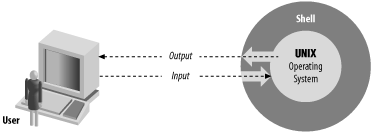
# shell基础知识

## 什么是Shell

Shell是命令解释器(command interpreter)，是Unix操作系统的用户接口，程序从用户接口得到输入信息，shell将用户程序及其输入翻译成操作系统内核（kernel）能够识别的指令，并且操作系统内核执行完将返回的输出通过shell再呈现给用户，下图所示用户、shell和操作系统的关系：



Shell也是一门编程语言，即shell脚本，shell是解释执行的脚本语言，可直接调用linux命令。 .java -> .class

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

## Shell种类

操作系统内核（kernel）与shell是独立的套件，而且都可被替换：

不同的操作系统使用不同的shell；

同一个kernel之上可以使用不同的shell。

常见的shell分为两大主流：

**sh**：

Bourne shell（sh） ,Solaris,hpux默认shell

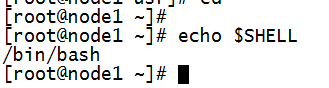
Bourne again shell（bash） ,Linux系统默认shell

**csh**：

C shell(csh)

tc shell(tcsh)

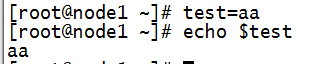
## 查看使用Shell



## Shell环境定义

### 临时环境变量

所谓临时变量是指在用户在当前登陆环境生效的变量，用户登陆系统后，直接在命令行上定义的环境变量便只能在当前的登陆环境中使用。当退出系统后，环境变量将不能下次登陆时继续使用。



### 将环境变量永久生效

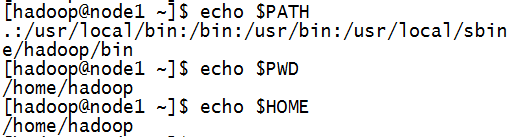
通过将环境变量定义写入到配置文件中，用户每次登陆时系统自动定义，则无需再到命令行重新定义。定义环境变量的常见配置文件如下：

**/etc/profile** 针对系统所有用户生效，此文件应用于所有用户每次登陆系统时的环境变量定义

**$HOME\_name/.bash\_profile**  针对特定用户生效，$HOME为用户的宿主目录，当用户登陆系统后，首先继承/etc/profile文件中的定义，再应用$HOME/.bash\_profile文件中的定义。

### 系统预定义的环境变量

系统环境变量对所有用户有效，如：$PATH、$HOME、$SHELL、$PWD等等，如下用echo命令打印上述的系统环境变量：



# shell脚本编程

同传统的编程语言一样，shell提供了很多特性，这些特性可以使你的shell脚本编程更为有用。

## 创建Shell脚本

一个shell脚本通常包含如下部分：

### 首行

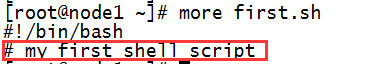
第一行内容在脚本的首行左侧，表示脚本将要调用的shell解释器，内容如下：

#!/bin/bash

#！符号能够被内核识别成是一个脚本的开始，这一行必须位于脚本的首行，/bin/bash是bash程序的绝对路径，在这里表示后续的内容将通过bash程序解释执行。

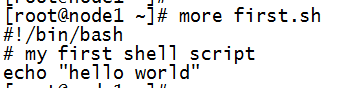
### 注释

注释符号# 放在需注释内容的前面，如下：



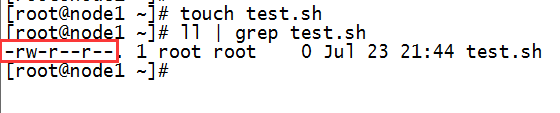
### 内容

可执行内容和shell结构

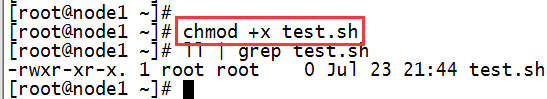


## Shell脚本的权限

一般情况下，默认创建的脚本是没有执行权限的



没有权限不能执行，需要赋予可执行权限



## Shell脚本的执行

### 1 输入脚本的绝对路径或相对路径

/root/helloWorld.sh

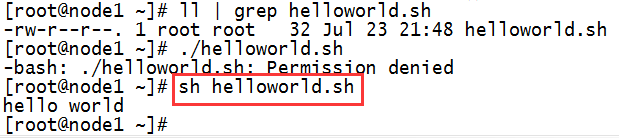
./helloWorld.sh

### 2 bash或sh +脚本

bash /root/helloWorld.sh

sh helloWorld.sh

注：当脚本没有x权限时，root和文件所有者通过该方式可以正常执行。



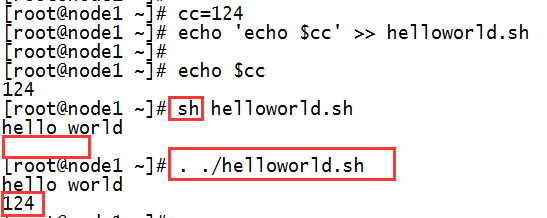
### 3 在脚本的路径前再加". " 或source

source /root/helloWorld.sh

. ./helloWorld.sh

区别：第一种和第二种会新开一个bash，不同bash中的变量无法共享

但是使用. ./脚本.sh 这种方式是在同一个shell里面执行的。



可以使用pstree查看

source eg.sh

## Shell变量

变量：是shell传递数据的一种方式，用来代表每个取值的符号名。

当shell脚本需要保存一些信息时，如一个文件名或是一个数字，就把它存放在一个变量中。

### 变量设置规则：

1，变量名称可以由字母，数字和下划线组成，但是不能以数字开头，环境变量名建议大写，便于区分。

2，在bash中，变量的默认类型都是字符串型，如果要进行数值运算，则必须指定变量类型为数值型。

3，变量用等号连接值，等号左右两侧不能有空格。

4，变量的值如果有空格，需要使用单引号或者双引号包括。

### 变量分类

Linux Shell中的变量分为用户自定义变量,环境变量，位置参数变量和预定义变量。

可以通过set命令查看系统中存在的所有变量

系统变量：保存和系统操作环境相关的数据。$HOME、$PWD、$SHELL、$USER等等

位置参数变量：主要用来向脚本中传递参数或数据，变量名不能自定义，变量作用固定。

预定义变量：是Bash中已经定义好的变量，变量名不能自定义，变量作用也是固定的。

### 用户自定义变量

用户自定义的变量由字母或下划线开头，由字母，数字或下划线序列组成，并且大小写字母意义不同，变量名长度没有限制。

#### 设置变量：

习惯上用大写字母来命名变量。变量名以字母表示的字符开头，不能用数字。

#### 变量调用

在使用变量时，要在变量名前加上前缀“$”.

使用echo 命令查看变量值。eg:echo $A

#### 变量赋值：

##### 1,定义时赋值：

变量＝值

等号两侧不能有空格

eg:

STR="hello world"

A=9

##### 2, 将一个命令的执行结果赋给变量

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

eg: aa=$((4+5))

bb=`expr 4 + 5 `

##### 3，将一个变量赋给另一个变量

eg : A=$STR

**变量叠加**

eg:#aa=123

eg:#cc="$aa"456

eg:#dd=${aa}789

单引号和双引号的区别：

现象：单引号里的内容会全部输出，而双引号里的内容会有变化

原因：单引号会将所有特殊字符脱意

NUM=10

SUM="$NUM hehe"     echo $SUM     输出10 hehe

SUM2='$NUM hehe'     echo $SUM2    输出$NUM hehe

#### 列出所有的变量：

# set

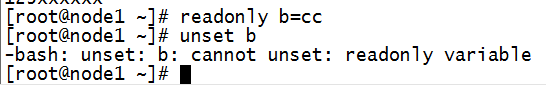
#### 删除变量：

# unset  NAME

eg :

# unset A 撤销变量 A

# readonly B=2 声明静态的变量 B=2 ，不能 unset



用户自定义的变量，作用域为当前的shell环境。

### 环境变量

用户自定义变量只在当前的shell中生效，而环境变量会在当前shell和其所有子shell中生效。如果把环境变量写入相应的配置文件，那么这个环境变量就会在所有的shell中生效。

export 变量名=变量值 申明变量

作用域：当前shell以及所有的子shell

### 位置参数变量

|  |  |
| --- | --- |
| $n | n为数字，$0代表命令本身，$1-$9代表第一到第9个参数,  十以上的参数需要用大括号包含，如${10}。 |
| $\* | 代表命令行中所有的参数，把所有的参数看成一个整体。以"$1 $2 … $n"的形式输出所有参数 |
| $@ | 代表命令行中的所有参数，把每个参数区分对待。以"$1" "$2" … "$n" 的形式输出所有参数 |
| $# | 代表命令行中所有参数的个数。添加到shell的参数个数 |

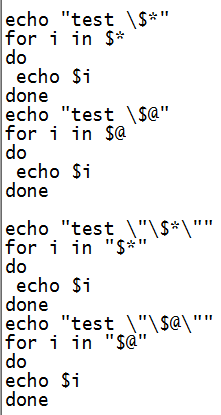
shift指令：参数左移，每执行一次，参数序列顺次左移一个位置，$# 的值减1，用于分别处理每个参数，移出去的参数不再可用

**$\* 和 $@的区别**

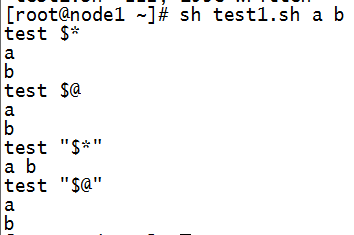
$\* 和 $@ 都表示传递给函数或脚本的所有参数，不被双引号" "包含时，都以"$1" "$2" … "$n" 的形式输出所有参数

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 … $n"的形式输出所有参数；"$@" 会将各个参数分开，以"$1" "$2" … "$n" 的形式输出所有参数

shell脚本中执行测试：



输出结果：



### 预定义变量

|  |  |
| --- | --- |
| $? | 执行上一个命令的返回值   执行成功，返回0，执行失败，返回非0（具体数字由命令决定） |
| $$ | 当前进程的进程号（PID），即当前脚本执行时生成的进程号 |
| $! | 后台运行的最后一个进程的进程号（PID），最近一个被放入后台执行的进程 & |

# vi pre.sh

pwd >/dev/null

echo $$

ls /etc >/dev/null &

echo $!

### read命令

read [选项] 值

read -p(提示语句) -n(字符个数) -t(等待时间，单位为秒) –s(隐藏输入)

eg:

read –t 30 –p “please input your name: ” NAME

echo $NAME

read –s –p “please input your age : ” AGE

echo $AGE

read –n 1 –p “please input your sex [M/F]: ” GENDER

echo $GENDER

## 运算符

num1=11

num2=22

sum=$num1+$num2

echo $sum

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

expr命令：对整数型变量进行算术运算

**(注意：运算符前后必须要有空格)**

expr 3 + 5     
expr 3 – 5

echo `expr 10 / 3`

10/3的结果为3，因为是取整  
expr  3 \\* 10

\ 是转义符

计算（2 ＋3 ）×4 的值

1 .分步计算  
 S=`expr 2 + 3`  
 expr $S \\* 4

2.一步完成计算

expr `expr 2 + 3` \\* 4

S=`expr \`expr 2 + 3\` \\* 4`

echo $S

或

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

**$()与${}的区别**

$( )的用途和反引号``一样，用来表示优先执行的命令

eg:echo $(ls a.txt)

${ } 就是取变量了  eg：echo ${PATH}

$((运算内容)) 适用于数值运算

eg: echo $((3+1\*4))

## 条件测试

### 内置test命令

内置test命令常用操作符号[]表示，将表达式写在[]中，如下：

[ expression ]

或者：

test expression

**注意**：expression首尾都有个空格

eg: [ ] ;echo $?

测试范围：整数、字符串、文件

表达式的结果为真，则test的返回值为0，否则为非0。

当表达式的结果为真时，则变量$?的值就为0，否则为非0

#### 字符串测试：

test  str1 == str2    测试字符串是否相等 =

test  str1 != str2    测试字符串是否不相等  
test  str1            测试字符串是否不为空,不为空，true，false  
test  -n str1     测试字符串是否不为空  
test  -z  str1    测试字符串是否为空

eg:

name=linzhiling

[ “$name” ] && echo ok

； 命令连接符号

&& 逻辑与 条件满足，才执行后面的语句

[ -z “$name” ] && echo invalid || echo ok

|| 逻辑或，条件不满足，才执行后面的语句

test “$name” == ”yangmi” && echo ok || echo invalid

#### 整数测试:

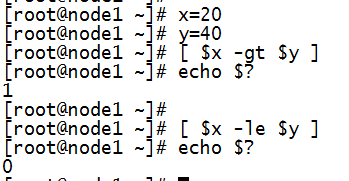
test   int1 -eq  int2    测试整数是否相等 equals  
test   int1 -ge  int2    测试int1是否>=int2  
test   int1 -gt  int2    测试int1是否>int2  
test   int1 -le  int2    测试int1是否<=int2  
test   int1 -lt  int2    测试int1是否<int2  
test   int1 -ne  int2    测试整数是否不相等

eg:

test 100 –gt 100

test 100 –ge 100

如下示例两个变量值的大小比较：



-gt表示greater than大于的意思，-le表示less equal表示小于等于。

#### 文件测试：

test  -d  file      指定文件是否目录

test  –e  file     文件是否存在 exists  
test  -f  file     指定文件是否常规文件  
test –L File     文件存在并且是一个符号链接

test  -r  file    指定文件是否可读  
test  -w  file    指定文件是否可写

test  -x  file    指定文件是否可执行

eg:

test -d install.log

test –r install.log

test –f xx.log ; echo $?

[ -L service.soft ] && echo “is a link”

test -L /bin/sh ;echo $?

[ -f /root ] && echo “yes” || echo “no”

#### 多重条件测试：

条件1 –a 条件2 逻辑与 两个都成立，则为真

条件1 –o 条件2 逻辑或 只要有一个为真，则为真

！ 条件 逻辑非 取反

eg:

num=520

[ -n “$num” –a “$num” –ge 520 ] && echo “marry you” || echo “go on”

age=20

pathname=outlog

[ ! -d“$ pathname”] && echo usable || echo used

## 流程控制语句

### if/else命令

1. **单分支if条件语句**

if [ 条件判断式 ]

    then

        程序

fi

**或者**

if [ 条件判断式 ] ; then

    程序

fi

eg:#!/bin/sh

if  [ -x  /etc/rc.d/init.d/httpd ]

    then

    /etc/rc.d/init.d/httpd restart

fi

单分支条件语句需要注意几个点

if语句使用fi结尾，和一般语言使用大括号结尾不同。

[ 条件判断式 ] 就是使用test命令判断，所以中括号和条件判断式之间必须有空格

then后面跟符号条件之后执行的程序，可以放在[]之后，用“;”分割，也可以换行写入，就不需要"；"了。

2,**多分支if条件语句**

if [ 条件判断式1 ]

    then

        当条件判断式1成立时，执行程序1

elif  [ 条件判断式2 ]

    then

        当条件判断式2成立时，执行程序2

...省略更多条件

else

    当所有条件都不成立时，最后执行此程序

fi

示例：

read -p "please input your name: " NAME

eg:

#!/bin/bash

read -p "please input your name:" NAME

#echo $NAME

if [ “$NAME” == root ]

then

echo "hello ${NAME}, welcome !"

elif [ $NAME == tom ]

then

echo "hello ${NAME}, welcome !"

else

echo "SB, get out here !"

fi

### case命令

case命令是一个多分支的if/else命令，case变量的值用来匹配value1,value2,value3等等。匹配到后则执行跟在后面的命令直到遇到双分号为止(;;)case命令以esac作为终止符。

**格式**

CMD=$1

case $CMD in

start)

echo "starting"

;;

Stop)

echo "stoping"

;;

\*)

echo "Usage: {start|stop} “

esac

### for循环

for循环命令用来在一个列表条目中执行有限次数的命令。比如，你可能会在一个姓名列表或文件列表中循环执行同个命令。for命令后紧跟一个自定义变量、一个关键字in和一个字符串列表（可以是变量）。第一次执行for循环时，字符串列表中的第一个字符串会赋值给自定义变量，然后执行循环命令，直到遇到done语句；第二次执行for循环时，会右推字符串列表中的第二个字符串给自定义变量，依次类推，直到字符串列表遍历完。

第一种：

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

第二种：

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

do

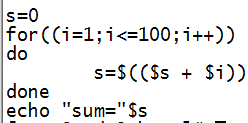
echo "welcome $i times"

done

或

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

练习：计算从1到100的加和。



### while循环

while命令根据紧跟其后的命令(command)来判断是否执行while循环，当command执行后的返回值(exit status)为0时，则执行while循环语句块，直到遇到done语句，然后再返回到while命令，判断command的返回值，当得打返回值为非0时，则终止while循环。

第一种

while expression

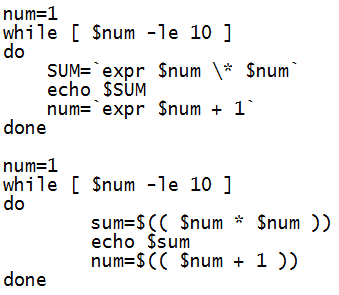
do

command

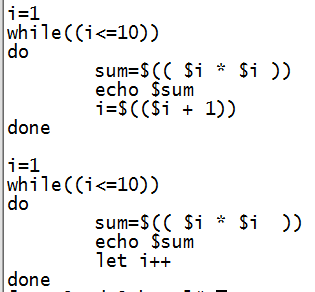
…

done

练习：求1-10 各个数的平方和



第二种方式：



## 自定义函数

函数代表着一个或一组命令的集合，表示一个功能模块，常用于模块化编程。

以下是关于函数的一些重要说明：

在shell中，函数必须先定义，再调用

使用return value来获取函数的返回值

函数在当前shell中执行，可以使用脚本中的变量。

函数的格式如下：

函数名()

{

命令1…..

命令2….

return 返回值变量

}

[ function ] funname [()]

{

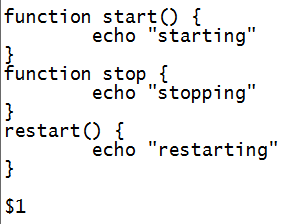
  action;

  [return int;]

}

function start() / function start / start()

eg:



注意：

如果函数名后没有（），在函数名和{ 之间，必须要有**空格**以示区分。

函数返回值，只能通过$? 系统变量获得，可以显示加：return 返回值，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 -x script  
这将执行该脚本并显示所有变量的值。

在shell脚本里添加

set -x  对部分脚本调试  
sh -n script  
不执行脚本只是检查语法的模式，将返回所有语法错误。

sh –v script

执行并显示脚本内容

# awk和sed

**cut [选项]  文件名        默认分割符是制表符**

选项：

    -f 列号：    提取第几列

    -d 分隔符：    按照指定分隔符分割列

eg:#cut -f  2  aa.txt   提取第二列

eg:#cut -d ":" -f 1,3 /etc/passwd  以:分割，提取第1和第3列

eg:#cat /etc/passwd | grep /bin/bash | grep -v root | cut -d ":" -f 1    获取所有可登陆的普通用户用户名

cut的局限性    不能分割空格   df -h  不能使用cut分割

df -h | grep sda1 | cut -f 5

**awk**

一个强大的文本分析工具

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。

语法：awk ‘条件1{动作1}条件2{动作2}...’文件名

条件（Pattern）:

一般使用关系表达式作为条件： >   >=  <=等

动作（Action）：

格式化输出

流程控制语句

eg:#df -h | awk '{print $1 "\t" $3}'      显示第一列和第三列

**FS内置变量**

eg:# cat /etc/passwd | grep "/bin/bash" | awk 'BEGIN {FS=":"} {print $1 "\t"$3 }'    输出可登陆用户的用户名和UID,这里使用FS内置变量指定分隔符为：,而且使用BEGIN保证第一行也操作，因为awk命令会在读取第一行后再执行条件

**指定分隔符还可以用-F更简单**

eg:# cat /etc/passwd | grep "/bin/bash" | awk -F:  '{print $1 "\t"$3 }'   效果同上

eg:判断一下根目录的使用情况

#df -h |grep sda1 | awk '{print $5}' | awk -F% '{print $1} $1<80{print "info"}$1>80{print "warning"}'

**BEGIN 在所有数据读取之前执行**

eg：#awk 'BEGIN {printf "first Line \n"} {printf $2 }' aa.txt     在输出之前使用BEGIN输出内容

**END 在所有数据执行之后执行**

eg：#awk 'END {printf "The End \n"} {print $2}' aa.txt   所有命令执行完后，输出一句"The End"

# df -h | grep sda2 | awk '{print $5}' | awk -F% '{print $1}'

# df -h | grep sda2 | awk '{print $5}' | cut -d"%" -f 1

获取所有用户信息里的用户名：

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

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

获取当前机器的ip地址：

# ifconfig eth0 | grep 'inet addr' | awk -F: '{print $2}' | awk '{print $1}'

sed： stream editor

s e d是一个非交互性文本**流**编辑器。它编辑文件或标准输入导出的文本拷贝。标准输入可 能是来自键盘、文件重定向、字符串或变量，或者是一个管道的文本。

注意： s e d 并不与初始化文件打交道， 它操作的只是一个拷贝，然后所有的改动如果没有重定向到一个文件，将输出到屏幕。

语法：sed [选项]’[动作]’ 文件名

常用选项:

-n　　使用安静（silent）模式。显示经过sed特殊处理的数据。

-e　　允许多点编辑。

-i　　直接修改读取的档案内容，而不是由屏幕输出。

|  |  |
| --- | --- |
| 命令 | 功能描述 |
| a\ | 新增， a 的后面可以接字串，在下一行出现 |
| c\ | 替换 |
| d | 删除 |
| i\ | 插入， i 的后面可以接字串 |
| p | 打印 |
| s | 查找并替换，例如 1,20s/old/new/g |

eg:

sed '2p' sed.txt 显示第二行和所有数据

sed -n '2,3p' sed.txt 显示第二和第三行

df -h | sed -n '1p' 接收命令结果数据

sed ‘2a liuyifei’sed.txt 在第二行后面添加数据

sed ‘4i fengjie \

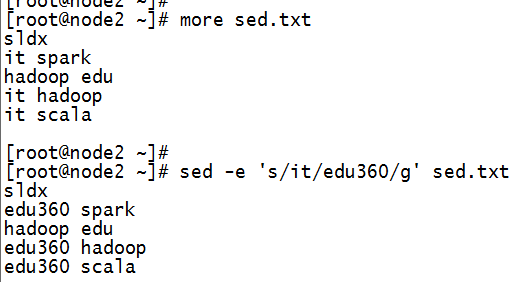
canglaoshi’sed.txt 在第4行之前添加两行数据

sed ‘2c this is replace’sed.txt 替换第二行数据

sed ‘s/it/edu360/g’ sed.txt 把sed.txt文件中的it替换为edu360,并输出

sed -e '1s/1/34/g;3s/yangmi//g' sed.txt 同时进行多个替换

sed –i ‘s/it/edu360/g’ sed.txt 要想真正替换，需要使用-i参数



使用sed获取机器的ip地址

ifconfig eth0 | grep 'inet addr'| sed 's/^.\*addr://g' | sed 's/ Bcast.\*$//g'

# 定时器

crontab 命令允许用户提交、编辑或删除相应的作业。每一个用户都可以有一个crontab 文件来保存调度信息。可以使用它运行任意一个 s h e l l 脚本或某个命令。

crontab命令格式

作用：用于生成cron进程所需要的crontab文件  
crontab的命令格式

# crontab -e

使用编辑器编辑当前的crontab文件。

  crontab文件格式  
minute hour  day-of-month  month-of-year  day-of-week  commands

分< >时< >日< >月< >星期< >要运行的命令     < >表示空格

其中  
    Minute              一小时中的哪一分钟     [0～59]   
    hour                  一天中的哪个小时        [0～23]   
    day-of-month    一月中的哪一天           [1～31]   
    month-of-year   一年中的哪一月           [1～12]   
    day-of-week      一周中的哪一天           [0～6]     0表示星期天  
    commands         执行的命令

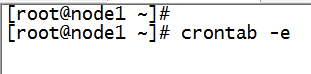
书写注意事项  
1,全都不能为空，必须填入，不知道的值使用通配符\*表示任何时间  
2,每个时间字段都可以指定多个值，不连续的值用,间隔，连续的值用-间隔。  
3,命令应该给出绝对路径  
4,用户必须具有运行所对应的命令或程序的权限

如何使用crontab 运行多个任务:

方法1：在crontab -e 里 写多个

输入命令 crontab –e

敲回车



开始编写任务：



方法2：把所有的任务，写入到一个可执行的文件

再在crontab -e里面配置执行任务



分钟   小时   天   月   星期   命令/脚本

示例：

eg:4点备份

0      4        \*      \*     \*

eg:每周二，周五，下午6点  的计划任务

0     18       \*      \*     2,5

eg:1到3月份，每周二周五，下午6点的计划任务

0     18       \*     1-3   2,5

eg:周一到周五下午，5点半提醒学生15分钟后关机

30   17      \*       \*     1-5   /usr/bin/wall < /etc/issue

45   17      \*       \*     1-5  /sbin/shutdown -h now

eg:学校的计划任务， 12点14点，检查apache服务是否启动

\*/2  12-14  \*   3-6,9-12   1-5

eg:再添加一个备份，把/etc目录备份到/backup下，然后把错误的情况也记录下来，正确的文件都丢到/dev/null下，看不见（相当于一个黑洞）

\*/2  12-14  \*   3-6,9-12   1-5   /bin/cp -r /etc  /backup/etc.20170407 2> /backup/etc.bak.err

/dev/null

eg:每月 1、1 0、2 2日的4:45运行/apps/bin目录下的backup.sh

45  4  1,10,22  \*  \* /apps/bin/backup.sh

eg:每周六、周日的 1 : 10运行一个find命令

10  1  \*  \*  6,0  /bin/find -name "core" -exec rm {} \;

eg:在每天 18:00至23 :00之间每隔30分钟运行/apps/bin目录下的dbcheck.sh

0,30  18-23  \*  \*  \* /apps/bin/dbcheck.sh

eg:每星期六的 11:00 pm运行/apps/bin目录下的qtrend.sh

0  23  \*  \*  6  /apps/bin/qtrend.sh