目录

[一、 shell脚本编程基础 2](#_Toc489953204)

[1. 构建基本脚本 3](#_Toc489953205)

[（1） 创建shell脚本文件 3](#_Toc489953206)

[（2）显示消息 3](#_Toc489953207)

[（3）使用变量 3](#_Toc489953208)

[（4）反引号 3](#_Toc489953209)

[（5）输出重定向 3](#_Toc489953210)

[（6）输入重定向 4](#_Toc489953211)

[（7）管道 4](#_Toc489953212)

[（8）执行数学运算（整数） 4](#_Toc489953213)

[（10）退出脚本 4](#_Toc489953214)

[2. 使用结构化命令 5](#_Toc489953215)

[（1） 使用 if-then语句 5](#_Toc489953216)

[（2） if -then –else语句 5](#_Toc489953217)

[（3） 嵌套if 5](#_Toc489953218)

[（4） test命令 6](#_Toc489953219)

[（5） 复合条件测试 7](#_Toc489953220)

[（6） if-then的高级特性 7](#_Toc489953221)

[（7） case命令 8](#_Toc489953222)

[3.更多的结构化命令 8](#_Toc489953223)

[（1） for命令 8](#_Toc489953224)

[（2） C语言风格的for命令 9](#_Toc489953225)

[（3）while命令 9](#_Toc489953226)

[(4) until 命令 9](#_Toc489953227)

[（5）嵌套循环 10](#_Toc489953228)

[（6）循环处理文件数据 10](#_Toc489953229)

[（7）控制循环 10](#_Toc489953230)

[4.处理用户输入 10](#_Toc489953231)

[（1） 命令行参数 10](#_Toc489953232)

[(2) 特殊参数变量 11](#_Toc489953233)

[(3)移动变量： 11](#_Toc489953234)

[(4)处理选项 11](#_Toc489953235)

[（5）将选项标准化 12](#_Toc489953236)

[（6）获得用户输入 13](#_Toc489953237)

[5．呈现数据 13](#_Toc489953238)

[(1) 理解输入和输出 13](#_Toc489953239)

[(2) 在脚本中重定向输出 14](#_Toc489953240)

[（3）在脚本重定向输入 14](#_Toc489953241)

[（4）创建自己的重定向 15](#_Toc489953242)

[(5)列出打开的文件描述符 15](#_Toc489953243)

[（6）阻止命令输出 16](#_Toc489953244)

[（7）创建临时文件 16](#_Toc489953245)

[(8) 记录消息 17](#_Toc489953246)

[6 控制脚本 17](#_Toc489953247)

[（1） 处理信号 17](#_Toc489953248)

[(2) 以后台模式运行脚本 18](#_Toc489953249)

[（3）在非控制台下运行脚本 19](#_Toc489953250)

[（4）作业控制 19](#_Toc489953251)

[(5) 调整谦让度 20](#_Toc489953252)

[（7）启动时运行 22](#_Toc489953253)

[二、 高级shell脚本编程 23](#_Toc489953254)

[1. 创建函数 23](#_Toc489953255)

[（1） 基本的脚本函数 23](#_Toc489953256)

[（2） 返回值 24](#_Toc489953257)

[（3）在函数中使用变量 24](#_Toc489953258)

[（4） 数组变量和函数 25](#_Toc489953259)

[（5）函数递归 25](#_Toc489953260)

[（6）创建库 25](#_Toc489953261)

[(7) 在命令行上使用函数 25](#_Toc489953262)

[2. 图形化桌面上的脚本编程 26](#_Toc489953263)

[(1) 创建文本菜单 26](#_Toc489953264)

[（2） 使用窗口 27](#_Toc489953265)

[（3）使用图形 27](#_Toc489953266)

[3. 初识sed和gawk 28](#_Toc489953267)

[(1) 文本处理 28](#_Toc489953268)

[（2） sed编辑器基础 31](#_Toc489953269)

[4.正则表达式 31](#_Toc489953270)

[5.sed进阶 31](#_Toc489953271)

[6.gawk进阶 31](#_Toc489953272)

[7.使用其他shell 31](#_Toc489953273)

[三、 高级shell脚本编程主题 31](#_Toc489953274)

[1.使用数据库 31](#_Toc489953275)

[2.使用web 31](#_Toc489953276)

[3.使用E-mail 31](#_Toc489953277)

[4.编写脚本是用工具 31](#_Toc489953278)

[5.shell脚本编程进阶 31](#_Toc489953279)

# shell脚本编程基础

## 构建基本脚本

### 创建shell脚本文件

shell脚本里， #用作注释；

文件的第一行格式为: #!/bin/bash (!后面指定要使用的shell,此处为bash);

### （2）显示消息

echo: 显示一个简单的文本字符串

echo –n： 在同一行显示一个文本字符串作为命令输出

### （3）使用变量

使用环境变量：在环境变量名称前加$使用

eg: echo $UID

显示美元字符： \&

使用用户变量：

变量名不可超过20个字符，由字母，数字，下划线组成，区分大小写；

变量赋值： =；

变量名和等号之间不允许有空格；

shell会自动决定变量的类型；

使用变量通过 $ 引用；

### （4）反引号

`: 美式键盘与~同一个键

功能：反引号允许将shell命令的输出赋值给变量

### （5）输出重定向

command > outputfile: 将command命令产生的输出重定向到outputfile,如果之前文件存在，则覆盖

command >> outputfile: 将command 命令产生的输出重定向到outputfile, 如果之前文件存在，则追加

### （6）输入重定向

< : 将文件的内容重定向到命令

wc < test

wc: 提供对数据中文本的计数，默认输出3个值：文本的行数，文本的词数，文本的字节数

<<: 内联重定向 （需要深入学习）

### （7）管道

有时需要将奉送某个命令的输出作为另一个命令的输入，可以用重定向，但有些笨拙。

|：管道放在命令之间，将一个命令的重定向到另一个上；linux会同时运行两个命令，在系统内部将他们连接起来。

rpm –qa |sort |more

### （8）执行数学运算（整数）

expr 命令：

expr 允许在命令行上处理数学表达式，但是比较笨拙；

要将一个数学表达式的结果赋给一个变量，你需要使用反引号来获取expr命令的输出；

var=`expr var1 / var2`

使用方括号[]：

将一个数学运算的结构赋值给某个变量： var=$[ 1 + 5], var2=[$var1 \* 2]

注意：[]只支持整数运算（9）shell浮点解决方案

内建的bash计算器(bc)：

bash计算器是允许你在命令行输入浮点表达式、解释表达式、计算并返回结果的一种编程语言，能够识别 数字、变量、注释、表达式、编程语句、函数

在脚本中使用bc：

格式： variable=` echo “options; expression” | bc`

eg： var1=`echo “scale=4; 3.44/5” | bc` scale是一个内建变量，表示浮点运算保留的精度；

bc命令能够识别输入重定向，允许直接在控制台重定向数据。

### （10）退出脚本

退出状态码: $? (0-255)

查看退出状态码: echo $?

默认情况下：shell脚本会以脚本中的最后一个命令的退出状态码退出

exit命令： exit命令允许在脚本结束时指定一个退出状态码，指定范围不可超过255， 如果超过则对256取余作为退出状态码

## 使用结构化命令

### 使用 if-then语句

基本语句格式：

if command

then

commands

fi

说明： 如果if 行定义的那个命令退出状态码为0（执行成功），则位于then部分的命令会被执行；在then部分，可以使用多个命令，bash会将这部分命令当成一个块；

如此格式也是可以的： if command; then

commands

fi

### if -then –else语句

if command

then

commands

else

commands

fi

当if语句中的命令返回退出状态码为0时，then部分会被执行，否则执行else部分。

### 嵌套if

格式：

if command1

then

command set 1

elif command2

command set 2

elif command3

command set 3

fi

bash会依次执行if语句，只有第一个返回退出状态码为0的语句中的then会被执行。

### test命令

格式： if test condition

then commands

fi

或者

if [ condition ]

then commands

fi

test命令可以判断三类条件：

数值比较:

n1 -eq n2: n1是否与n2相等

n1 -ge n2: n1是否大于等于2

-gt: 大于

-le： 小于等于

-lt: 小于

-ne： 不等于

说明： test命令无法处理val变量中存储的浮点值

字符串比较：

=： 比较串是否相同

！=: 比较串是否不相同

< : 是否小于

>: 是否大于

说明：使用 > 和 < 比较需要使用转义字符，用来和重定向符号区分；大于小于顺序与sort命令所采用的不同，采用按照ascii码值的大小来比较

-n str: 长度是否非0

-z str: 长度是否为0

文件比较：

-d file: 文件是否存在且是一个目录

-e ：文件是否存在

-f :

-r：可读

-s:

-w：可写

-x: 可执行

-O: 属主

-G:

-nt

-ot

### 复合条件测试

if-then允许使用布尔逻辑来组合测试

[condition1] && [condition2] : 逻辑与

[condtiion1] || [condition2]：逻辑或

### if-then的高级特性

用于数学表达式的双圆括号（（））:

用于高级字符串处理功能的双方括号 [[]]

使用双圆括号（（））：

圆括号允许将高级数学表达式放入比较中（test只允许在比较中进行简单的算术操作）；

格式： （（expression））

expression可以是任意的数学赋值或比较表达式以及如下操作：

val++： 后增

val--:

++val: 先增

--val:

! ： 逻辑反

~： 位求反

\*\*： 幂运算

<< : 左位移

>>: 右位移

&： 位与

|： 位或

&&：逻辑与

||： 逻辑或

使用双[[]]

格式： [[exprssion]]

expression使用了test命令中采用的标准字符串进行比较；同时提供了另外一个特性（模式匹配（正则表达式））

### case命令

case命令旨在替代多个elif语句

格式：

case variable in

pattern1 | pattern2) commands;;

pattern3) commands;;

\*) default commands;;

esac

## 3.更多的结构化命令

### for命令

命令格式：

for var in list

do

commands

done

说明：在do和done之间可以包含一条或者多条标准的bash shell命令

读取列表中的简单值：

读取列表中的复杂值：

如果列表中包含单引号等字符时，需要使用转义字符；或者在双引号来定义用到单引号的值。

for循环假定每个值都是用空格分割的，如果在单独的数据值中包含空格，则必须用双引号将这些值圈起来。

从变量读取列表

从命令读取值：使用反引号执行任何能产生输的命令， 然后在for中使用该命令的输出

更改字段分割符：特殊的环境变量IFS. 默认情况下， bash shell会将空格，制表符，换行符当作字段分割符。

使用通配符读取目录：用for遍历满是文件的目录

### （2） C语言风格的for命令

命令格式：

for((var assign; condition; iteration process))

使用C语言风格的变量引用方式：

给变量赋值可以有空格；

条件变量中的变量不以$开头；

迭代过程中不使用expr命令格式。

使用多个变量

C风格的for允许迭代使用多个变量，但有多个变量时，只能在循环中定义一种条件

### （3）while命令

命令格式：

while test command

do

other commands

done

说明：while中的test和if -then中的相同；while命令的关键是，指定的test命令的退出状态码必须随着循环中运行命令的改变，否则会死循环。

使用多个测试命令：while语句定义多个测试命令，只有最后一个测试命令的退出状态码会被用来决定时候结束循环；每个测试命令都在单独的一行上的。

### (4) until 命令

命令格式：

until test commands

do

other commands

done

说明：

util直到测试命令的结果为0时结束循环；同while一样，支持多个测试命令，只有最后一个起作用。

### （5）嵌套循环

嵌套循环时，命令运行的次数是乘积关系

### （6）循环处理文件数据

### （7）控制循环

break命令：

跳出单个循环：

根据条件判断，尝试跳出正在处理的循环

跳出内部循环：

处理多个循环时，break命令会自动终止你所在最里面的循环

跳出外部循环：

break n： n为1，跳出当前循环；n为2，停止上一级循环

continue命令：

提早结束执行循环内部的命令但不完全终止整个循环的途径。

在while和until循环中使用continue命令，但要小心；

continue n: 与break n效果相同；

shell在执行continue之后，会跳过剩余的命令，如果在continue之后增加测试条件变量，就会出问题。

## 4.处理用户输入

### （1） 命令行参数

读取参数：

bash shell会将一些位置参数的特殊变量分配给命令行输入的所有参数。$0: 程序名， $1: 第一个参数， 一直到$9；

如果输入多个参数，中间需要以空格；

可以在命令行中使用文本字符串，如果要输入带有空格的文本字符串，则需要使用单引号和双引号；

如果需要多于9个命令行参数，仍然可以处理，但需要修改一下变量名。在第9个变量之后，使用时必须在变量数字周围加括号，如${10};

读取程序名：可以用$0参数来获取shell命令行启动的程序的名字，但是$0还会包含文件的路径，如果不需要路径的时候，使用basename命令，如`basename $0`

### (2) 特殊参数变量

在bash shell中有些特殊变量，会记录命令行参数。

**参数计数：**

$#特殊变量包含脚本运行时就有的命令行参数的个数

${!# }则代表最后一个命令行参数, 或者param=$#来使用最后一个命令行参数，但注意命令行上没有任何参数时，$#的值为0， ${!#}会返回命令行用到的脚本名。

**抓取所有的数据：**

$\*和$@提供了对所有参数的快速访问，都能在单个变量中存储所有的命令行参数。

$\*变量会将命令行上提供的所有参数当作单个单词保存。$\*会将这些当作一个参数，而不是多个对象。

$@会将命令行上提供的所有参数当作同一字符串中多个独立的单词。它允许遍历所有的值，将提供的每个参数分割开来，通常用for实现。

### (3)移动变量：

shift命令会根据它们的相对位置来移动命令行参数。使用shift命令时，默认将每个参数对应的变量索引位置减一，也就是左移。shift后也可以加数字，表示移动几个变量的位置。shift n.

### (4)处理选项

**查找选项**

选项是跟在单破折线后面的那个字母，能改变命令的行为；

处理简单选项，可以使用case;

分离参数和选项， linux中处理这个问题的标准方式是用特殊字符来将二者分开，这个特殊字符是--；

处理带值的选项，有些选项会带上一个额外的参数值，当命令行选项要求额外的参数时，脚本必须能检测并能正确的处理， 将多个选项放入一个参数中时就不能工作了。

**getopt命令**

getopt命令可以接受一系列任意形式的命令行选项和参数，并自动将他们转换成适当的格式，命令格式如下：

getopt optstring options parameters

例： getopt ab:cd -a -b test -cd a b， 则命令行输出为-a -b test -c -d – a b;

如果指定一个不在optionstring中的选项，getopt会输出一条错误信息，可以在getopt命令后加-q选项忽略报错信息。

在脚本中使用getopt，方法是用getopt命令生成的格式化后的版本来替换已有的命令行选项和参数。用set命令能够做到。但getopt用空格来分隔变量而不是用“”来分别变量。

与getopt将命令行上找到的选项和参数处理后只生成一个输出不同，getopts命令能够和已有的shell参数变量对应的顺序工作。每次调用它时，它只处理一个命令行上检测到的参数，处理完所有的参数后，它会退出并返回一个大于零的退出状态码，命令格式为：

getopts optstring variable，有效的选项在optstring中，如果需要加参数，则加一个冒号，如果要忽略错误信息，在optstring前加一个冒号，getoptstrings将当前参数保存在 命令行中定义的variable中。

getopts会用到两个环境变量：如果选项需要跟一个参数值， OPTARG环境变量就会保存这个值；OPTFIND变量保存了参数列表中getopts正在处理的参数位置。

getopts命令解析命令行参数时，会移除开头的单破折线。

getopts可以用双引号区分参数；

getopts可以将选项字母和参数放在一起使用，不用加空格；

getopts可以将命令行上找到的所有未定义的选项统一输出成问号；

getopts在处理每一个选项时，它会将OPTFIND环境变量值加1，在getopts完成处理时，可以将OPTIND和shift命令一起使用来移动参数。

### （5）将选项标准化

linux中用到的一些命令行选项的通用含义

-a: 显示所有对象

-c: 生成一个计数

-d: 指定一个目录

-e： 扩展一个对象

-f: 指定读入数据的文件

-h: 显示命令的帮助信息

-i: 忽略文本大小

-l: 产生输出的长格式版本

-n：使用非交互模式

-o: 指定将所有输出重定向到的输出文件

-q: 以安静模式运行

-r: 递归地处理目录和文件

-s： 以安静模式运行

-v: 生成详细输出

-x: 排除某个对象

-y: 对所有问题回答yes

### （6）获得用户输入

**基本的读取**

read命令接受从标准输入（键盘）或另一个文件描述符的输入。在收到输入后，read命令会将数据放进一个标准变量；

echo -n 去掉换行；

read -p允许直接在read命令行指定提示符；

read命令会为提示符输入的所有数据分配一个变量，也可以指定多个变量，输入的每个数据值都会分配给表中的下一个变量。如果变量表在数据之前用完了，剩下的数据就会分配给最后一个变量；

可以在read命令行中不指定变量，这样read命令将会它收到的任何数据都放进特殊的环境变量REPLY中。

**超时**

使用read命令时有个危险，就是脚本很有可能会等脚本用户的输入一直等下去，为避免这种情况，可以使用-t选项来指定一个定时器；同时read命令也可对输入的字符计数，当输入的字符达到的字符数时，它会自动退出，将输入的数据赋给变量。

**隐藏方式读取**

read命令的-s选项会阻止将传给read命令的数据显示在显示器上（数据会显示，只是read命令将文本颜色设置成和背景颜色一样）

**从文件中读取**

read命令可以读取linux系统上文件里保存的数据，每次调用read命令会从文件中读取一行文本，当文件中再没有内容时，read命令会退出并返回非零状态退出码；比较难的是将文件中的数据传给read命令，最常见的办法是将文件运行cat命令后的输出通过管道直接传给含有read的while命令。

## 5．呈现数据

### (1) 理解输入和输出

脚本显示输出的两种方式：在显示器上输出屏幕显示，将输出重定向到文件

标准文件描述符：

linux将每个对象当作文件来处理，用文件描述符来标识每个文件对象，文件描述符是一个非负整数，可以唯一的标识会话中打开的文件，每个过程最多可以有9个文件描述符。bash保留了最早的3个文件描述符。

0： STDIN 标准输入： 对于终端界面来说，标准输入是键盘，shell从STDINT文件描述符对应的键盘获得输入，在用户输入时处理每个字符；使用重定向符号“<”时，linux会利用重定向指定的文件来替换标准输入的文件描述符；

1: STDOUT 标准输出：对于终端界面来说，标准输出就是终端的显示器，shell的所有输出（包括程序和脚本）会定向到标准输出中；通过“>”, 可以将内容重定向到指定的重定向文件中；当执行命令时出现错误时，shell不会将错误信息重定向到重定向指向的文件中

2: STDERR 标准错误，shell通过STDERR来处理错误消息，shell或shell中运行的程序和脚本出错时生成的错误信息都会发送到STDERR；默认情况下，STDOUT 和STDERR文件描述符指向同样的地方，默认错误消息也会输出到显示器上；重定向STDOUT并不会重定向STDERR。

重定向错误

只重定向错误：将STDERR对应的文件描述符值放在重定向符号前，否则不会工作，eg： ls -al abce 2> test; 在同一个输出中混用STDOUT和STDERR，eg: ls -al test badtest test2 2> test5

重定向错误和数据：如果想重定向错误和输出，必须用两个重定向符号，需要在想要重定向的每个数据前添加对应的文件描述符，并将它们指向对应的保存数据的输出文件，eg: ls -al test1 test2 badtest 2> file\_6 1> file\_7；可以将STDERR和STDOUT的输出重定向到同一个输出文件，bash提供了特殊的重定向符号**&>,** 使用该方式时，命令生成的所有输出都会发送到同一位置，bash shell会自动给错误消息分配较标准输出更高的优先级，这样方便在一处地方查看错误信息。

### (2) 在脚本中重定向输出

**临时重定向**：在脚本前中生成错误信息，需要使用输出重定向符来将输出重定向到STDERR文件描述符，在重定向到文件描述符时，必须在文件描述符数字之前加一个&：

echo “This is an error message ” >&2， 如果别人使用该脚本的时候，可以通过STDERR描述符重定向错误消息，将正常和错误消息区分开。

**永久重定向**：如果脚本中有大量数据需要重定向，可以用exec命令告诉shell在脚本执行期间重定向某个特定文件描述符。但是需要注意，一旦重定向了STDOUT或STDERR，你就无法轻易将它们重定向回原来的位置。

### （3）在脚本重定向输入

可以用在脚本中重定向STDOUT和STDERR同样的方法来将STDIN从键盘重定向到其他位置，格式如： exec 0< testfile；将STDIN重定向到文件后，当read命令试图从STDIN读取数据时，它会到文件去取数据，而不是键盘。

### （4）创建自己的重定向

shell中最多可以有9个打开的文件描述符， 其他的6个文件描述符会从3排到8，并且当作输入或输出重定向均可。可以将这些文件描述符中的任意一个分配给文件，然后在脚本中使用。

**创建输出文件描述符**：

可以使用exec命令来给输出分配文件描述符，一旦你给一个文件位置分配了另外一个文件描述符，那个重定向就会一直有效，直到重新分配；如果将输出追加到现有文件中，使用exec 3>>test13out

**重定向文件描述符**

如何从已重定向的文件描述符中恢复？

eg: exec 3>&1

exec 1>&3

**创建输入文件描述符**

可以用和重定向输出文件描述符同样的办法来重定向输入文件描述符，在重定向到文件之前，先将STDIN文件描述符保存到另外一个文件描述符，然后再读取完文件之后在将STDIN恢复到它原来的位置。

**创建读写文件描述符**

可以用同一个文件描述符来从文件中读取数据，并将数据写到同一个文件中。，在向同一个文件进行读取和写入操作，shell会维护一个内部指针，知名现在在文件中的什么位置，任何读或写都会从文件指针上次保存的位置开始。

**关闭文件描述符**

如果创建了新的输入或输出的文件描述符，shell会在脚本退出时自动关闭它们，不过有些情况仍然需要手动关闭文件描述符。关闭文件描述符命令格式：

exec 3>&- : 关闭文件描述符3。文件描述符关闭之后，会阻止脚本中再使用它就不能在脚本中向它写入任何数据了；关闭文件描述符以后，如果后面在脚本中打开了同一个输出文件，shell会用一个新文件来替换已有文件。

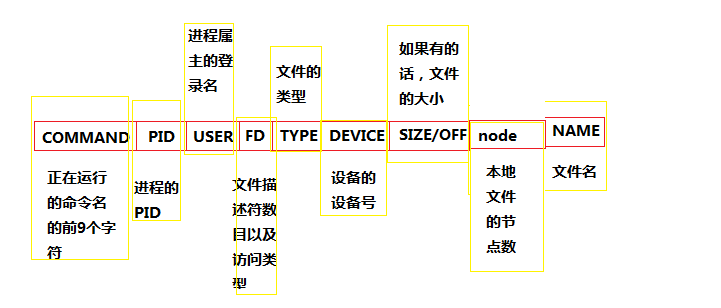
### (5)列出打开的文件描述符

有时，要记住哪个文件描述符被重定向到了哪里很难，为了便于理清楚，bash shell提供了lsof命令。 lsof会显示当前Linux系统上打开的每个文件的有关信息，包括后台运行的所有进程和登录到系统的任何用户；

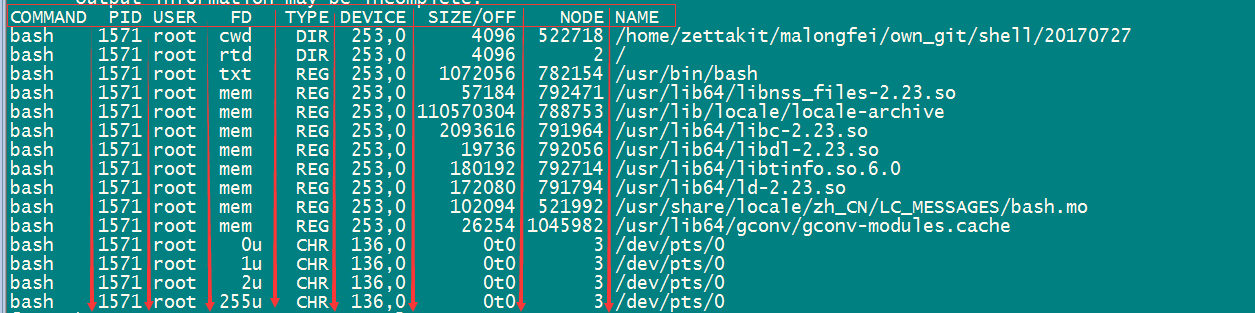
lsof -p PID: 指定进程， 进程当前的PID：$$

lsof -d 0.1.2: 指定要显示文件描述符个数

lsof内容输出格式如下：



例：



### （6）阻止命令输出

如果不想显示脚本的输出，可以将内容重定向到一个称作null文件的特殊文件，shell输出到null文件的任何数据都不会保存；linux上null文件的标准位置是/dev/null；

阻止任何错误消息而不保存它们的一个通用方法：

ls -al badfile test16 2> /dev/null；

/dev/null文件不含任何内容，通常用它来快速移除现有文件中的数据而不用先删除文件再创建，清除后该文件内容清空，但是文件依旧存在。

### （7）创建临时文件

linux系统有一个特殊的留给临时文件用的目录位置，使用/tmp目录来存放不需要一直保留的文件，大多数发行版的linux系统在启动时会自动删除/tmp目录的所有文件；

系统上的任何用户都有权限在/tmp下创建临时文件；

mktemp：在/tmp目录下创建一个唯一的临时文件。

**创建本地临时文件**

用mktemp命令在本地目录中创建一个临时文件，只要指定一个文件名模板就行了，模板可以包含任意文本文件名，在文件名末尾加6个X就行了，从而保证文件名在目录中是唯一的。

**在/tmp目录创建临时文件**

-t选项会强制mktemp命令在系统的临时目录创建文件，使用这个特性时，mktemp会返回用来创建临时文件的全路径，而不只是文件名，于是便可以在任何目录下引用该临时文件。

**创建临时目录**

-d 选项告诉mktemp命令来创建一个临时目录。

### (8) 记录消息

有时候需要将输出一边发送到显示器，一边发送到日志文件，这时候不用将输出重定向，只要用特殊的命令tee命令就行；tee命令相当于管道的一个T型接头，它将STDINT过来的数据同时发送给两个目的地，一个STDOUT，另一个为tee命令所指定的文件名。

可以用tee命令和管道命令来一起将任何命令的输出重定向：

如： date | tee testfile 该命令会将date的输出到屏幕上，同时也重定向到文件中。

注意：tee命令会在每次使用时覆盖输出文件内容， 如果要追加，必须加-a选项。

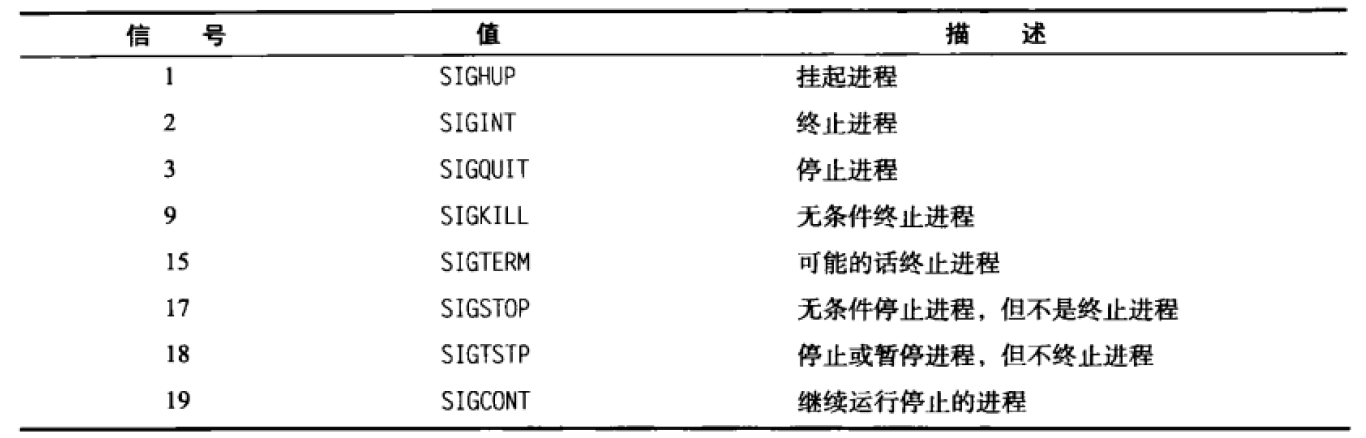
## 6 控制脚本

### 处理信号

可以用linux的信号来控制shell脚本的运行，通过对脚本进行编程使其在收到linux系统的特定信号时执行特定的命令。

**重温linux信号**：

常见的linux信号



默认情况下， bash shell会忽略收到的SIGQUIT（3）信号和SIGTERM（5）信号，但是会处理SIGHUP（1）和SIGINT（2）信号，收到进程挂起信号会，bash shell会退出，在退出之前，会将该信号传给shell启动的所有进程；SIGINT信号可以中断shell,linux内核会停止将CPU的处理时间分配给shell，shell同样会将SIGINT信号传给shell启动的所有进程。

shell会将这些信号传给shell脚本程序来处理。而shell脚本的默认行为是忽略这些信号，要避免这种情况，可以在脚本中加入识别信号和处理信号结果的命令。

**产生信号**

终止进程： CTRL + C，生成SIGINT信号

暂停进程：CTRL +　Z, 生成SIGTSTP信号，这会停止进程，但是会让程序继续保留在内存中，并能从上次停止的位置继续运行； 被停止的进程的状态为T。

**捕捉信号**

trap命令允许你来指定shell脚本观察那些LINUX信号并从shell中拦截，如果脚本收到了trap命令中列出的信号，它会阻止它被shell处理，而在本地处理。

trap的命令格式是：

trap commands signals

**捕捉脚本的退出**

要捕捉脚本的退出，在trap命令后加EXIT信号就行

**移除捕捉**

可以用单破折线作为命令，后跟要移除的信号来移除一组信号捕捉，将其恢复到正常状态。

### (2) 以后台模式运行脚本

一些脚本在命令行界面直接运行不太方便，此时需要后台运行，后台运行的进程不会和终端会话上的STDIN，STDOUT，STDERR关联。

**后台运行脚本**

后台运行脚本，只需在命令后加个“&”符号即可。

eg: test1.sh &

当&符号放到命令后，它会将命令和bash shell分离开来，并将它作为系统上的独立后台进程，第一行显示，作业号和进程ID；当后台进程仍在运行时，仍然会使用终端显示器来显示STDOUT和STDERR消息。

**运行多个后台作业**

可以在命令行提示符下同时启动任意多个后台作业

**退出终端**

在终端会话使用后台进程时，如果进程会话退出了，后台进程也会退出。

### （3）在非控制台下运行脚本

如果要让脚本一直后台 运行，即使退出了终端会话也不终端，可以使用**nohup**命令，该命令在关闭会话时会忽略任何终端发来的SIGHUP信号；同样，该命令会丢失到STDOUT与STDERR的链接，为了保存该命令产生的输出，nohup命令会自动将STDERR和STDOUT的消息重定向到一个名为nohup.out的文件；nohup.out文件包含了所有命令的输出，只追加，不清空，多个命令使用nohup时要注意。

### （4）作业控制

**查看作业**

查看作业的命令： jobs：查看shell当前正在处理的作业

jobs命令的参数：

-l：列出进程的PID及作业号

-n：只列出上次shell发出的通知后改变了状态的作业

-p：只列出作业的PID

-r：只列出运行中的作业

-s：只列出已停止的作业

jobs命令输出的说明：

带“+”号的作业被当作默认的作业，带“-”号的作业会在的的当前默认作业完成处理时成为下一个默认作业，任何时候只有一个带加号的作业和一个带减号的作业。

**重启停止的作业**

以后台方式重启一个作业，可以用bg命令加上作业号：

eg: bg 2;

以前台模式重启作业，可用fg命令，加上作业号：

eg: fg 2

### (5) 调整谦让度

在多任务操作中，内核负责将每个CPU时间分配给系统上运行每个进程；默认情况下， 从shell启动的所有进程在linux系统上都有同样的调度优先级，以优先级0来启动（-20~20）。如果想改变特定命令的优先级，可以通过nic命令实现。

**nice命令at**

例子：nice -n 10(调度优先级)；

nice阻止普通用户来增加命令的优先级

**renice命令：**

想改变系统上已运行命令的优先级，使用renice命令，该命令允许你指定进程的PID来改变它的优先级。

只能对于你的进程执行renice;

只能通过renice降低进程的优先级；

root用户可以通过renice来调整任何进程的优先级到任何级别。

**（6）定时运行作业**

**用at命令来计划执行作业**

at命令允许指定linux系统何时运行脚本，at命令会将作业提交到队列中，指定shell何时运行该作业；at的守护进程atd会以后台模式运行，并检查作业队列来运行作业。大多数linux发行版会在启动时运行此守护进程。

atd守护进程会检查系统上的一个特殊目录（通常位于/var/spool/at）来获取at命令提交的作业。默认情况下，atd守护进程会妹60s检查一下这个目录，有作业时，atd会检查作业设置运行的时间。如果时间和当前时间匹配，atd守护进程会运行此作业。

***at命令的格式***：

at [-f filename] time

time参数指定了Linux系统何时运行该作业：

标准的小时和分钟格式： 如 10:15

~/AM/~PM指示符： 10:15~PM

特定可命名的时间：now、noon、midnight或teatime

标准日期格式: MMDDYY MM/DD/YY, DD.MM.YY

文本日期：Jul 4

也可以指定时间增量：

+25min： 基于当前时间

如果指定了一个已经过去的时间，at命令会在第2天的时间运行该作业。

使用at命令时，作业会被提交到作业队列。作业队列会保存通过at命令提交的待处理的作业，针对不同优先级，存在26中不同的作业队列。作业队列通常用小写字母a~z来引用。作业队列的字母排序越高，优先级越低。如果想以更高优先级运行作业，使用-q参数指定不同的队列字母。

***获取作业的输出***

当作业在linux系统上运行时，没有屏幕会关联到该作业。任何发到STDOUT和STDERR的输出都会以邮件系统发送给提交该作业的用户的E-mail地址；

at命令会生成一条警告信息，说明系统用哪个shell来运行脚本/bin/bash，以及关联到该作业的作业号和该作业计划运行的时间；

当作业完成时，屏幕不会有任何输出，但系统生成了一封邮件信息，该邮件信息会显示脚本产生的输出，如果脚本没有产生任何输出，默认情况下不会生成邮件消息，如果要在脚本没有输出的情况下也生成邮件，使用-m选项。

***列出等待的作业***

使用atq命令查看系统中哪些作业在等待

***删除作业***

使用atrm来删除等待中的作业：

atrm 作业号。

**计划定期执行脚本**

linux系统使用cron程序来计划要定期执行的作业。cron程序会在后台运行检查特殊的称作cron时间表的表，来获得计划执行的作业。

***cron时间表***

cron的时间表的格式：

**min hour dayofmonth month dayofweek command**

可以使用特定值，可以使用通配符

例：

15 10 \* \* \* command: 每天10点15运行命令

15 16 \* \* 1 command: 每周一4点15运行命令

00 12 \* \* \* if [`date +%d -d tomorow`] : then: command: 每个月的最后一天执行命令

命令列表必须指定为全路径名，你可以添加任何想要的命令行参数和重定向符号；cron程序会用提交作业的用户账户运行该脚本，因此必须有访问该命令和命令中指定的输出文件的权限。

***构建cron时间表***

列出当已有的时间表： crontab -l

为cron时间表添加目录： crontab -e

***cron目录***

当你创建的脚本不要求有精确的执行时间时，用预配置的cron脚本目录会更方便。

四个基本目录：

hourly:

daily:

monthly:

weekly:

如，你有脚本需要每天运行一次，将其复制到daily目录即可，cron会每天执行它。

***anacron程序***

cron程序的前提是假定linux系统是24小时运行的，如果系统被关闭，则不会被运行了。

anacron程序会使关机时错过执行的计划在再次开机时自动运行。

anacron程序只会处理位于cron目录的程序，如/etc/cron.monthly。它用时间戳来决定作业是否在适当的计划间隔内运行。每个cron目录都有个时间戳文件，位于/var/spool/anacron；anacron程序有自己的用来检查作业目录的表，通常位于/etc/anacrontab.

anacron时间表的基本格式：

period delay identifier command

period: 定义作业多久运行一次，以天为单位

delay: 指定系统启动后多少分钟anacron程序开始执行错过的脚本

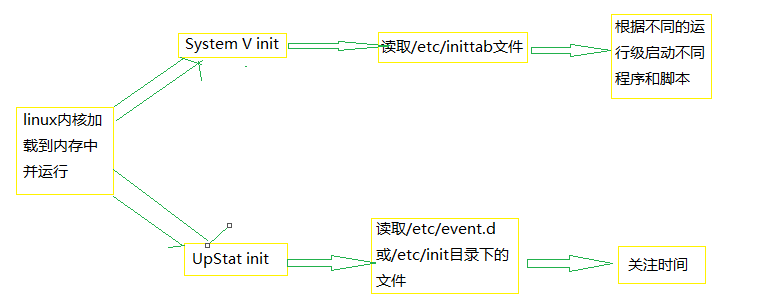
identifier: 是一种特别的非空白字符字符串，如cron-weekly。它用于唯一识别日志消息和错误E-mail中的作业。

command: 包含了run-parts程序和一个cron脚本目录名。run-parts程序负责运行目录中传给它的任何脚本。

### （7）启动时运行

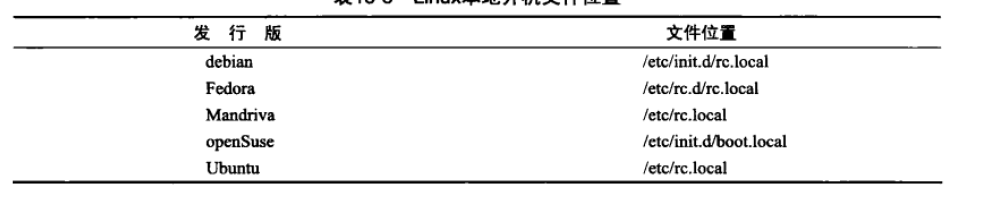
**开机时运行脚本**

***开机过程：***



定义自己的开机脚本：

大多数Linux发行版提供了一个本地开机文件专门让系统管理员添加开机时运行的脚本。



在本地开机文件中，可以指定特定的命令和语句，或者输入任何想在开机是执行的脚本，如果要用脚本，必须指定脚本的全路径名，这样系统在开机时才能找到。

**在新shell中启动**

每个用户的主目录下有两个文件，bash shell会用他们来启动脚本并设置环境变量：

.bash\_profile文件：

.bashrc文件：

当shell是新的登录生成的话，bash shell会运行.bash\_profile文件；

当新shell启动时，包括新的登录的情况，bash shell会运行.bashrc文件。

# 高级shell脚本编程

## 创建函数

## 基本的脚本函数

**创建函数：**

两种格式：

（1）function name { //唯一函数名

commands // bash shell命令

}

（2） name() {

commands

}

**使用函数**

函数定义不必一定在shell脚本最前面，但是如果在函数定义前使用函数，会收到错误信息，提示该函数找不到；函数名必须是唯一的，如果函数名重复，也不会产生错误消息，这种情况要格外小心。

### （2） 返回值

**默认退出状态码**

默认情况下，函数的退出状态码是函数中最后一条命令返回的退出状态码，在函数执行结束后，可以使用标准的$?变量来决定函数的退出状态码

**使用return命令**

使用return命令退出函数并返回特定的退出状态码，return命令指定允许一个整数值来定义函数的退出码状态。

但需注意：

函数一结束就取返回值；

退出状态码必须在0-255之间；

如果在用 $?变量之前执行了其他命令，函数的返回值可能会丢失，因为$?变量返回的是执行的最后一条命令的退出状态码。

如果要返回大于256的数值，同样也不可以使用此种方式。

**使用函数输出**

可以将函数的输出保存到shell变量中，利用这种方式可以获得任何类型的函数输出，并将其保存到变量中。这种方式可以返回浮点值和字符串值。

由于函数使用特殊参数变量作为自己的参数值，它不能直接从脚本的命令行获取脚本参数值，不在调用时手动将它们传过去；

### （3）在函数中使用变量

**向函数传递参数**

函数可以使用标准的参数环境变量来代表命令行上传给函数的参数，如$0, $1, $2，也可以用特殊变量$#来判断传给函数的参数数目；

在脚本中指定函数时，必须将参数和函数放在同一行，如：

func1 $value 10

**在函数中处理变量**

**全局变量**

全局变量是在shell脚本中任何地方都有效的变量；默认情况下，脚本中定义的任何变量都是全局变量。在函数外定义的变量可在函数内正常访问。

如果变量在函数内被赋予了新值，在脚本中引用该变量时，新值依然有效，这样比较危险。

**局部变量**

函数内部使用的任何变量都可以被声明成局部变量，那么做时，只要在变量声明的前面加上local关键字就可以，如：

local temp

也可以在变量赋值时在复制语句中使用local关键字

local temp=$[ $value + 5]

local关键字保证了该变量只局限在函数中，如果在脚本中该函数之外有同样的名字的变量，shell会将保持这两个变量的值时分离的。

### （4） 数组变量和函数

**向函数传数组参数**

将数组变量当作单个参数传递的话，它不会起作用，如果你试图将该数组变量当成一个函数参数，函数只会取数组变量的第一个值；要解决这个问题，必须将该数组变量的值分解成单个值然后将这些值作为函数参数使用；在函数内部，可以将所有的参数重组到新的数组变量中。

在函数内部，数组仍可以像其他数组一样使用。

**从函数返回数组**

从函数里向shell脚本传回数组变量，用echo语句按正确顺序输出单个数组值，然后再将它们重新放进一个新的数组变量中。

### （5）函数递归

局部函数变量提供的一个特性就是自成体系，自成体系的函数不需要使用任何外部资源，除了脚本在命令上传给它的变量。

这个特性使得函数可以递归地调用。

### （6）创建库

bash shell允许创建函数库文件，需要时在多个脚本中引用该文件。

第一步： 创建一个包含脚本中所需函数的公用库文件：

第二步： 在要用函数的脚本文件中包含myfuncs库文件；

和环境变量一样，shell函数只对定义它的shell会话有效。

使用函数库的关键在于source命令，source命令会在当前的shell上下文中执行命令，而不是创建一个新的shell来执行命令。可以用source命令来在shell脚本中运行库文件脚本。这样函数就对脚本可用了。

source命令有个快捷别名，称作点操作符。在shel脚本中运行myfuncs库文件，秩序添加下面一行：

. ./myfuncs 或 source ./myfuncs

### (7) 在命令行上使用函数

和在shell脚本中脚本函数当命令使用一样，也可以在命令行界面上脚本函数当命令用。在shell中定义了函数，就可以在整个系统上直接用了，而无需担心脚本是不是在PATH环境变量里。

**在命令行上创建函数**

**在一行内定义整个函数：**

必须记住在每个命令后面加个分号，这样shell就能知道在哪里分开命令了：

如： function divem { echo $[ $1 / $2];}

**用多行来定义函数：**

在定义时， bash shell会使用次提示符来提示输入更多命令，这种方法，不用在每条命令的末尾放一个分号，只要按回车键就行， 在函数的尾部使用花括号，shell就会知道你已经完成了函数的定义。

注意：在命令行上创建函数时如果起了跟内建命令或另一个命令相同的名字，会覆盖原来的命令。

**在.bashrc文件中定义函数**

在命令行上直接定义shell函数的明显缺点是当退出shell时，函数就消失了。一个简答的方法是在一个每次启动新shell时都会加载的地方定义函数。最好的地方是.bashrc文件。bash shell会在每次启动是在主目录查找整个文件，不管是交互式的还是从现有shell中启动一个新的shell。

**直接定义函数**

在主目录的 .bashrc文件中直接定义函数。

**读取函数文件**

可以在主目录的.bashrc文件中，使用source命令或者点操作符将已有库文件中的函数添加到.bashrc脚本中，但需确保引用库文件的路径名正确。

shell还会将定义好的函数传给子shell进程，这样这些函数在该shell会话中的任何shell脚本中也都可用。

## 图形化桌面上的脚本编程

### (1) 创建文本菜单

shell脚本菜单的核心是case命令，case命令会根据用户在菜单上的选择来执行特定的命令。

**创建菜单布局**

clear命令用于清除当前终端会话的terminfo数据；  
 用echo命令来显示菜单元素；

默认情况下，echo命令只可显示可打印的文本字符，在创建菜单项时，可以打印非打印字符，如制表符，但需用-e选项；

**创建菜单函数**

可以将用echo创建的菜单显示封装成为一个shell函数。

**添加菜单逻辑**

使用case命令，增加不同的处理逻辑。

**整合shell脚本菜单**

对于每一个case增加具体的处理。

**使用select命令**

select命令允许从单个命令行创建菜单，然后再提取输入的答案并自动处理。select命令的格式如下：

select variable in list

do

commands

done

list参数是构成彩菜单的空格分割的文本选项列表，select命令会在列表中将每个选项作为一个编好号的显示，然后为显示选项显示一个特殊的由PS3环境变量定义的提示符。5

### 使用窗口

**dialog包**

**msgbox部件**

**yesno部件**

**inputbox部件**

**textbox部件**

**menu部件**

**fselect部件**

**dialog选项**

**此处由于未有dialog相关实战环境，暂了解**

**在脚本中使用dialog命令**

在脚本中使用dialog命令非常迅捷，但必须记住两件事**：**

i: 如果有Cancel或No按钮，检查dialog命令的退出状态码；

ii: 重定向STDERR来获得输出值

### （3）使用图形

**KDE环境**

KDE图形化默认包含kdialog包，kdialog包使用kdialog命令在KDE桌面上生成类似于dialog式部件的标准窗口，这些窗口能和其他KDE应用窗口很好地融合，不会产生笨重的感觉。

**kdialog部件**

和dialog命令类似，kdialog命令使用命令行选项来指定具体使用那种类型的窗口部件，kdialog命令的格式为：

kdialog display-options window-options arguments

window-options选项指定使用那种类型的窗口部件

**使用kdilog**

在shel脚本中使用kdialog窗口部件，方法类似于如何使用dialog，最大的不同是kdialog窗口部件用STDOUT来输出值，而不是STDERR。

**GNOME环境**

GNOME图形化环境支持两个流行的可生成标准窗口的包。gdialog和zenity.

**zenity部件**

zenity允许用命令创建不同的窗口部件。zenity命令行程序与kdialog和dialog程序的工作方式有些不同，许多部件类型都用命令行上的其他选项定义，而不是将它们包括在选项的参数里。zenity命令会将值返回到STDOUT中，与kdialog一样。

显示日历：zenity –calendar

文件选择选项： zenity –file-selection

**在脚本中使用zenity**

## 初识sed和gawk

sed和gawk工具能够极大的简化需要进行的数据处理任务，这些工具能够自动地格式化、插入、修改或删除文本元素.

### 文本处理

**sed编辑器**

sed编辑器被称作流编辑器，与普通的交互式文本编辑器恰好相反。在交互式文本编辑器中，用键盘命令来交互式地插入、删除或替换数据中的文本，流编辑器则会在编辑器处理数据之前基于预先提供的一组规则来编辑数据流。

sed编辑器可以基于输入到命令行的或是存储在命令文本文件中的命令来处理数据流中的数据。每次从输入中读取一行，用提供的编辑器命令匹配数据、按命令中指定的方式修流中的数据，然后将生成的数据输出到STDOUT。在流编辑器将所有命令与一行数据进行匹配后，它会读取下一行数据并重复这个过程。在流编辑处理器处理完流的所有数据行后，它就会终止。

sed命令的格式：

sed options script file，选项参数允许修改sed命令的行为，包含 -e script, -f file, -n选项。如果需要用多个命令，必须用-e选项来在命令行上指定它们，或者用-f选项来在单独的文件中指定。

**在命令行定义编辑器命令**

默认情况下，sed编辑器会将指定的命令应用到STDIN输入流上，这样就可以直接将数据管道输出到sed编辑器上处理。

例: echo “This is a test” | sed ‘s/test/big test/’，s命令会用斜线间指定的第二个文本字符串来替换第一个文本字符串。

sed命令几乎瞬间就执行完并返回数据。可以在sed编辑器处理完整个文件之前就开始观察结果。

重要的是要记住，sed编辑器自身不会修改文本文件的数据。它只会将修改后的数据发送到STDOUT。

**在命令行上使用多个编辑器命令**

要在sed命令上执行多个命令时，只要-e选项就可以了。命令之间必须用分号分隔，并且在命令末尾和分号之间不能有空格；也可以用bash shell中的次提示符来分隔命令，而不用分号。只要输入第一个单引号来开始编写，bash会继续提示你输入更多命令，知道输入了封尾的单引号。

**从文件中读取编辑器命令**

如果有大量要处理的sed命令，将它们放进一个文件中会更方便。可以在sed 命令中用-f选项来指定文件。在这种情况下，不用在每条命令后面放一个分号。sed编辑器知道每行都有一条单独的命令，与在命令行上输入命令一样，sed编辑器会从指定文件中读取命令，并将它们应用到数据文件中的每一行中。

**gawk程序**

sed程序也有自身的限制，我们还需要一个用来处理文件中的数据的更高级工具，能提供一个类编程环境，允许修改和重新组织文件中的数据，gawk程序具有此特点。

gawk程序是Unix中的原始awk程序的GNU版本。Gawk程序提供了一种编程语言而不只是编辑器命令。在gawk编程语言中，可以做：

定义变量来保存数据；

使用算术和字符串操作符来处理数据；

使用结构化编程概念；

提取数据文件中的数据元素并将它们按另一顺序或格式重新放置，从而生成格式化报告。

**gawk命令格式**

gawk程序的基本格式为：

gawk options program file

可用选项： -F fs，-f file， -v var=value，-mf N，-mr N，-W keyboard

gawk的强大之处在于程序脚本，可以写脚本来读取问本行的数据，然后处理并显示数据，创建任何类型的输出报告。

**从命令行读取程序脚本**

跟sed编辑器一样，gawk程序会针对数据流中的每行文本执行一遍程序脚本，由于程序脚本被设为显示固定的文本字符串，所以不管你在数据流中输入什么文本，你都会得到同样的文本输出。

要终止gawk程序，你必须发出信号说明数据流易经结束了。bash shell用EOF字符来结束， CTRL+D组合键会在bash 中产生一个EOF字符。使用者对组合键就能终止gawk程序并返回到命令行界面提示符下。

**使用数据字段变量**

gawk的基本特性之一是它处理文本文件中数据的能力。它会自动给每行中的每个数据元素分配一个变量。默认情况下，gawk会将如下变量分配给它在文本行中发现的每个数据字段。

$0： 整个文本行

$1: 文本行中的第一个数据字段

$2: 文本行中的第2个数据字段

$n: 问本行中的第n个数据字段，

每个数据字段在文本行中都是通过字段分割符来划分的。gawk读取一行文本时，它会用预定义的字段分割符划分每个数据字段。gawk中默认的字段分隔符是任意的空白字符（例如空格或者制表符）。

如果要读取用其他字段分隔符的文件，可以用-F选项指定：

gawk -F: ‘{print $1}’ /etc/passwd

**在程序脚本中使用多个命令**

gawk编程语言允许你讲多条命令组合成一个正常的程序。要在命令行上的程序脚本中使用多条明king，只要在每条命令之间放个分号即可。

例： echo “My name is Rich” | gawk ‘{$4=”Christine”; print $0}’

**从文件中读取程序**

与sed编辑器一样，gawk编辑器允许将程序存储到文件中，然后再在命令行中引用。

可以在程序文件中指定多条命令。只要将每条命令放到一个新的行就好了，不需要分号；gawk程序在引用变量时未像shell脚本一样使用美元符。

**在处理数据前运行脚本**

gawk允许指定程序脚本何时运行。默认情况下，gawk会从输入中读取一行文本，然后针对文本的数据执行程序脚本。有时候可能需要在处理数据前运行脚本，比如为拔高创建开头部分。BEGIN关键字用来做此事，它会强制gawk在读取数据前执行BEGIN关键字后指定的程序脚本。

gawk ‘BEGIN {print “Hello World”}’

此例中，print命令会在读取任何数据前显示文本，在现实文本之后，它会快速退出而不用等待任何数据。原因是gawk处理任何数据前，BEGIN关键字只执行指定的脚本。要想在正常的程序脚本中处理数据，必须用另一个脚本段来定义程序：

gawk ‘BEGIN {print “The data4 FILE contents:”} {print $0}’ data4

现在在gawk执行了BEGIN脚本后，会用到第二段脚本来处理任何文件数据。这么做要小心，两段脚本在gawk命令行上会被当成一个文本字符串，需要相应的加上单引号。

在处理数据后运行脚本

与BEGIN关键字类似，END关键字允许你指定一个程序脚本，gawk会在读完数据后执行它。

### （2）sed编辑器基础

成功使用sed编辑器的关键在于掌握它各式各样的能帮你定制文本编辑行为的命令和格式。

**更多的替换选项**

sed编辑器可以用s命令来替换文本内容，还有substitute命令选项可供使用。

**替换标记**

substitute命令在替换多行中的文本时能正常工作，但默认情况下它只替换每行中出现的第一处。要让替换命令对一行中不同地方出现的文本都起作用，必须使用替换标记。替换标记会在替换命令字符串之后设置：

s/pattern/replacement/flags

有4种可用的替换标记：

数字：表明新文本将替换第几处模式匹配的地方

g: 表明新文本将会替换所有已有文本出现的地方

p：表明原来行的内容要打印出出来

w file: 将替换的结果写到文件中

sed编辑器的正常输出是在STDOUT中，而只有那些包含匹配模式的行才会保存在指定的输出文件中。

**替换字符**

有些文本字符串中的字符不方便在替换模式中使用，如果linux中的正斜线。由于正斜线通常用作字符串分隔符，因而如果它出现在了模式文本中的话，必须用反斜线来转义，这回带来一些困惑和错误。

解决这个问题，sed编辑器允许选择其他字符来作为substitute命令中的字符串分隔符：

sed ‘s!/bin/bash!/bin/csh!’ /etc/passwd

**使用地址**

**删除行**

**插入和附加文本**

**修改行**

**转换命令**

**回顾打印**

**用sed和文件一起工作**

## 4.正则表达式

## 5.sed进阶

## 6.gawk进阶

## 7.使用其他shell

# 高级shell脚本编程主题

## 1.使用数据库

## 2.使用web

## 3.使用E-mail

## 4.编写脚本是用工具

## 5.shell脚本编程进阶