通文件并可读可写 → if test -f
$1 -a -r $1 -a -w $1
```

```
then
```

```
    cat $1    //显示文件内容
```

```
elif [ -d $1 ]    //否则如果是目录
```

```
then
```

```
    cd $1    //进入目录
```

```
    if [ -e ls.sh ]    //如果 ls.sh 该文件存在
```

```
    then
```

```
        chmod +x ls.sh    //赋予可执行的权限
```

```
        ./ls.sh    //执行
```

```
    else
```

```
        touch ls.sh    //如果不存在则创建 ls.sh
```

```
        echo "#!/bin/bash" >> ls.sh    //将程序写入 ls.sh 中保存
```

```
        echo "ls -li /etc > etc.list" >> ls.sh    //将要执行的命令写入
```

ls.sh 中保存

```
        chmod +x ls.sh    //赋予可执行的权限
```

```
        ./ls.sh
```

```
    fi
```

```
fi
```

2.5 Case 语句

case 常用的语法形式如下：

```
case $1 in
    "1")
        echo you inputed "1"
        ;;
    "2")
        echo you inputed "2"
        ;;
```

```
*)
    echo you inputed other number
;;
esac
例子 1
echo "Is it morning? Please answer yes or no."
read YES_OR_NO
case "$YES_OR_NO" in
    yes|y|Yes|YES)
        echo "Good Morning!";;
    [nN]*) /* 表示 n 或 N 开头的任意字段 */
        echo "Good Afternoon!";;
    *)
        echo "Sorry, $YES_OR_NO not recognized. Enter yes or no."
        exit 1;;
esac
```

例子 2: 编写一个加减乘除取模计算器

```
echo "please input the first number:"
read a
echo "please input the second number:"
read b
echo "please input your operator:"
read c
case $c in
    "+")
        echo "the result of $a + $b is $((a + b))"
        ;;
    "-")
        echo "the result of $a - $b is $((a - b))"
        ;;
    "*")
        echo "the result of $a * $b is $((a * b))"
        ;;
    "/")
        echo "the result of $a / $b is $((a / b))"
        ;;
    *)
        echo "no true operator!"
        ;;
esac
```

2.6 for 循环

例子 1:

```
for x in one two three four
do
    echo number $x
done
```

例子 2:

```
for x in /etc/????.???? /var/lo* /home/* ${PATH} //列举
do
    echo $x
done
```

例子 3: /etc/r*中的文件和目录

```
for myfile in /etc/r*
do
    if [ -d "$myfile" ]
    then
        echo "$myfile(dir)"
    else
        echo "$myfile"
    fi
done
```

例子 4:

```
for x in /var/log/*
do
    echo `basename $x` is a file living in /var/log
done
```

例子 5: //冒泡排序

```
#!/bin/bash
a=(3 10 6 5 9 2 8 1 4 7)
for (( i=1; i<10; i++ ))
do
    for (( j=0; j<10-i; j++ ))
    do
        if [ ${a[j]} -gt ${a[j+1]} ]
        then
            temp=${a[j]}
            a[j]=${a[j+1]} //或者 a[j]=${a[$((j+1))]}
            a[j+1]=$temp
        fi
    done
done
```



```
for (( i=0; i<10; i++ ))
do
    echo ${a[i]}
done
```

2.7 While 语句

```
myvar=0
while [ $myvar -ne 10 ]
do
    echo $myvar
    myvar=$((myvar+1))
done
```

举例：

```
#!/bin/bash
#this is my first shell project
loopcount=0
result=0
while [ $loopcount -lt 100 ]
do
    loopcount=$((loopcount + 1))
    result=$((loopcount + $result))
done
echo "The result of \'1+2+3+...+100\' is $result"
```

2.9 until 语句

```
myvar=0
until [ $myvar -eq 10 ]
do
    echo $myvar
    myvar=$((myvar+1))
done
```

3 Shell 函数

```
函数名(){    命令 1    ... ... }
function 函数名() {    ... ... }
#declare a function named hello
function hello()
{
```

```
    echo "Hello,$1 today is `date`"
    return 11
}
echo "now going to the function hello"
hello "I LOVE CHINA"
echo $?
echo "back from the function"
```

例 2:

实现两个数相加

C 语言实现:

```
#include <stdio.h>
int add(int a, int b)
{
    return a + b;
}
int main()
{
    int a = 10;
    int b = 20;
    int c = add(a, b);
    printf("%d\n", c);
    return 0;
}
```

Shell 实现:

```
#!/bin/bash
function add()
{
    return $(( $1+$2 ))
}
a=10
b=20
add a b
echo $?
```