Shell 编程

基本格式

Shell 脚本的文件名后缀通常是.sh （当然你也可以使用其他后缀或者没有后缀，.sh 是为了规范）

程序编写格式：

[java] view plain copy

```
#!/bin/bash
#  注释使用#号
```

代码示例：

[java] view plain copy

```
//使用 vi 编辑器编写 shell 脚本（a.sh 不存在则会新建）
vi a.sh
```

进入 vi 编辑模式后编写执行代码

[java] view plain copy

```
//固定格式,记住就可以了
#!/bin/bash
//执行的代码
echo Hello World
```

赋予权限并执行：

[java] view plain copy

```
//赋予可执行权限
chmod +x a.sh
//执行(调用/bin/bash 执行 a.sh 脚本)
./a.sh
```

执行结果：

```
[root@master01 opt]# chmod +x a.sh
[root@master01 opt]# ./a.sh
Hello World
[root@master01 opt]#
```

下面是几种运行情况：

[java] view plain copy

```
1.  a.sh
```

这样的话需要保证脚本具有执行权限并且在环境变量 PATH 中有(.) ,这样在执行的时候会先从当前目录查找。

```
export PATH=.:$PATH
```

[java] view plain copy

```
1.  ./a.sh
```

只要保证这个脚本具有执行权限即可

[java] view plain copy

```
1.  /usr/local/a.sh
```

只要保证这个脚本具有执行权限即可

[java] view plain copy

```
1.  bash a.sh
```

直接可以执行 ,甚至这个脚本文件中的第一行都可以不引入/bin/bash ,它是将 hello.sh 作为参数传给 bash 命令来执行的。

[java] view plain copy

```
1.  bash -x /path/to/aa.sh
```

bash 的单步执行

[java] view plain copy

```
1. bash -n /path/to/aa.sh
```

bash 语法检查

变量

变量不需要声明，初始化不需要指定类型

变量命名

1、只能使用数字，字母和下划线，且不能以数字开头

2、变量名区分大小写

3、建议命令要通俗易懂

注意：变量赋值是通过等号(=)进行赋值,在变量、等号和值之间不能出现空格。

显示变量值使用 echo 命令(类似于 **Java** 中的 system.out) ，加上$变量名，也可以使用${变量名}

例如：

[java] view plain copy

```
1. echo $JAVA_HOME
2. echo ${JAVA_HOME}
```

变量的申明和使用：

```
[root@master01 opt]# num=10
[root@master01 opt]# echo $num
10
[root@master01 opt]#
```

变量分类：

Shell 变量有这几类：本地变量、环境变量、局部变量、位置变量、特殊变量。

本地变量：

1. 只对当前 shell 进程有效的，对当前进程的子进程和其它 shell 进程无效。
2. 定义：VAR_NAME=VALUE
3. 变量引用：${VAR_NAME} 或者 $VAR_NAME
4. 取消变量：unset VAR_NAME
5. 相当于 java 中的私有变量(private)，只能当前类使用，子类和其他类都无法使用。

比如在一个 bash 命令窗口下再使用 bash，则变成了子进程，本地变量不会被这个子进程所访问。

```
[root@master01 opt]# num=10
[root@master01 opt]# echo $num
10
[root@master01 opt]# bash
[root@master01 opt]# echo $num

[root@master01 opt]#
```

环境变量：

自定义的环境变量对当前 shell 进程及其子 shell 进程有效，对其它的 shell 进程无效

定义：export VAR_NAME=VALUE

对所有 shell 进程都有效需要配置到配置文件中

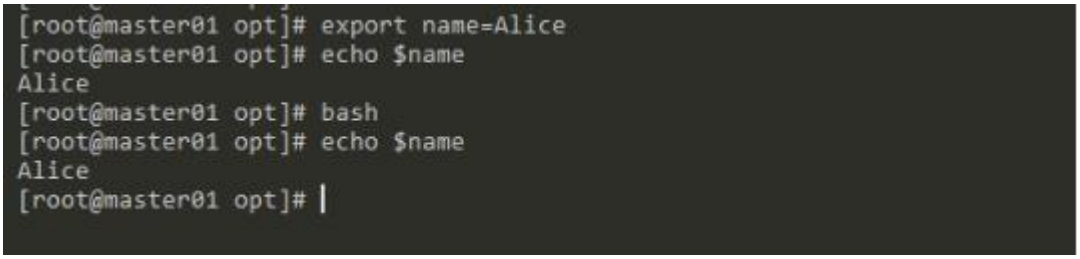
[java] view plain copy

```
1. vi /etc/profile
2. source /etc/profile
```

相当于 java 中的 protected 修饰符,对当前类，子孙类，以及同一个包下面可以共用。

和 windows 中的环境变量比较类似

自定义的环境变量：



局部变量：

1. 在函数中调用，函数执行结束，变量就会消失
2. 对 shell 脚本中某代码片段有效
3. 定义：local VAR_NAME=VALUE
4. 相当于 java 代码中某一个方法中定义的局部变量，只对这个方法有效。

位置变量：

比如脚本中的参数：

$0：脚本自身

$1：脚本的第一个参数

$2：脚本的第二个参数

相当于 java 中 main 函数中的 args 参数，可以获取外部参数。

编写脚本：

```
[root@master01 opt]# cat a.sh
#!/bin/bash

echo $0
echo $1
echo $2
```

执行示例：

```
[root@master01 opt]# ./a.sh 4 5
./a.sh
4
5
[root@master01 opt]#
```

特殊变量：

$?：接收上一条命令的返回状态码

返回状态码在 0-255 之间

$#：参数个数

$*：或者$@：所有的参数

$$：获取当前 shell 的进程号（PID）(可以实现脚本自杀)(或者使用 exit 命令直接退出也可以使用 exit [num])

引号

Shell 编程中有三类引号：单引号、双引号、反引号。

''单引号不解析变量

[java] view plain copy

```
1.  echo '$name'
```

""双引号会解析变量

[java] view plain copy

```
1.  echo "$name"
```

``反引号是执行并引用一个命令的执行结果，类似于$(...)

[java] view plain copy

```
1.  echo `$name`
```

示例：

```
[root@master01 opt]# num=20
[root@master01 opt]# echo '$num'
$num
[root@master01 opt]# echo "$num"
20
[root@master01 opt]# echo `echo $num`
20
[root@master01 opt]#
```

循环

for 循环

通过使用一个变量去遍历给定列表中的每个元素，在每次变量赋值时执行一次循环体，直至赋值完成所有元素退出循环

格式 1

[java] view plain copy

```
for ((i=0;i<10;i++))
do
  ...
Done
```

格式 2

[java] view plain copy

```
for i in 0 1 2 3 4 5 6 7 8 9
do
...
Done
```

格式 3

[java] view plain copy

```
1. for i in {0..9}
2. do
3.   ...
4. done
```

注意：for i in {0..9} 等于 for i in {0..9..1}，第三个参数为跨步。

例如：

{0..9..2} 表示 0,2,4,6,8

while 循环

适用于循环次数未知，或不便用 for 直接生成较大的列表时

格式：

[java] view plain copy

```
1. while 测试条件
2. do
3. 循环体
4. done
```

如果**测试**条件为“真”，则进入循环，测试条件为假，则退出循环。

```
[root@master01 opt]# cat a.sh
#!/bin/bash

num=0
while [ $num -lt 10 ]
do
echo $num
let num=$num+1
done
[root@master01 opt]#
```

打印结果为 0~9.

循环控制

循环控制命令——break

break 命令是在处理过程中跳出循环的一种简单方法 ,可以使用 break 命令退出任何类型的循环，包括 while 循环和 for 循环

循环控制命令——continue

continue 命令是一种提前停止循环内命令，而不完全终止循环的方法，这就需要在循环内设置 shell 不执行命令的条件

条件

bash 条件测试

格式：

[java] view plain copy

1. `test EXPR`
2. `[ EXPR ]`：注意中括号和表达式之间的空格

整型测试：

-gt：大于：

-lt：小于

-ge：大于等于

-le：小于等于

-eq：等于

-ne：不等于

例如[$num1 -gt $num2]或者 test $num1 -gt $num2

字符串测试：

=：等于，例如判断变量是否为空［"$str" = ""］或者[-z $str]

!=：不等于

示例：

```
num1=4
num2=5
str1=Alice
str2=Bob
if [ $num1 -gt $num2 ]
then
echo num1 large than num2
else
echo num1 lower than num2
fi

if [ -z $str1 ]
then
echo str1 is empty
else
echo str is not empty
fi
```

打印结果：

```
[root@master01 opt]# ./a.sh
num1 lower than num2
str is not empty
[root@master01 opt]#
```

判断

if 判断:

单分支

[java] view plain copy

```
1.  if 测试条件;then
2.  选择分支
3.  fi
```

双分支

[java] view plain copy

```
1.  if 测试条件
2.   then
3.    选择分支 1
4.    else
5.    选择分支 2
6.  fi
```

多分支

[java] view plain copy

```
if 条件1; then
            分支1
elif 条件2; then
            分支2
elif 条件3; then
            分支3
           ...
else
            分支n
i
```

双分支示例：

```
if [ -z $str1 ]
then
echo str1 is empty
else
echo str is not empty
fi
```

Case 判断

有多个测试条件时，case 语句会使得语法结构更清晰

格式：

[java] view plain copy

```
case 变量引用 in
                PATTERN1)
                        分支1
                        ;;
                PATTERN2)
                        分支2
                        ;;
                ...
                *)
```

```
10.                         分支 n
11.                         ;;
12.                 esac
```

PATTERN :类同于文件名通配机制，但支持使用|表示或者

a|b：a 或者 b

*：匹配任意长度的任意字符

?：匹配任意单个字符

[a-z]：指定范围内的任意单个字符

示例：

```
[root@master01 opt]# cat a.sh
#!/bin/bash

num=10
case $num in
        1)
            echo 1
                ;;
        2)
            echo 2
                ;;
        10)
            echo 10
                ;;
        *)
            echo somethin else
                ;;
esac
```

算术运算

[java] view plain copy

```
1.  let varName=算术表达式
```

```
2.
3.  varName=$[算术表达式]
4.
5.  varName=$((算术表达式))
6.
7.  varName=`expr $num1 + $num2`
```

使用这种格式要注意两个数字和+号中间要有空格。

示例：

```
[root@master01 opt]# cat b.sh
#!/bin/bash

num=1
let num=$num+1
num=$[ $num+1 ]
num=$(($num+1))
num=`expr $num + 1`
echo $num
[root@master01 opt]#
```

逻辑运算符

if [条件 A && 条件 B] 在 shell 中怎么写?

if [条件 A && 条件 B];then 是不对的

解决方法：

(1)需要用到 shell 中的逻辑操作符

-a 与

-o 或

！ 非

如 if [条件 A -a 条件 B]

```
[root@master01 opt]# cat b.sh
#!/bin/bash

num1=10
num2=20
num3=15
if [ $num1 -lt $num3 -a $num2 -gt $num3 ]
then
        echo num is between 10 and 20
else
        echo somthing else
fi
[root@master01 opt]#
```

(2)if [条件 A] && [条件 B]

(3)if((A&&B))

(4)if [[A&&B]]

```
[root@master01 opt]# cat b.sh
#!/bin/bash

num1=10
num2=20
num3=15
if [[ $num1 -lt $num3 && $num2 -gt $num3 ]]
then
        echo num is between 10 and 20
else
        echo somthing else
fi
[root@master01 opt]#
```

自定义函数

格式：

[java] view plain copy

```
1. function 函数名(){
2. ...
3. }
```

1. 引用自定义函数文件时，使用 source func.sh
2. 有利于代码的重用性
3. 函数传递参数（可以使用类似于 Java 中的 args，args[1]代表 Shell 中的$1）
4. 函数的返回值，只能是数字

```
[root@master01 opt]# cat func.sh
#!/bin/bash

function func(){
        echo this is a function
}

func
[root@master01 opt]#
```

函数的调用：

```
[root@master01 opt]# cat b.sh
#!/bin/bash

source func.sh
func
[root@master01 opt]#
```

read

read 命令接收标准输入（键盘）的输入，或者其他文件描述符的输入。得到输入后，read 命令将数据放入一个标准变量中。

格式

[java] view plain copy

```
1. read VAR_NAME
```

read 如果后面不指定变量，那么 read 命令会将接收到的数据放置在环境变量 REPLY 中

[java] view plain copy

```
1. #表示输入时的提示字符串：
2. read -p "Enter your name:" VAR_NAME
```

```
[root@master01 test]# read -p "Enter you name:" name
Enter you name:
```

[java] view plain copy

```
1. # -t 表示输入等待的时间
2. read -t 5 -p "enter your name:" VAR_NAME
```

[java] view plain copy

```
1. # -s 表示安全输入，键入密码时不会显示
2. read  -s  -p "Enter your password: " pass
```

declare

用来限定变量的属性

-r 只读

-i 整数：某些算术计算允许在被声明为整数的变量中完成，而不需要特别使用 expr 或 let 来完成。

-a 数组

示例：

只读

```
[root@master01 test]# num=10
[root@master01 test]# declare -r num
[root@master01 test]# num=20
bash: num: readonly variable
[root@master01 test]#
```

整数

```
[root@master01 test]# num4=10
[root@master01 test]# num4=$num4+1
[root@master01 test]# echo $num4
10+1
[root@master01 test]# declare -i num5
[root@master01 test]# num5=10
[root@master01 test]# num5=$num5+1
[root@master01 test]# echo $num5
11
[root@master01 test]#
```

数组

```
[root@master01 test]# declare -a arr
[root@master01 test]# arr=(1 2 3 4 5)
[root@master01 test]# echo ${arr[*]}
1 2 3 4 5
[root@master01 test]#
```

字符串操作

获取长度：

[java] view plain copy

```
1.  ${#VAR_NAME}
```

字符串截取

[java] view plain copy

```
1.  ${variable:offset:length}或者${variable:offset}
```

取尾部的指定个数的字符

[java] view plain copy

```
1.  ${variable: -length}：注意冒号后面有空格
```

大小写转换

小-->大：

[java] view plain copy

```
1.  ${variable^^}
```

大-->小：

[java] view plain copy

```
1.  ${variable,,}
```

示例：

```
[root@master01 test]# name=Alice
[root@master01 test]# echo ${#name}
5
[root@master01 test]# echo ${name:0:3}
Ali
[root@master01 test]# echo ${name^^}
ALICE
[root@master01 test]# echo ${name,,}
alice
[root@master01 test]#
```

数组

定义：declare -a：表示定义普通数组

特点

1. 支持稀疏格式
2. 仅支持一维数组

数组赋值方式

1. 一次对一个元素赋值 a[0]=$RANDOM
2. 一次对多个元素赋值 a=(a b c d)

按索引进行赋值 a=([0]=a [3]=b [1]=c)

使用 read 命令 read -a ARRAY_NAME 查看元素

[java] view plain copy

1. `${ARRAY[index]}`：查看数组指定角标的元素
2. `${ARRAY}`：查看数组的第一个元素
3. `${ARRAY[*]}`或者`${ARRAY[@]}`：查看数组的所有元素

获取数组的长度

[java] view plain copy

```
1. ${#ARRAY[*]}
2. ${#ARRAY[@]}
```

获取数组内元素的长度

[java] view plain copy

```
1. ${#ARRAY[0]}
```

注意：${#ARRAY[0]}表示获取数组中的第一个元素的长度，等于${#ARRAY}

从数组中获取某一片段之内的元素（操作类似于字符串操作）

格式：

[java] view plain copy

```
1. ${ARRAY[@]:offset:length}
```

1. offset：偏移的元素个数
2. length：取出的元素的个数
3. ${ARRAY[@]:offset:length}：取出偏移量后的指定个数的元素
4. ${ARRAY[@]:offset}：取出数组中偏移量后的所有元素

数组删除元素：

[java] view plain copy

```
1. unset ARRAY[index]
```

示例：

```
[root@master01 test]# declare -a arr2
[root@master01 test]# arr2=(a b c d)
[root@master01 test]# arr2[1]=x
[root@master01 test]# echo ${arr2[*]}
a x c d
[root@master01 test]# echo ${#arr2[*]}
4
[root@master01 test]# echo ${#arr2[1]}
1
[root@master01 test]# echo ${arr2[1]}
x
[root@master01 test]#
```

其他命令

date

显示当前时间

1. 格式化输出 +%Y-%m-%d
2. 格式%s 表示自 1970-01-01 00:00:00 以来的秒数
3. 指定时间输出 --date='2009-01-01 11:11:11'
4. 指定时间输出 --date='3 days ago' （3 天之前，3 天之后可以用-3）

示例：

```
[root@master01 opt]# echo `date +%Y-%m-%d-%H:%M:%S`
2016-04-18-22:11:05
[root@master01 opt]# echo `date +%s`
1460988671
[root@master01 opt]# echo `date --date='2009-01-01 11:11:11'`
Thu Jan 1 11:11:11 CST 2009
[root@master01 opt]# echo `date --date='3 days ago'`
Fri Apr 15 22:11:22 CST 2016
[root@master01 opt]#
```

后台运行脚本

在脚本后面加一个&

[java] view plain copy

```
1. test.sh &
```

这样的话虽然可以在后台运行，但是当用户注销(logout)或者网络断开时,终端会收到

Linux HUP 信号(hangup)信号从而关闭其所有子进程

nohup 命令

不挂断的运行命令，忽略所有挂断(hangup)信号

[java] view plain copy

```
1. nohup test.sh &
```

1. nohup 会忽略进程的 hangup 挂断信号，所以关闭当前会话窗口不会停止这个进程的执行。
2. nohup 会在当前执行的目录生成一个 nohup.out 日志文件

标准输入、输出、错误、重定向

标准输入、输出、错误可以使用文件描述符 0、1、2 引用

使用重定向可以把信息重定向到其他位置

1. ls >file 或者 ls 1>file (ls >>file)
2. lk 2>file(lk 是一个错误命令)

3. ls >file 2>&1
4. ls > /dev/null(把输出信息重定向到无底洞)

例子：

[java] view plain copy

```
1. command  >/dev/null 2>&1
```

Crontab 定时器

1. linux 下的定时任务
2. 编辑使用 crontab -e
3. 一共 6 列，分别是:分 时 日 月 周 命令

示例：（表示每隔分钟执行一次 bash /opt/date.sh 命令）

```
* * * * * bash /opt/date.sh
```

查看使用 crontab -l

```
[root@master01 opt]# crontab -l
* * * * * bash /opt/date.sh
[root@master01 opt]#
```

删除任务 crontab -r

```
[root@master01 opt]# crontab -l
* * * * * bash /opt/date.sh
[root@master01 opt]# crontab -r
[root@master01 opt]# crontab -l
no crontab for root
[root@master01 opt]#
```

查看 crontab 执行日志

[java] view plain copy

```
1. tail -f /var/log/cron
```

必须打开 rsyslog 服务 cron 文件中才会有执行日志(service rsyslog status)

[java] view plain copy

```
1. tail -f /var/spool/mail/root(查看 crontab 最近的执行情况)
```

查看 cron 服务状态

[java] view plain copy

```
1. service crond status
```

启动 cron 服务

[java] view plain copy

```
1. service crond start
```

小结及示例：

基本格式：

*　　*　　*　　*　　*　　command

分　时　日　月　周　命令

第 1 列表示分钟 1～59 每分钟用*或者 */1 表示

第 2 列表示小时 1～23（0 表示 0 点）

第 3 列表示日期 1～31

第 4 列表示月份 1～12

第 5 列标识号星期 0～6（0 表示星期天）

第 6 列要运行的命令

crontab 文件的一些例子：

30 21 * * * /usr/local/etc/rc.d/lighttpd restart

上面的例子表示每晚的 21:30 重启 apache。

45 4 1,10,22 * * /usr/local/etc/rc.d/lighttpd restart

上面的例子表示每月 1、10、22 日的 4 : 45 重启 apache。

10 1 * * 6,0 /usr/local/etc/rc.d/lighttpd restart

上面的例子表示每周六、周日的 1 : 10 重启 apache。

0,30 18-23 * * * /usr/local/etc/rc.d/lighttpd restart

上面的例子表示在每天 18 : 00 至 23 : 00 之间每隔 30 分钟重启 apache。

0 23 * * 6 /usr/local/etc/rc.d/lighttpd restart

上面的例子表示每星期六的 11 : 00 pm 重启 apache。

* */1 * * * /usr/local/etc/rc.d/lighttpd restart

每一小时重启 apache

* 23-7/1 * * * /usr/local/etc/rc.d/lighttpd restart

晚上 11 点到早上 7 点之间，每隔一小时重启 apache

0 11 4 * mon-wed /usr/local/etc/rc.d/lighttpd restart

每月的 4 号与每周一到周三的 11 点重启 apache

0 4 1 jan * /usr/local/etc/rc.d/lighttpd restart

一月一号的 4 点重启 apache

ps 和 jps

1. ps：用来显示进程的相关信息
2. ps 显示当前 shell 启动的所有进程
3. ps -e 显示系统中所有进程
4. ps -ef|grep java

5. jps 类似 linux 的 ps 命令,不同的是 ps 是用来显示所有进程 ,而 jps 只显示 java 进程,准确的说是显示当前用户已启动的部分 java 进程信息，信息包括进程号和简短的进程 command。

问题 :某个 java 进程已经启动 ,用 jps 却显示不了该进程进程号 ,使用 ps -ef|grep java 却可以看到？

java 程序启动后，默认（请注意是默认）会在/tmp/hsperfdata_userName 目录下以该进程的 id 为文件名新建文件，并在该文件中存储 jvm 运行的相关信息，其中的 userName 为当前的用户名，/tmp/hsperfdata_userName 目录会存放该用户所有已经启动的 java 进程信息。而 jps、jconsole、jvisualvm 等工具的数据来源就是这个文件（/tmp/hsperfdata_userName/pid)。所以当该文件不存在或是无法读取时就会出现 jps 无法查看该进程号。

原因：1，磁盘读写、目录权限问题。2，临时文件丢失，被删除或是定期清理。3，java 进程信息文件存储地址被设置，不在/tmp 目录下

登录 Shell 和交互 shell

交互式的：顾名思义，这种 shell 中的命令时由用户从键盘交互式地输入的，运行的结果也能够输出到终端显示给用户看。

非交互式的：这种 shell 可能由某些自动化过程启动，不能直接从请求用户的输入，也不能直接输出结果给终端用户看。输出最好写到文件。比如使用 Shell 脚本。

登录式：意思是这种是在某用户由/bin/login 登陆进系统后启动的 shell，跟这个用户绑定。这个 shell 是用户登陆后启动的第一个进程。login 进程在启动 shell 时传递第 0 个参数指明 shell 的名字，该参数第一个字符为"-"，指明这是一个 login shell。比如对 bash 而言，启动参数为"-bash"。

非登录式：不需 login 而由某些程序启动的 shell。传递给 shell 的参数，是没有'-'前缀的。还以 Bash 为例，当以非 login 方式启动时，它会调用~/.bashrc，随后~/.bashrc 中调用/etc/bashrc，最后/etc/bashrc 调用所有/etc/profile.d 目录下的脚本。

一旦打开一个交互式 login shell，或者以--login 选项登录的非交互式 shell，都会首先加载并执行/etc/profile 中的命令，然后再依次加载~/.bash_profile, ~/.bash_login, 和~/.profile 中的命令。

当 bash 以 login shell 启动时，它会执行/etc/profile 中的命令，然后/etc/profile 调用/etc/profile.d 目录下的所有脚本；然后执行~/.bash_profile，~/.bash_profile 调用~/.bashrc，最后~/.bashrc 又调用/etc/bashrc。要识别一个 shell 是否为 login shell，只需在该 shell 下执行 echo $0。

注意： /etc/profile 中的设置只对 Login Shell 生效，而 crontab 运行脚本的 shell 环境是 non-login 的，不会加载/etc/profile 的设置。

Shell 应用示例

根据时间创建文件夹

需求：创建 10 个目录，目录名称以当天时间开头，后面拼上目录编码

例如：1970-01-01_1

```
[root@master01 test]# cat date.sh
#!/bin/bash

for i in {1..10}
do
        date=`date +%Y-%m-%d`
        echo ${date}_$i
done
[root@master01 test]#
```

编写脚本 monitor.sh

持续观察服务器每天的运行状态，需要结合 shell 脚本程序和计划任务，定期跟踪记录不同时段服务器的 cpu 负载，内存，交换空间，磁盘使用量等信息

[java] view plain copy

```
1. #!/bin/bash
2. #this is the second script!
```

```
day_time=`date+"%F %R"`
cpu_test=`uptime`
mem_test=`free -m | grep "mem" | awk '{print $2}'`
swap_test=`free -m | grep "mem" | awk '{print $4}'`
disk_test=`df -hT`
user_test=`last -n 10`
echo "now is $day_time"
echo "%cpu is $cpu_test"
echo "Numbet of Mem size(MB) is $mem_test"
echo "Number of swap size(MB) is $swap_test"
echo "the disk shiyong qingkuang is $disk_test"
echo "the users login qingkuang is $user_test"
```

设置 cron 任务

[java] view plain copy

```
*/15 * * * * bash /monitor.sh
55 23 * * * tar cxf /var/log/runrec /var/log/running.today && --remove-files
```

备注：设置 cron 计划任务是为了时时执行此脚本，来监控系统状态，并记入日志，便于对系统更好的管理！