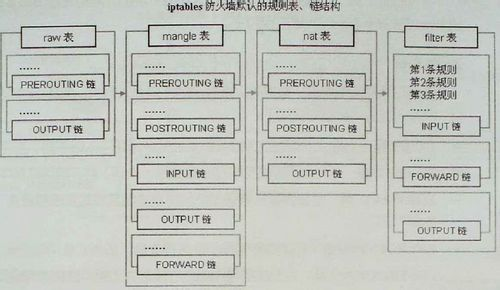
## 01/ linux操作增强

### 1.1 防火墙配置

防火墙根据配置文件/etc/sysconfig/iptables来控制本机的“出、入”网络访问行为

其对行为的配置策略有四个策略表



#### 1.1.1 必备技能

查看防火墙状态

service iptables status

关闭防火墙

service iptables stop

启动防火墙

service iptables start

禁止防火墙自启

chkconfig iptables off

#### ~~1.1.2 扩展了解~~

1、列出iptables规则

iptables -L -n

列出iptables规则并显示规则编号

iptables -L -n --line-numbers

2、列出iptables nat表规则（默认是filter表）

iptables -L -n -t nat

3、清除默认规则（注意默认是filter表，如果对nat表操作要加-t nat）

#清除所有规则

iptables -F

#重启iptables发现规则依然存在，因为没有保存

service iptables restart

#保存配置

service iptables save

4、禁止ssh登陆（如果服务器在机房，一定要小心）

iptables -A INPUT -p tcp --dport 22 -j DROP

#删除规则

iptables -D INPUT -p tcp --dport 22 -j DROP

加入一条INPUT规则开放80端口

iptables -I INPUT -p tcp --dport 80 -j ACCEPT

~~加入一条NAT规则~~

~~iptables -t nat -A PREROUTING -d 119.254.228.20 -p tcp -m tcp --dport 8022 -j DNAT --to-destination 172.16.0.111:22~~

~~删除一条NAT规则~~

~~iptables -t nat -D PREROUTING 6~~

### 1.2 高级文本处理命令

#### 1.2.1 cut命令

cut命令可以从一个文本文件或者文本流中提取文本列。

cut语法

[root@www ~]# cut -d'分隔字符' -f fields ## 用于有特定分隔字符

[root@www ~]# cut -c 字符区间 ## 用于排列整齐的信息

选项与参数：

-d：后面接分隔字符。与 -f 一起使用；

-f：依据 -d 的分隔字符将一段信息分割成为数段，用 -f 取出第几段的意思；

-c：以字符 (characters) 的单位取出固定字符区间；

PATH 变量如下

[root@www ~]# echo $PATH

*/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/bin:/usr/X11R6/bin:/usr/games*

*# 1 | 2 | 3 | 4 | 5 | 6 | 7*

将 PATH 变量取出，找出第五个路径。

# echo $PATH | cut -d':' -f 5

*/usr/local/bin*

将 PATH 变量取出，找出第三和第五个路径。

#echo $PATH | cut -d ':' -f 3,5

*/sbin:/usr/local/bin*

将 PATH 变量取出，找出第三到最后一个路径。

echo $PATH | cut -d ':' -f 3-

*/sbin:/usr/sbin:/usr/local/bin:/usr/X11R6/bin:/usr/games*

将 PATH 变量取出，找出第一到第三个路径。

#echo $PATH | cut -d ':' -f 1-3

*/bin:/usr/bin:/sbin:*

将 PATH 变量取出，找出第一到第三，还有第五个路径。

#echo $PATH | cut -d ':' -f 1-3,5

*/bin:/usr/bin:/sbin:/usr/local/bin*

实用例子:只显示/etc/passwd的用户和shell

#cat /etc/passwd | cut -d ':' -f 1,7

*root:/bin/bash*

*daemon:/bin/sh*

*bin:/bin/sh*

#### 1.2.2 sed命令

**1/ 删除：d命令**

$ sed '2d' example -----删除example文件的第二行。

$ sed '2,$d' example -----删除example文件的第二行到末尾所有行。

$ sed '$d' example -----删除example文件的最后一行。

$ sed '/test/'d example -----删除example文件所有包含test的行。

**2/ 替换：s命令**

$ sed 's/test/mytest/g' example

*## 在整行范围内把test替换为mytest。如果没有g标记，则只有每行第一个匹配的test被替换成mytest。*

$ sed -n 's/^test/mytest/p' example

*## (-n)选项和p标志一起使用表示只打印那些发生替换的行。也就是说，如果某一行开头的test被替换成mytest，就打印它。*

$ sed 's/^192.168.0.1/&localhost/' example

*## &符号表示追加一个串到找到的串后。所有以192.168.0.1开头的行都会被替换成它自已加 localhost，变成192.168.0.1localhost。*

$ sed -n 's/\(love\)able/\1rs/p' example

*## love被标记为1，所有loveable会被替换成lovers，而且替换的行会被打印出来。*

$ sed 's#10#100#g' example

*## 不论什么字符，紧跟着s命令的都被认为是新的分隔符，所以，“#”在这里是分隔符，代替了默认的“/”分隔符。表示把所有10替换成100。*

*选定行的范围：逗号*

$ sed -n '/test/,/check/p' example

*## 所有在模板test和check所确定的范围内的行都被打印。*

$ sed -n '5,/^test/p' example

*## 打印从第五行开始到第一个包含以test开始的行之间的所有行。*

$ sed '/test/,/check/s/$/sed test/' example

*## 对于模板test和west之间的行，每行的末尾用字符串sed test替换。*

*多点编辑：e命令*

$ sed -e '1,5d' -e 's/test/check/' example

*## (-e)选项允许在同一行里执行多条命令。如例子所示，第一条命令删除1至5行，第二条命令用check替换test。命令的执行顺序对结果有影响。如果两个命令都是替换命令，那么第一个替换命令将影响第二个替换命令的结果。*

$ sed --expression='s/test/check/' --expression='/love/d' example

*## 一个比-e更好的命令是--expression。它能给sed表达式赋值。*

**3/ 从文件读入：r命令**

$ sed '/test/r file' example

*-----file里的内容被读进来，显示在与test匹配的行下面，如果匹配多行，则file的内容将显示在所有匹配行的下面。*

**4/ 写入文件：w命令**

$ sed -n '/test/w file' example

-----在example中所有包含test的行都被写入file里。

**5/ 追加命令：a命令**

$ sed '/^test/a\\--->this is a example' example

*## '--->this is a example'被追加到以test开头的行后面，sed要求命令a后面有一个反斜杠。*

**6/ 插入：i命令**

$ sed '/test/i\\some thing new -------------------------' example

如果test被匹配，则把反斜杠后面的文本插入到匹配行的前面。

**7/ 下一个：n命令**

$ sed '/test/{ n; s/aa/bb/; }' example

-----如果test被匹配，则移动到匹配行的下一行，替换这一行的aa，变为bb，并打印该行，然后继续。

**8/ 退出：q命令**

$ sed '10q' example

-----打印完第10行后，退出sed。

#### 1.2.3 awk命令

awk是一个强大的文本分析工具，相对于grep的查找，sed的编辑，awk在其对数据分析并生成报告时，显得尤为强大。简单来说awk就是把文件逐行的读入，以空格为默认分隔符将每行切片，切开的部分再进行各种分析处理。

假设last -n 5的输出如下

[root@www ~]# last -n 5 ## 仅取出前五行

*root pts/1 192.168.1.100 Tue Feb 10 11:21 still logged in*

*root pts/1 192.168.1.100 Tue Feb 10 00:46 - 02:28 (01:41)*

*root pts/1 192.168.1.100 Mon Feb 9 11:41 - 18:30 (06:48)*

*dmtsai pts/1 192.168.1.100 Mon Feb 9 11:41 - 11:41 (00:00)*

*root tty1 Fri Sep 5 14:09 - 14:10 (00:01)*

如果只是显示最近登录的5个帐号

# last -n 5 | awk '{print $1}'

*root*

*root*

*root*

*dmtsai*

*root*

awk工作流程是这样的：读入有'\n'换行符分割的一条记录，然后将记录按指定的域分隔符划分域，填充域，$0则表示所有域,$1表示第一个域,$n表示第n个域。默认域分隔符是"空白键" 或 "[tab]键",所以$1表示登录用户，$3表示登录用户ip,以此类推。

如果只是显示/etc/passwd的账户

# cat /etc/passwd |awk -F ':' '{print $1}'

*root*

*daemon*

*bin*

*sys*

这种是awk+action的示例，每行都会执行action{print $1}。

-F指定域分隔符为':'

如果只是显示/etc/passwd的账户和账户对应的shell,而账户与shell之间以tab键分割

#cat /etc/passwd |awk -F ':' '{print $1"\t"$7}'

root /bin/bash

daemon /bin/sh

bin /bin/sh

sys /bin/sh

如果只是显示/etc/passwd的账户和账户对应的shell,而账户与shell之间以逗号分割,而且在所有行添加列名name,shell,在最后一行添加"blue,/bin/nosh"。

cat /etc/passwd |awk -F ':' 'BEGIN {print "**name,shell**"} {print $1","$7} END {print "**blue,/bin/nosh**"}'

***name,shell***

*root,/bin/bash*

*daemon,/bin/sh*

*bin,/bin/sh*

*sys,/bin/sh*

*....*

***blue,/bin/nosh***

awk工作流程是这样的：先执行BEGIN，然后读取文件，读入有/n换行符分割的一条记录，然后将记录按指定的域分隔符划分域，填充域，$0则表示所有域,$1表示第一个域,$n表示第n个域,随后开始执行模式所对应的动作action。接着开始读入第二条记录······直到所有的记录都读完，最后执行END操作。

搜索/etc/passwd有root关键字的所有行

#awk -F: '/root/' /etc/passwd

root:x:0:0:root:/root:/bin/bash

这种是pattern的使用示例，匹配了pattern(这里是root)的行才会执行action(没有指定action，默认输出每行的内容)。

搜索支持正则，例如找root开头的: awk -F: '/^root/' /etc/passwd

搜索/etc/passwd有root关键字的所有行，并显示对应的shell

# awk -F':' '/root/{print $7}' /etc/passwd

/bin/bash

这里指定了action{print $7}

统计/etc/passwd:文件名，每行的行号，每行的列数，对应的完整行内容:

#awk -F ':' '{print "filename:" FILENAME ",linenumber:" NR ",columns:" NF ",linecontent:"$0}' */etc/passwd*

*filename:/etc/passwd,linenumber:1,columns:7,linecontent:root:x:0:0:root:/root:/bin/bash*

*filename:/etc/passwd,linenumber:2,columns:7,linecontent:daemon:x:1:1:daemon:/usr/sbin:/bin/sh*

*filename:/etc/passwd,linenumber:3,columns:7,linecontent:bin:x:2:2:bin:/bin:/bin/sh*

*filename:/etc/passwd,linenumber:4,columns:7,linecontent:sys:x:3:3:sys:/dev:/bin/sh*

使用printf替代print,可以让代码更加简洁，易读

awk -F ':' '{printf("filename:%s,linenumber:%s,columns:%s,linecontent:%s\n",FILENAME,NR,NF,$0)}' /etc/passwd

### 1.3 crontab配置

#### 1、命令功能

通过crontab 命令，我们可以在固定的间隔时间执行指定的系统指令或 shell script脚本。时间间隔的单位可以是分钟、小时、日、月、周及以上的任意组合。这个命令非常适合周期性的日志分析或数据备份等工作。

#### 2、安装crontab

yum install crontabs

服务操作说明：

service crond start  ## 启动服务

service crond stop   ## 关闭服务

service crond restart ## 重启服务

service crond reload  ## 重新载入配置

## 查看crontab服务状态：

service crond status

## 手动启动crontab服务：

service crond start

## 查看crontab服务是否已设置为开机启动，执行命令：

chkconfig --list

## 加入开机自动启动：

chkconfig  --level 35 crond on

#### 3、命令格式

crontab [-u user] file

crontab [-u user] [ -e | -l | -r ]

参数说明：

-u user：用来设定某个用户的crontab服务，例如，“-u ixdba”表示设定ixdba用户的crontab服务，此参数一般有root用户来运行。

file：file是命令文件的名字,表示将file做为crontab的任务列表文件并载入crontab。

-e：编辑某个用户的crontab文件内容。如果不指定用户，则表示编辑当前用户的crontab文件。

-l：显示某个用户的crontab文件内容，如果不指定用户，则表示显示当前用户的crontab文件内容。

-r：删除定时任务配置，从/var/spool/cron目录中删除某个用户的crontab文件，如果不指定用户，则默认删除当前用户的crontab文件。

-i：在删除用户的crontab文件时给确认提示。

命令示例：

crontab file [-u user] ## 用指定的文件替代目前的crontab。

***必掌握：***

crontab -l [-u user] ## 列出用户目前的crontab.

crontab -e [-u user] ## 编辑用户目前的crontab.

#### 4、调度配置

##### 4.1 配置说明

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0～23（0表示0点） 7-9表示：8点到10点之间

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

##### 4.2 配置示例

\*/1 \* \* \* \* date >> /root/date.txt

上面的例子表示每分钟执行一次date命令

30 21 \* \* \* /usr/local/etc/rc.d/httpd 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/httpd 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/httpd 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/httpd 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/httpd 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/httpd restart

上面的例子每一小时重启apache

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

上面的例子晚上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/httpd restart

上面的例子每月的4号与每周一到周三的11点重启apache

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

上面的例子一月一号的4点重启apache

## 02/ shell编程

Shell是用户与内核进行交互操作的一种接口，目前最流行的Shell称为bash Shell

Shell也是一门编程语言**<解释型的编程语言>**，即shell脚本<就是在用linux的shell命令编程>

一个系统可以存在多个shell，可以通过cat /etc/shells命令查看系统中安装的shell，不同的shell可能支持的命令语法是不相同的

### 2.1 基本格式

代码写在普通文本文件中，通常以 .sh为后缀名

vi hello.sh

|  |
| --- |
| #!/bin/bash ## 表示用哪一种shell解析器来解析执行我们的这个脚本程序  echo "hello world" ## 注释也可以写在这里  ## 这是一行注释 |

执行脚本

sh hello.sh

或给脚本添加x权限，直接执行

chmod 755 hello.sh

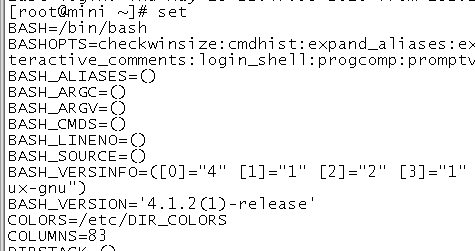
./hello.sh

### 2.2 基本语法

#### 2.2.1 系统变量

Linux Shell中的变量分为“系统变量”和“用户自定义变量”

可以通过set命令查看系统变量



系统变量：$HOME、$PWD、$SHELL、$USER等等

#### 2.2.2 自定义变量

**1、语法**

变量＝值 （例如STR=abc）

等号两侧不能有空格

变量名称一般习惯为大写

使用变量： $arg

双引号和单引号有区别，

双引号仅将空格脱意，

单引号会将变量引用比如$param脱意

**2、示例**

STR="hello world"

A=9

echo $A

echo $STR

如果想打印 hello worlds is greater 怎么办？

*echo $STRs is greate 行吗？*

不行，正确写法是：

echo ${*STR}s is greate*

unset A 撤销变量 A

readonly B=2 声明静态的变量 B=2，不能 unset

export A #可把变量提升为当前shell进程中的全局环境变量，可供其他子shell程序使用

注意理解export：

|  |
| --- |
| [root@shizhan01 scripts]# vi a.sh  #!/bin/bash  a="a in a.sh"  echo $a  /root/scripts/b.sh |

|  |
| --- |
| [root@shizhan01 scripts]# vi b.sh  #!/bin/bash  b="b in b.sh"  echo $b  echo $a |

然后执行 ./a.sh ，会发现 b脚本中并没有把a脚本中定义的a变量打印出来

如果要在b中打印出a脚本的变量a，需要在a脚本中把变量a做export定义

此时，a变量就成了a.sh脚本所在bash进程的全局变量，该进程的所有子进程都能访问到变量a

另一种方式：

如果在a.sh脚本中用如下方式调用b.sh

|  |
| --- |
| . ./b.sh ## 注意：重点关注最前面那个 “.”号 |

或者

|  |
| --- |
| source ./b.sh ## |

则，b.sh就在a.sh所在的bash进程空间中运行

**总结：**

**1、a.sh中直接调用b.sh，会让b.sh在a所在的bash进程的“子进程”空间中执行**

**2、而子进程空间只能访问父进程中用export定义的变量**

**3、一个shell进程无法将自己定义的变量提升到父进程空间中去**

**4、“.”号执行脚本时，会让脚本在调用者所在的shell进程空间中执行**

**3、反引号赋值**

A=`ls -la` ## 反引号，运行里面的命令，并把结果返回给变量A

A=$(ls -la) ## 等价于反引号

**4、特殊变量**

$? 表示上一个命令退出的状态码(0-255)

$$ 表示当前进程编号

$0 表示当前脚本名称

$n 表示n位置的输入参数（n代表数字，n>=1）

$# 表示参数的个数，常用于循环

$\*和$@ 都表示参数列表

***注：$\*与$@区别***

$\* 和 $@ 都表示传递给函数或脚本的所有参数

* 不被双引号" "包含时——

$\* 和 $@ 都以$1 $2 … $n 的形式组成参数列表

* 当它们被双引号" "包含时——

"$\*" 会将所有的参数作为一个整体，以**"**$1 $2 … $n**"**的形式组成一个整串；

"$@" 会将各个参数分开，以"$1" "$2" … "$n" 的形式组成一个参数列表

|  |
| --- |
| [root@bigdata01 ~]# vi p.sh  #!/bin/bash  for i in $@  do  echo $i  done  echo '-------------'  for i in "$@"  do  echo $i  done  echo '-------------'  for i in $\*  do  echo $i  done  echo '-------------'  for i in "$\*"  do  echo $i  done |
| [root@bigdata01 ~]# sh p.sh 1 2 3 4  1  2  3  4  -------------  1  2  3  4  -------------  1  2  3  4  -------------  1 2 3 4 |

### 2.3 算数表达式运算

##### 1、用expr

格式 expr m + n 或$((m+n)) 注意expr运算符间要有空格

例如计算（2＋3 ）×4 的值

1 .分步计算  
 S=`expr 2 + 3`  
 expr $S \\* 4 ## \*号需要转义

2.一步完成计算

expr `expr 2 + 3 ` \\* 4

echo `expr \`expr 2 + 3\` \\* 4`

##### 2、用(())

((1+2))

(((2+3)\*4))

count=1

((count++))

echo $count

但是要想取到运算结果，需要用$引用

a=$((1+2))

##### 3、用$[]

a=$[1+2]

echo $a

##### 4、用let

i=1

let i++

let i=i+2

### 2.5 流程控制

#### 2.5.1 if语法

##### 1、语法格式

if condition   
then   
    statements   
[elif condition   
    then statements. ..]   
[else   
    statements ]   
fi

##### 2、示例

|  |
| --- |
| #!/bin/bash  read -p "please input your name:" NAME ## read命令用于从控制台读取输入数据  ## printf '%s\n' $NAME  if [ $NAME = root ]  then  echo "hello ${NAME}, welcome !"  elif [ $NAME = qianfeng ]  then  echo "hello ${NAME}, welcome !"  else  echo "SB, get out here !"  fi |

##### 3、判断条件

1/ 条件判断基本语法

[ condition ]  *(注意condition前后要有空格)*

#非空返回true，可使用$?验证（0为true，>1为false）

[ qfedu ]

#空返回false

[ ]

注意[ ]内部的=周边的空格，有区别：

|  |
| --- |
| [root@shizhan01 scripts]# if [ a = b ];then echo ok;else echo notok;fi  notok  [root@shizhan01 scripts]# if [ a=b ];then echo ok;else echo notok;fi  ok |

短路（理解为三元运算符）

[ condition ] && echo OK || echo notok

条件满足，执行&&后面的语句；条件不满足，执行|| 后面的语句

2/ 条件判断组合

注：[] 与[[ ]] 的区别：[[ ]] 中逻辑组合可以使用 && || 符号

而[] 里面逻辑组合可以用 -a -o

|  |
| --- |
| [root@mini ~]# if [ a = b && b = c ]; then echo ok;else echo notok;fi  *-bash: [: missing `]'*  *notok*  [root@mini ~]# if [ a = b -a b = b ]; then echo ok;else echo notok;fi  *notok*  [root@mini ~]# if [ a = b -o b = b ]; then echo ok;else echo notok;fi  *ok*  [root@mini ~]# if [[ a = b && b = b ]]; then echo ok;else echo notok;fi  *notok*  [root@mini ~]# if [[ a = b || b = b ]]; then echo ok;else echo notok;fi  *ok* |

3/ 常用判断运算符

**字符串比较**：

= 字符串是否相等

!= 字符串是否不相等

-z 字符串长度是为0返回true

-n 字符串长度是不为0返回true

if [ 'aa' = 'bb' ]; then echo ok; else echo notok;fi

if [ -n "aa" ]; then echo ok; else echo notok;fi

if [ -z "" ]; then echo ok; else echo notok;fi

**整数比较：**

-lt 小于

-le 小于等于

-eq 等于

-gt 大于

-ge 大于等于

-ne 不等于

还可以用转义的数学符号 \<

**文件判断：**

-d 是否为目录

if [ -d /bin ]; then echo ok; else echo notok;fi

-f 是否为文件

*if [ -f /bin/ls ]; then echo ok; else echo notok;fi*

-e 是否存在

*if [ -e /bin/ls ]; then echo ok; else echo notok;fi*

#### 2.5.2 while语法

##### 1、方式一

while expression

do

command

…

done

##### 2、方式二

i=1

while ((i<=3))

do

echo $i

let i++

done

#### 2.5.3 case语法

case $1 in

start)

echo "starting"

;;

stop)

echo "stoping"

;;

\*)

echo "Usage: {start|stop}"

esac

#### 2.5.4 for语法

##### 1、方式一

for N in 1 2 3

do

echo $N

done

或

for N in 1 2 3; do echo $N; done

或

for N in {1..3}; do echo $N; done

##### 2、方式二

for ((i = 0; i <= 5; i++))

do

echo "welcome $i times"

done

或

for ((i = 0; i <= 5; i++)); do echo "welcome $i times"; done

### 2.6 函数使用

#### 2.6.1 函数定义

|  |
| --- |
| #!/bin/sh  # func1.sh  hello() ## 函数定义  {  echo "Hello there today's date is `date +%Y-%m-%d`"  return 2 ###返回值其实是状态码，只能在[0-255]范围内  }  hello  echo $? 获取函数的return值  echo "now going to the function hello"  echo "back from the function" |

函数调用：

function hello()

或 function hello

或 hello

*注意：*

*1.必须在调用函数地方之前，先声明函数，shell脚本是逐行运行。不会像其它语言一样先预编译*

*2.函数返回值，只能通过$? 系统变量获得，可以显示加：return 返回，如果不加，将以最后一条命令运行结果，作为返回值。 return后跟数值n(0-255)*

*~~脚本调试：~~*

*sh -vx helloWorld.sh*

*或者在脚本中增加set -x*

#### 2.6.2 函数参数

|  |
| --- |
| #!/bin/bash  # fun1.sh  funWithParam(){  echo "第一个参数为 $1 !"  echo "第二个参数为 $2 !"  echo "第十个参数为 $10 !"  echo "第十个参数为 ${10} !"  echo "第十一个参数为 ${11} !"  echo "参数总数有 $# 个!"  echo "作为一个字符串输出所有参数 $\* !"  }  funWithParam 1 2 3 4 5 6 7 8 9 34 73 |

*注意，$10 不能获取第十个参数，获取第十个参数需要${10}。当n>=10时，需要使用${n}来获取参数。*

#### 2.6.3 函数返回值

|  |
| --- |
| #!/bin/bash  # fun2.sh  funWithReturn(){  echo "这个函数会对输入的两个数字进行相加运算..."  echo "输入第一个数字: "  read aNum  echo "输入第二个数字: "  read anotherNum  echo "两个数字分别为 $aNum 和 $anotherNum !"  return $(($aNum+$anotherNum))  }  funWithReturn  echo "输入的两个数字之和为 $? !" |

#### 2.6.4 跨脚本调用函数

假如上述的脚本文件fun2.sh保存在此路径： /root/fun2.sh

则可在脚本fun\_other.sh中调用脚本fun2.sh中的函数

|  |
| --- |
| #!/bin/bash  # fun\_other.sh  . /root/fun2.sh *## 注：* ***.*** *和* ***/*** *之间有空格*  # 或者 source /root/fun2.sh  funWithParam 11 22 33 44 55 66 77 88 99 100 101 |

## 03/ shell编程综合练习

自动化软件部署脚本

### 3.1 需求

#### 1、需求描述

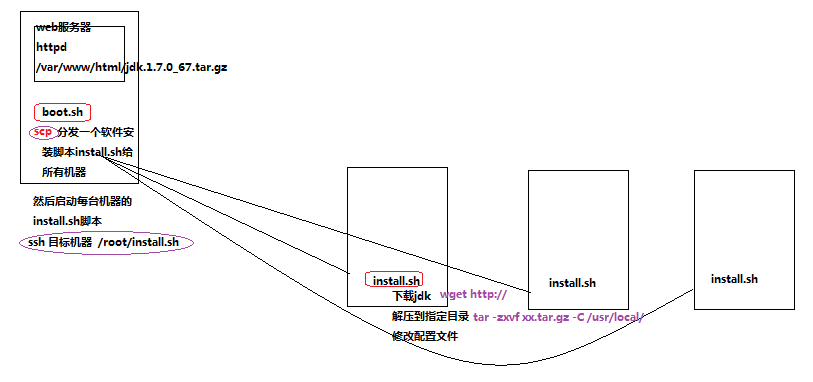
公司内有一个N个节点的集群，需要统一安装一些软件（jdk）

需要开发一个脚本，实现对集群中的N台节点批量自动下载、安装jdk

#### 2、思路

1/ 编写一个**启动脚本**，用来发送一个软件**安装脚本**到每一台机器

2/ 然后启动每台机器上的软件安装脚本来执行软件下载和安装



#### 3、expect的使用

痛点：使用scp命令远程拷贝文件时，会有人机交互的过程，如何让脚本完成人机交互？

妙药： expect

用法示例：

先观察 ssh localhost 的过程

再看expect的功能

|  |
| --- |
| #!/bin/bash/expect  ## exp\_test.sh  set timeout -1;  spawn ssh localhost;  expect {  "(yes/no)" {send "yes\r";exp\_continue;}  "password:" {send "hadoop\r";exp\_continue;}  eof {exit 0;}  } |

*执行： expect -f exp\_test.sh*

### 3.2 准备内网软件下载服务器

选择一台服务器（比如mini）作为软件源服务器

1、安装httpd *（如果已有，可跳过）*

yum install -y httpd

service httpd start

chkconfig --level 35 httpd on

#### 2、制作局域网yum源

1/ 挂载centos安装光盘到/mnt/cdrom *（如果已有，可跳过）*

mkdir /mnt/cdrom

mount -t iso9660 -o loop /dev/cdrom /mnt/cdrom

2/ 将本地yum库放入httpd服务器

ln -s /mnt/cdrom /var/www/html/centos

*检查点：用浏览器访问 http://mini/centos 看能否看到光盘内容*

#### 3、编写repo配置

vi /etc/yum.repos.d/centos.repo

|  |
| --- |
| [c6-httpd]  name=CentOS-httpd  baseurl=http://192.168.33.3/centos  gpgcheck=0  enabled=1 |

#### 4、分发repo配置到局域网

从母鸡shizhan01上把centos.repo拷贝给所有需要自动安装软件的服务器（仔鸡mini1/mini2）

cd /etc/yum.repos.d/

scp /etc/yum.repos.d/innet.repo mini1:$PWD

scp /etc/yum.repos.d/innet.repo mini2:$PWD

#### 5、准备一个jdk安装包放在内网web服务器上

### 3.3 脚本开发

#### 1、启动脚本

vi boot.sh

|  |
| --- |
| #!/bin/bash  SERVERS="hadoop02 hadoop03"  PASSWORD=hadoop  BASE\_SERVER=192.168.137.81  ## 实现免密登陆配置的函数  auto\_ssh\_copy\_id() {  expect -c "set timeout -1;  spawn ssh-copy-id $1;  expect {  \*(yes/no)\* {send -- yes\r;exp\_continue;}  \*assword:\* {send -- $2\r;exp\_continue;}  eof {exit 0;}  }";  }  ssh\_copy\_id\_to\_all() {  for SERVER in $SERVERS  do  auto\_ssh\_copy\_id $SERVER $PASSWORD  done  }  ## 调用免密登陆配置函数，实现母鸡到各仔鸡的免密登陆配置  ssh\_copy\_id\_to\_all  ## 完成分发install.sh到各仔鸡的操作  ## 并让仔鸡启动install.sh  for SERVER in $SERVERS  do  scp install.sh root@$SERVER:/root  ssh root@$SERVER /root/install.sh  done |

#### 2、安装执行脚本

vi install.sh

|  |
| --- |
| #!/bin/bash  BASE\_SERVER=192.168.137.81  ## 为本机安装wget命令  yum install -y wget  ## 使用wget从母鸡的web服务器上下载jdk压缩包  wget $BASE\_SERVER/soft/ jdk-8u60-linux-x64.gz  ## 将下载的压缩包解压  tar -zxvf jdk-8u60-linux-x64.gz -C /usr/local  ## 修改profile配置文件  cat >> /etc/profile << EOF  export JAVA\_HOME=/usr/local/jdk1.8.0\_60  export PATH=\$PATH:\$JAVA\_HOME/bin  EOF |

#### 3、启动脚本

只要在baseServer即mini上启动boot.sh即可

### 作业：

批量部署脚本，存在一些不太完美的地方，比如，目标机器名需要写死在脚本中，请改造这一点：

可以将所有需要安装软件的机器名写在一个文件：比如slaves中

让脚本自动读取slaves文件中的机器名来批量安装

cat slaves | while read host

do

echo $host

expect -c "set timeout -f

spawn ssh-copy-id $host"

done

2/目前脚本在软件安装的时候，是逐台进行的，可以改造成并行进行

3、假设有3台机器，环境都满足，ssh的客户端都安装好了，然后，免密登录都没有配置。那么，需要咱们实现自动部署脚本，完成以下功能：自动对每一台机器实现免密登录配置。