目录

[第一部分 bash简介 2](#_Toc501631930)

[第二部分 bash示例和书写流程 3](#_Toc501631931)

[第三部分 bash基础语法 4](#_Toc501631932)

[1 条件判断 4](#_Toc501631933)

[1.1 test判断语句 4](#_Toc501631934)

[1.2 []条件判断 5](#_Toc501631935)

[1.3 测试逻辑表达式 7](#_Toc501631936)

[2 if then else语句 7](#_Toc501631937)

[3 case语句 9](#_Toc501631938)

[4 for循环 10](#_Toc501631939)

[5 until循环 11](#_Toc501631940)

[6 while循环 12](#_Toc501631941)

[7 使用break和continue控制循环 13](#_Toc501631942)

[第四部分 bash数组 14](#_Toc501631943)

[1 数组定义 14](#_Toc501631944)

[2 数组操作 15](#_Toc501631945)

[第五部分 Bash的函数 16](#_Toc501631946)

[1 函数定义 16](#_Toc501631947)

[2 函数调用和传参 17](#_Toc501631948)

[3 函数返回值 17](#_Toc501631949)

[4 应用实例 17](#_Toc501631950)

[第六部分 数值运算 18](#_Toc501631951)

[第七部分 字符运算 20](#_Toc501631952)

[1 字符串定义 20](#_Toc501631953)

[2 字符串操作 21](#_Toc501631954)

[2.1 string操作公式表 21](#_Toc501631955)

[2.2 应用实例 21](#_Toc501631956)

[第八部分 bash自带参数 22](#_Toc501631957)

[第九部分 bash调试 22](#_Toc501631958)

[1 bash命令调试 22](#_Toc501631959)

[第十部分 bash注释 23](#_Toc501631960)

[1 单行注释 23](#_Toc501631961)

[2 多行注释 23](#_Toc501631962)

[第十一部分 bash内建指令 23](#_Toc501631963)

[1 内建命令查看方法 23](#_Toc501631964)

[2 常用内建命令说明 24](#_Toc501631965)

[第十二部分 bash实例 25](#_Toc501631966)

[**第十三部分crontab（计时器）用法详解** 26](#_Toc501631967)

# **第一部分 bash简介**

在介绍bash之前，需要先介绍它的起源——shell。shell俗称壳，它是指UNIX系统下的一个命令解析器；主要用于用户和系统的交互。UNIX系统上有很多种Shell。首个shell，即Bourne Shell，于1978年在V7(AT&T的第7版)UNIX上推出。后来，又演变出C shell、bash等不同版本的shell。

bash，全称为Bourne-Again Shell。它是一个为GNU项目编写的Unix shell。bash脚本功能非常强大，尤其是在处理自动循环或大的任务方面可节省大量的时间。bash是许多Linux平台的内定Shell，这也是我们介绍它主要的原因。

Shell 编程跟 java、php 编程一样，只要有一个能编写代码的文本编辑器和一个能解释执行的脚本解释器就可以了。

Linux 的 Shell 种类众多，常见的有：

Bourne Shell（/usr/bin/sh或/bin/sh）

Bourne Again Shell（/bin/bash）

C Shell（/usr/bin/csh）

K Shell（/usr/bin/ksh）

Shell for Root（/sbin/sh）

……

本教程关注的是 Bash，也就是 Bourne Again Shell，由于易用和免费，Bash 在日常工作中被广泛使用。同时，Bash 也是大多数Linux 系统默认的 Shell。

在一般情况下，人们并不区分 Bourne Shell 和 Bourne Again Shell，所以，像 #!/bin/sh，它同样也可以改为 #!/bin/bash。

#! 告诉系统其后路径所指定的程序即是解释此脚本文件的 Shell 程序。

sh/bash/csh/Tcsh/ksh/pdksh等shell的区别

sh(全称 Bourne Shell): 是UNIX最初使用的 shell，而且在每种 UNIX 上都可以使用。

Bourne Shell 在 shell 编程方面相当优秀，但在处理与用户的交互方面做得不如其他几种 shell。

bash（全称 Bourne Again Shell）: LinuxOS 默认的，它是 Bourne Shell 的扩展。

与 Bourne Shell 完全兼容，并且在 Bourne Shell 的基础上增加了很多特性。可以提供命令补全，命令编辑和命令历史等功能。它还包含了很多 C Shell 和 Korn Shell 中的优点，有灵活和强大的编辑接口，同时又很友好的用户界面。

csh(全称 C Shell): 是一种比 Bourne Shell更适合的变种 Shell，它的语法与 C 语言很相似。

Tcsh: 是 Linux 提供的 C Shell 的一个扩展版本。

Tcsh 包括命令行编辑，可编程单词补全，拼写校正，历史命令替换，作业控制和类似 C 语言的语法，他不仅和 Bash Shell 提示符兼容，而且还提供比 Bash Shell 更多的提示符参数。

ksh (全称 Korn Shell): 集合了 C Shell 和 Bourne Shell 的优点并且和 Bourne Shell 完全兼容。

pdksh: 是 Linux 系统提供的 ksh 的扩展。

pdksh 支持人物控制，可以在命令行上挂起，后台执行，唤醒或终止程序。

# **第二部分 bash示例和书写流程**

运行 Shell 脚本有两种方法：

1、作为可执行程序

代码保存为 test.sh，并 cd 到相应目录：

chmod +x ./test.sh #使脚本具有执行权限

./test.sh #执行脚本

注意，一定要写成 ./test.sh，而不是 test.sh，运行其它二进制的程序也一样，直接写 test.sh，linux 系统会去 PATH 里寻找有没有叫 test.sh 的，而只有 /bin, /sbin, /usr/bin，/usr/sbin 等在 PATH 里，你的当前目录通常不在 PATH 里，所以写成 test.sh 是会找不到命令的，要用 ./test.sh 告诉系统说，就在当前目录找。

2、作为解释器参数

这种运行方式是，直接运行解释器，其参数就是 shell 脚本的文件名，如：

/bin/sh test.sh

/bin/php test.php

这种方式运行的脚本，不需要在第一行指定解释器信息，写了也没用。

1 新建文件test.sh

$ **touch test.sh**

2 添加可执行权限

$ **chmod +x test.sh**

3 编辑test.sh，test.sh内容如下：

**#!/bin/bash**

**echo "hello bash"**

**exit 0**

说明：

#!/bin/bash : 它是bash文件声明语句，表示是以/bin/bash程序执行该文件。它必须写在文件的第一行！

echo "hello bash" : 表示在终端输出“hello bash”

exit 0 : 表示返回0。在bash中，0表示执行成功，其他表示失败。

4 执行bash脚本

$ **./test.sh**

在终端输出”hello bash”

# **第三部分 bash基础语法**

## 1 条件判断

条件判断有2中格式，分别是“**test EXPRESSION**”和“**[ EXPRESSION ]**”。

“test”判断语句，在实际中应用的比较少；相反的，“[]”判断语句应用很广。下面分别对它们进行介绍

### 1.1 test判断语句

**基本格式**

**test EXPRESSION**

**格式说明**

test是关键字，表示判断；

EXPRESSION是被判断的语句。

关于EXPRESSION的说明，参考如下：



**应用实例**

一、判断文件/home/skywang/123.txt是不是文件

$ **test -f /home/skywang/123.txt**

**其他说明**

在linux系统下，可以通过以下命令去测试上面的实例  
# 切换到执行目录(如切换到/home/skywang)

$ **cd /home/skywang**

# 判断123.txt是不是文件

$ **test -f 123.txt**

# 输出判断结果，0表示成功，其他表示失败。

$ **echo $?**

# 其中，echo表示输出文本到当前终端，$?表示上一命令的执行结果(在linux中bash中，命令成功返回0,失败返回其他)。

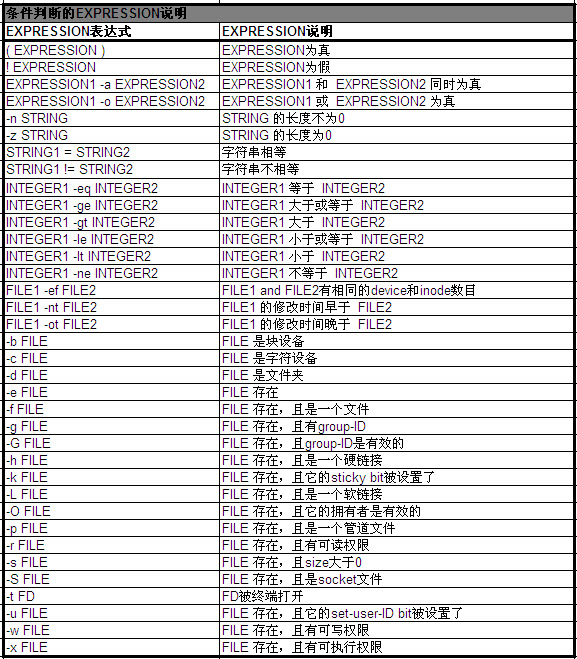
### 1.2 []条件判断

**基本格式**

**[ EXPRESSION ]**

**格式说明**  
中括号的左右扩弧和EXPRESSION之间都必须有空格！

关于EXPRESSION的说明，参考如下：



**应用实例**  
一、判断文件/home/skywang/123.txt是否存在

$ **[ -f /home/skywang/123.txt ]**

二、判断变量val是否等于字符串“skywang”

$ **[ "$val" = "skywang" ]**

三，判断变量num是否等于数字100

$ **[ "$num" -eq "100" ]**

### **1.3 测试逻辑表达式**

**基本格式**

-a : 逻辑与,操作符两边均为真,结果为真,否则为假。

-o : 逻辑或,操作符两边一边为真,结果为真,否则为假。

! : 逻辑否,条件为假,结果为真。

**应用实例**  
一、判断文件123.txt是不是可读写

$ **[ -r 123.txt -a -w 123.txt ]**

等价于

$ **[ -r 123.txt ] && [ -w 123.txt ]**

二、判断变量num是不是等于数字101或102

$ **[ "$num" -eq "101" -o "$num" -eq "102" ]**

等价于

$ **[ -r 123.txt ] || [ -w 123.txt ]**

三、判断文件123.txt是不可读

$ **[ ! -r 123.txt ]**

## 2 if then else语句

**基本格式**

**if 条件1 ;**

**then 命令1**

**elif 条件2**

**then 命令2**

**else 命令3**

**fi**

**格式说明**

if 语句必须以单词 fi 终止。elif 和 else 为可选项,如果语句中没有否则部分,那么就不需要 elif 和 else 部分。if 语句可以有许多 elif 部分。最常用的 if 语句是 if then fi 结构。

**应用实例**

一、判断文件文件123.txt是否存在，存在则输出“file exist”；没有则输出“file not exist”。bash文件内容如下：

#!/bin/bash

if [ -f 123.txt ];then

echo "file exist"

else

echo "file not exist"

fi

exit 0

二、提示用户输入值。若输入的值小于0，则输出“negtive number”；若等于0,则输出“number zero”，否则，输出“positive number”。bash文件内容如下：

#!/bin/bash

# 提示用户输入一个值

echo -n "please input a number:"

# 保存用户输入的值到num中

read num

if [ "$num" -lt "0" ];then

# 小于0,则输出“negtive number”

echo "negtive number"

elif [ "$num" -gt "0" ];then

# 大于0,则输出“positive number”

echo "positive number"

else

# 大于0,则输出"number zero"

echo "number zero"

fi

exit 0

## 3 case语句

case语句为多选择语句。可以用case语句匹配一个值与一个模式,如果匹配成功,执行相匹配的命令。

**基本格式**

**case 值 in**

**模式1}**

**命令1**

**...**

**;;**

**模式2)**

**命令2**

**...**

**;;**

Esac

**格式说明**

“模式”部分可能包括元字符，即:

\* 任意字符。

? 任意单字符。

[..] 类或范围中任意字符

**应用实例**

一、提示用户输入Y/y或N/n。若输入Y/y，则输出“yes”；若输入N/n,则输出“no”；否则，“incorrect input”。bash文件内容如下：

#!/bin/bash

# 提示用户输入一个值

echo -n "are you femail(Y/N):"

# 保存用户输入的值到val中

read val

case $val in

Y|y)

# 用户输入Y或y

echo "yes"

;;

N|n)

# 用户输入N或n

"no"

;;

\*)

# 其它输入

echo "incorrect input"

;;

esac

exit 0

## 4 for循环

**基本格式**

**for 变量名in列表**

**do**

**命令1**

**命令2...**

**Done**

**格式说明**

当变量值在列表里, for循环即执行一次所有命令,使用变量名访问列表中取值。命令可为任何有效的 shell命令和语句。变量名为任何单词。 in列表用法是可选的,如果不用它, for循环使用命令行的位置参数。

**应用实例**  
一、输入当前文件夹的一级子目录中文件名字。bash脚本内容如下：

#!/bin/bash

# 将ls的结果保存到变量CUR\_DIR中

CUR\_DIR=`ls`

# 显示ls的结果

echo $CUR\_DIR

for val in $CUR\_DIR

do

# 若val是文件，则输出该文件名

if [ -f $val ];then

echo "FILE: $val"

fi

done

exit 0

二、输出1-10之间数字的总和。bash脚本内容如下：

#!/bin/bash

sum=0

for ((i=1;i<10;i++))

do

((sum=$sum+$i))

done

echo "sum=$sum"

exit 0

## 5 until循环

until循环执行一系列命令直至条件为真时停止。until循环与 while循环在处理方式上刚好相反。一般 while循环优于until循环,但在某些时候 — 也只是极少数情况下, until循环更加有用。

**基本格式**

**until 条件**

**命令1**

**...**

**done**

**应用实例**

一、从0开始逐步递增，当数值等于5时，停止递增。Bash脚本内容如下：

#!/bin/bash

# 设置起始值为0

val=0

until [ "$val" -eq "5" ]

do

# 输出数值

echo "val=$val"

# 将数值加1

((val++))

done

exit 0

## 6 while循环

**基本格式**

**while 命令**

**do**

**命令1**

**命令2**

**...**

**Done**

**应用实例**

一、从0开始逐步递增，当数值等于5时，停止递增。Bash脚本内容如下：

#!/bin/bash

# 设置起始值为0

val=0

while [ "$val" -lt "5" ]

do

# 输出数值

echo "val=$val"

# 将数值加1

((val++))

done

exit 0

## 7 使用break和continue控制循环

**基本格式**

break命令允许跳出循环。

continue命令类似于 break命令,只有一点重要差别,它不会跳出循环,只是跳过这个循环步。

**应用实例**

一、[break应用]从0开始逐步递增，当数值等于5时，停止递增。Bash脚本内容如下：

#!/bin/bash

# 设置起始值为0

val=0

while true

do

if [ "$val" -eq "5" ];then

# 如果val=5，则跳出循环

break;

else

# 输出数值

echo "val=$val"

# 将数值加1

((val++))

fi

done

exit 0

二、[continue应用]从0开始逐步递增到10：当数值为5时，将数值递增2；否则，输出数值。Bash脚本内容如下：

#!/bin/bash

# 设置起始值为0

val=0

while [ "$val" -le "10" ]

do

if [ "$val" -eq "5" ];then

# 如果val=5，则将数值加2

((val=$val+2))

continue;

else

# 输出数值

echo "val=$val"

# 将数值加1

((val++))

fi

done

exit 0

# **第四部分 bash数组**

## 1 数组定义

1. array=(10 20 30 40 50)

一对括号表示是数组，数组元素用“空格”符号分割开。引用数组时从序号0开始。

2. 除了上面的定义方式外，也可以单独定义数组：

array[0]=10

array[1]=20

array[2]=30

array[3]=40

array[4]=50

3. var="10 20 30 40 50"; array=($var)

## 2 数组操作

(01) 显示数组中第2项

$ **echo ${array[i]}**

说明：数组是从序号0开始计算(即第1项为array[0])。

(02) 显示数组中的所有元素

$ **echo ${array[@]}**

或者

$ **echo ${array[\*]}**

(03) 显示数组长度

$ **echo ${#array[@]}**

或者

$ **echo ${#array[\*]}**

(04) 删除数组第2项元素

$ **unset array[1]**

说明：

unset仅仅只清除array[1]的值，并没有将array[1]删除掉

(05) 删除整个数组

$ **unset array**

(06) 输出数组的第1-3项

$ **echo ${array[@]:0:3}**

说明：

参考“${数组名[@或\*]:起始位置:长度}”

(07) 将数组中的0替换成1

$ **echo ${a[@]/0/1}**

说明：

${数组名[@或\*]/查找字符/替换字符

# **第五部分 Bash的函数**

## 1 函数定义

**基本格式**

**function 函数名()**

**{**

**...**

**}**

**格式说明**  
function可有可无。但建议保留，因为保留的话看起来更加直观。

**应用实例**

function foo()

{

# 定义局部变量i

local i=0

# 定义局部变量total=传入foo的参数总数

local total=$#

# 输出参数总数

echo "total param =$total"

# 输出传入foo的每一个参数

for val in $@

do

((i++))

echo "$i -- val=$val"

done

# 返回参数总数

return $total

}

## 2 函数调用和传参

**调用方法**

直接通过函数名去调用。

**应用实例**

**foo param1 param2 param3**

说明：

调用函数foo，并传入param1 param2 param3三个参数

## 3 函数返回值

**使用方法**

**return 返回值**

**方法说明**

例如，foo param1 param2 param3之后，再调用$?就是上次调用的返回值

## 4 应用实例

编辑一个函数foo：打印foo的输入参数的总数，并输入每个参数和参数对应的序号。

#!/bin/bash

function foo()

{

# 定义局部变量i

local i=0

# 定义局部变量total=传入foo的参数总数

local total=$#

# 输出参数总数

echo "total param =$total"

# 输出传入foo的每一个参数

for val in $@

do

((i++))

echo "$i -- val=$val"

done

# 返回参数总数

return $total

}

foo

foo param1 param2 param3

# 输出foo param1 param2 param3的返回值

echo "return value=$?"

exit 0

**输出结果**:

total param =0

total param =3

1 -- val=param1

2 -- val=param2

3 -- val=param3

return value=3

# **第六部分 数值运算**

数值比较请参考"第三部分"的1.2节，本部分只介绍数值运算。

常用的4种数值运算说明

数值元算主要有4种实现方式：(())、let、expr、bc。

工作效率：  
**(()) == let > expr > bc**

(01), (())和let是bash内建命令，执行效率高；而expr和bc是系统命令，会消耗内存，执行效率低。

(02), (())、let和expr只支持整数运算，不支持浮点运算；而bc支持浮点运算。

下面分别介绍这4种实现方式的使用方法。

应用实例一：分别用上面四种方式实现"3\*(5+2)"。

#!/bin/bash

# 数值i=3\*(5+2) (方法一:$(())实现)

val=$((3\*(5+2)))

echo "val=$val"

# 数值i=3\*(5+2) (方法二:let实现)

let "val=3\*(5+2)"

echo "val=$val"

# 数值i=3\*(5+2) (方法三:expr实现)

val=`expr 3 \\* \( 5 + 2 \)`

echo "val=$val"

# 数值i=3\*(5+2) (方法四:bc实现)

val=`echo "3\*(5+2)"|bc`

echo "val=$val"

exit 0

应用实例二：分别勇上面四种方式实现“数值+1”。

#!/bin/bash

val=0

# 数值加1 (方法一)

((val++))

echo "val=$val"

val=0

# 数值加1 (方法二)

let val++

echo "val=$val"

val=0

# 数值加1 (方法三)

val=`expr $val + 1`

echo "val=$val"

val=0

# 数值加1(方法四)

val=`echo "$val+1"|bc`

echo "val=$val"

exit 0

应用实例三：计算1/3,保留3位小数。

#!/bin/bash

# 数值i=3\*(5+2) (方法四:bc实现)

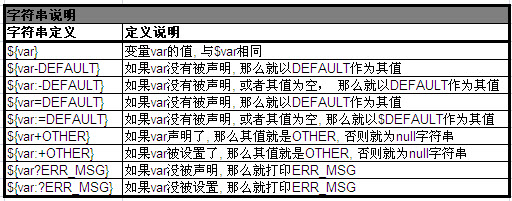
val=`echo "scale=3; 1/3"|bc`

echo "val=$val"

exit 0

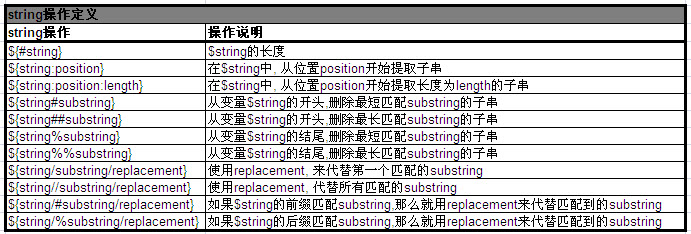
# **第七部分 字符运算**

## 1 字符串定义



## 2 字符串操作

### 2.1 string操作公式表



### 2.2 应用实例

首先，我们定义个str字符串变量，然后再对此变量进行字符串操作。

$ **str="hello world"**

(01) 显示字符串长度

$ **echo ${#str}**

(02) 提取world

$ **echo ${str:6}**

(03) 提取or

$ **echo ${str:7:2}**

(04) 删除hello

$ **echo ${str#hello}**

或

$ **echo ${str#\*lo}**

(05) 删除world

$ **echo ${str%world}**

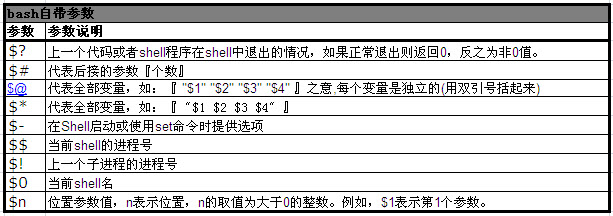
或

$ **echo ${str%wo\*}**

(06) 将所有的字符“l”替换为“m”

$ **echo ${str//l/m}**

# **第八部分 bash自带参数**



# **第九部分 bash调试**

## 1 bash命令调试

**bash [-nvx] scripts.sh**

选项与参数:

-n :不要执行 script,仅查询语法的问题;  
-v :再执行 sccript 前,先将 scripts 的内容输出到屏幕上;

-x :将使用到的 script 内容显示到屏幕上,这是很有用的参数!

例如，想要执行bash脚本，并查看bash的调用流程，可以通过以下命令：

$ **bash -x test.sh**

2 echo调试

**echo [OPTION] STRING**  
-n : 输出内容之后，不换行。默认是输入内容之后，换行。

-e : 开启反斜线“\”转义功能

-E : 开启反斜线“\”转义功能（默认）。

例如，输出“please input a number:”之后不换行。

$ **echo -n "please input a number:"**

3 printf

和echo一样，printf也能用于输出。语法格式和C语言中printf一样。

例如，输出“hello printf”之后换行。

$ **printf "hello printf\n"**

# **第十部分 bash注释**

## 1 单行注释

**# echo "single line"**

说明：#放在文件开头，表示注释掉本行。

## 2 多行注释

可以通过以下两种方法实现多行注释：

**:||{**

**....被注释的多行内容**

**}**

或者

**if false ; then**

**....被注释的多行内容**

**fi**

# **第十一部分 bash内建指令**

## 1 内建命令查看方法

基本格式

**type cmd**

格式说明

type是命令关键字，cmd表示查看的命令；若输出builtin，则该命令是bash的内建命令。例如:

$ **type echo**

除此之外，用户也可以通过man bash或者man builtins查看bash的全部内置命令.

## 2 常用内建命令说明

（01）echo

命令：echo arg

功能：在屏幕上显示出由arg指定的字串  
（02）read

命令格式：read变量名表

功能：从标准输入设备读入一行，分解成若干字，赋值给bash程序内部定义的变量

（03）shift

命令：shift [N] (N为大于0的整数；当N省略时，等价与于“shift 1”)

功能：所有的参数依次向左移动N个位置，并使用$#减少N，直到$#=0为止。

（04）alias

命令：alias name='value'

功能：别名。用name替换value，value要用单引号括住。

（05）export

命令：export变量名[=变量值]

功能：export可以把bash的变量向下带入子bash(即子bash中可以使用父bash的变量)，从而让子进程继承父进程中的环境变量。但子bash不能用export把它的变量向上带入父bash。

（06）readonly

命令：readonly 变量名

功能：定义只读变量。不带任何参数的readonly命令将显示出所有只读变量。  
（07）exec

命令：exec 命令参数

功能：当bash执行到exec语句时，不会去创建新的子进程，而是转去执行指定的命令，当指定的命令执行完时，该进程（也就是最初的bash）就终止了，所以bash程序中exec后面的语句将不再

被执行。

（08）"."(点)

命令：. bash程序文件名

功能：使bash读入指定的bash程序文件并依次执行文件中的所有语句。

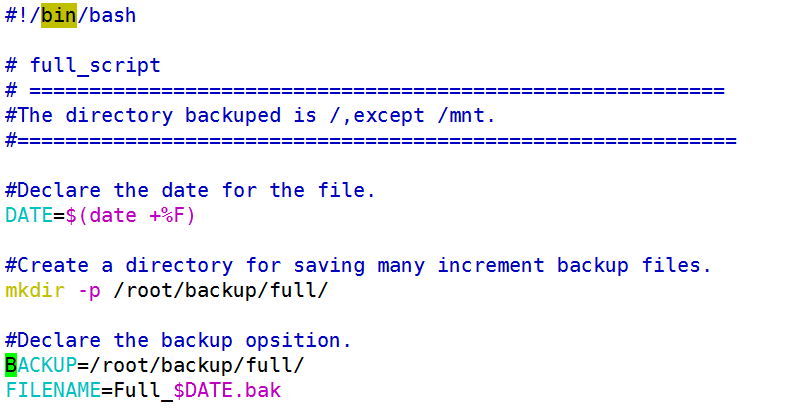
（09）exit

命令：exit N

功能：退出Shell程序。在exit之后可有选择地指定一个数位作为返回状态。

# **第十二部分 bash实例**

定时备份指定目录内容，不包括/mnt里挂接的外部系统文件，一个配置文件和一个脚本文件，备份的文件名为Full\_[日期时间].bak



#!/bin/bash

# full\_script

# ==========================================================

#The directory backuped is /,except /mnt.

#============================================================

#Declare the date for the file.

DATE=$(date +%F)

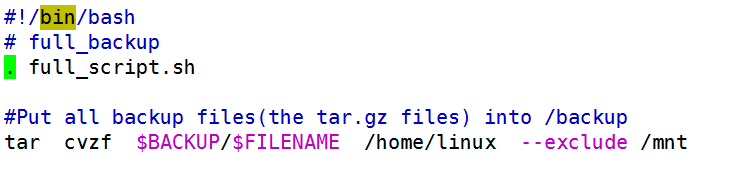
#Create a directory for saving many increment backup files.

mkdir -p /root/backup/full/

#Declare the backup opsition.

BACKUP=/root/backup/full/

FILENAME=Full\_$DATE.bak



#!/bin/bash

# full\_backup

. full\_script.sh

#Put all backup files(the tar.gz files) into /backup

tar cvzf $BACKUP/$FILENAME /home/linux --exclude /mnt

# **第十三部分**[**crontab（计时器）用法详解**](http://www.cnblogs.com/ccdc/archive/2012/06/01/2529471.html)

**关于crontab**：

crontab命令常见于Unix和类Unix的操作系统之中，用于设置周期性被执行的指令。该命令从标准输入设备读取指令，并将其存放于“crontab”文件中，以供之后读取和执行。该词来源于希腊语 chronos(χρ?νο?)，原意是时间。

　　通常，crontab储存的指令被守护进程激活， crond常常在后台运行，每一分钟检查是否有预定的作业需要执行。这类作业一般称为cron jobs。

**安装crontab:**

[root@CentOS ~]# yum install vixie-cron

[root@CentOS ~]# yum install crontabs

**说明：**  
vixie-cron软件包是cron的主程序；

crontabs软件包是用来安装、卸装、或列举用来驱动 cron 守护进程的表格的程序。

cron 是linux的内置服务，但它不自动起来，可以用以下的方法启动、关闭这个服务：

/sbin/service crond start #启动服务

/sbin/service crond stop #关闭服务

/sbin/service crond restart #重启服务

/sbin/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

**其他命令：**

#查看crontab服务是否已设置为开机启动，执行命令：

ntsysv

#加入开机自动启动:

chkconfig --level 35 crond on

#列出crontab文件

crontab -l

#编辑crontab文件

crontab -e

#删除crontab文件

$ crontab -r

#恢复丢失的crontab文件

#假设你在自己的$HOME目录下还有一个备份，那么可以将其拷贝到/var/spool/cron/<username>，其中<username >是用户名

#或者使用如下命令其中，<filename>是你在$HOME目录中副本的文件名

crontab <filename>

**日志文件：**/var/log/cron\*

**补充：**

**1、crontab相关命令**

**功能说明：**设置计时器。

**语　　法：**crontab [-u <用户名称>][配置文件] 或 crontab [-u <用户名称>][-elr]

**补充说明：**cron是一个常驻服务，它提供计时器的功能，让用户在特定的时间得以执行预设的指令或程序。只要用户会编辑计时器的配置文件，就可以使用计时器的功能。

**配置文件格式：**Minute Hour Day Month DayOFWeek Command

**参　　数：**

-e 　编辑该用户的计时器设置。

-l 　列出该用户的计时器设置。

-r 　删除该用户的计时器设置。

-u<用户名称> 　指定要设定计时器的用户名称。

**2、crontab 配置文件格式**

**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：**

#每晚的21:30 重启apache

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

#每月1、10、22日的4 : 45重启apache

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

#每周六、周日的1 : 10重启apache

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

#每天18 : 00至23 : 00之间每隔30分钟重启apache

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

#每星期六的11 : 00 pm重启apache

0 23 \* \* 6 /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

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

#每一小时重启apache

\* \*/1 \* \* \* /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 11 4 \* mon-wed /usr/local/etc/rc.d/lighttpd restart

#一月一号的4点重启apache

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

#每半小时同步一下时间

\*/30 \* \* \* \* /usr/sbin/ntpdate 210.72.145.44

**3、其他任务调度**

cron默认配置了调度任务，分别为：hourly、daily、weekly、mouthly，默认配置文件为/etc/anacrontab

将需要执行的脚本放到相应的目录下即可，目录分别为：

/etc/cron.hourly

/etc/cron.daily

/etc/cron.weekly

/ect/cron.mouthly