Linux操作系统及应用技术

vi编辑器

Linux下文本编辑的功能越来越强大。很多公司已将它们的Office软件移植到Linux上。但是，这些软件是在X Window下使用的，而且不是所有的Linux版本都附带的。所以，首先要学习基本的**文字编辑器**。

在Linux系统中，vi是常用的编辑器，它的文本编辑功能十分强大。它是一个全屏幕文本编辑器，是visual interface的简称，几乎每个Linux系统都提供了vi。初学者可能感到使用vi很困难，但经过一段时间的学习和使用后，就会体会到使用vi非常方便。

目录

本章要点

4.1 vi的工作方式

4.5 保存文件

4.2 vi的启动和退出

4.6 移动光标

4.3 文本输入

4.7 文本修改

4.4 打开文件

4.8 ex命令

vi编辑器有3种工作方式，即**命令方式**、**输入方式**和**ex转义方式**。通过相应的命令或操作，3种工作方式之间可以**相互转换**。

4.1.1 命令方式

当输入命令vi后，进入vi编辑器时，就处于vi的命令方式。此时，从键盘上输入的任何字符都被作为**编辑命令来解释**，如a（append）表示附加命令、i（insert）表示插入命令、x表示删除字符命令等。如果输入的字符不是vi的合法命令，则计算机将发出“报警声”，光标不移动，且在命令方式下输入的字符（即vi命令）并不在屏幕上显示出来，如输入i，屏幕上并无变化，但通过执行i命令，编辑器的工作方式却发生变化，由命令方式变为输入方式。

4.1.2 输入方式

通过输入vi的插入命令（i）、附加命令（a）、打开命令（o）、替换命令（s）、修改命令（c）或取代命令（r）即可从命令方式进入到输入方式。

在输入方式下，从键盘上输入的所有字符都被插入到正在编辑的**缓冲区**中，被当作该文件的正文。进入输入方式后，输入的可见字符都在屏幕上显示出来，而编辑命令不再起作用，仅作为普通字母出现。

例如，在命令方式下输入字母i，进入到输入方式，然后再输入i，就在屏幕上相应光标处添加一个字母i。设原来屏幕显示的情况如下：

```
/* this is an example */
main()
{    prntf("ok!");
}
~
~
…
```

4.1.2 输入方式

光标停在字符串prntf的字母n处。输入i命令，屏幕显示没有变化。接着再输入i，屏幕显示为：

```
/* this  is  an  example */
main()
{      printf("ok");
}
~
~
…
```

在字母n之前加入了一个字母i。

由输入方式回到命令方式的办法是按下【Esc】键（通常在键盘的左上角）。如果已在命令方式下，那么按下Esc键就会发出“嘟嘟”声。如果不能断定目前处于什么模式，可以多按几次【Esc】键，听到系统发出蜂鸣声后，证明已经进入命令模式。

4.1.3 ex转义方式

vi和ex编辑器的功能是相同的，二者的主要区别是用户界面：在vi编辑器中，命令通常是一个字符，如a、x、r等；而在ex编辑器中，命令是以回车结束的正文行。

vi编辑器有一个专门的“转义”命令，可访问很多面向行的ex命令。为使用ex转义方式，可输入一个冒号“:”。冒号作为ex命令提示符出现在状态行（通常在屏幕下一行）。按下**“中断”键**（通常是【Del】键）可终止正在执行的命令。多数文件管理命令都是在ex转义方式下执行的（如读取文件、将编辑缓冲区的内容写到文件中等）。

4.1.3 ex转义方式

例如：

:1, $ s/I/i/g　　回车

表示从文件第一行至文件的末尾（$）将大写I全部替换成小写i。

转义命令执行后，自动回到命令方式。

vi编辑器的3种工作方式之间的转换如图4-1所示。

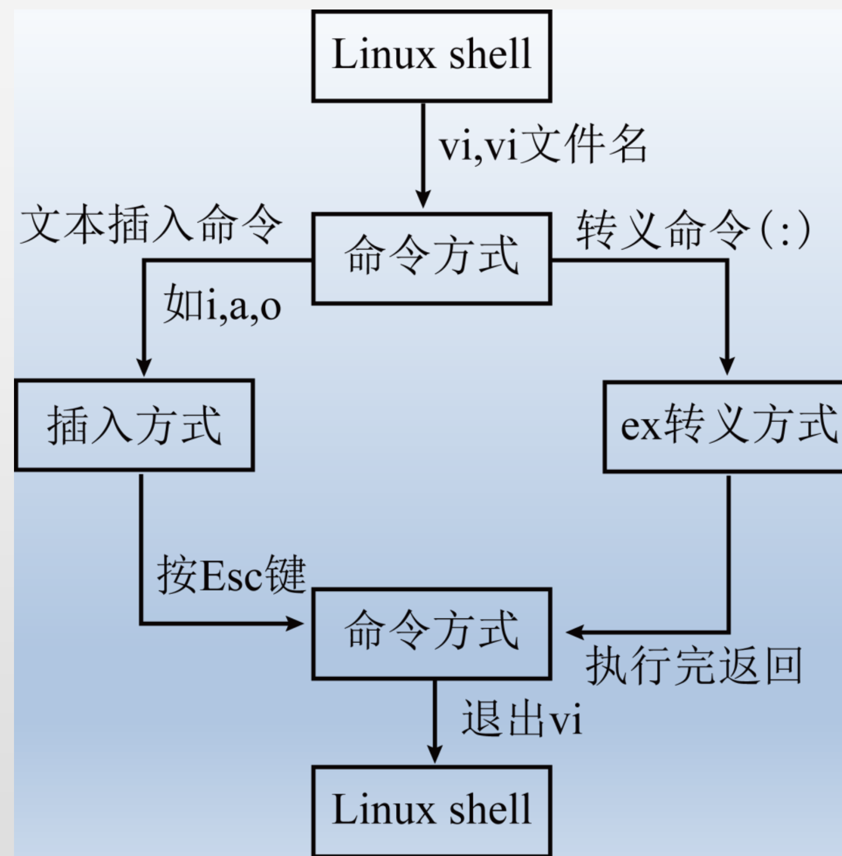


图4-1　vi编辑器3种工作方式之间的转换

目录

本章要点

4.1 vi的工作方式

4.2 vi的启动和退出

4.3 文本输入

4.4 打开文件

4.5 保存文件

4.6 移动光标

4.7 文本修改

4.8 ex命令

只有进入vi编辑器之后才可以使用vi的命令；完成文本编辑以后，应退出vi，回到**shell命令**状态下。

4.2.1 启动vi

在系统提示符下，输入命令vi和想要编辑（建立）的文件名，便可进入vi。例如：

```
[root@localhost root]#vi  example.c
~
~
~
~
……
~
~
"example.c"[未命名]                    0,0-1      全部
```

上述示例表示example.c是一个新文件，里面还没有任何东西。光标停在屏幕的左上角。在每一行开头都有一个“～”符号，表示空行。

4.2.1 启动vi

如果指定的文件已在系统中存在，输入上述形式的命令后，那么在屏幕上显示出该文件的内容，光标停在**左上角**。在屏幕的底行显示出一行信息，包括正在编辑的文件名、行数和字符个数。该行称为vi的状态行

例如：

```
[root@localhost root]#vi m1.c
main() {
  printf("Hello!\n");
}
~
……
~
"m1.c"[已转换]   4L,24C                4,1          全部
```

4.2.2 退出vi

当编辑完文件后，准备返回到shell状态时，应执行退出vi的命令。在vi的ex转义方式下有4种方法可以退出vi编辑器。

（1） **“:wq”** 的功能是将编辑缓冲区的内容写到指定的文件中，以退出编辑器，回到shell状态下。

其操作过程是，先输入**冒号“:”**，再输入命令wq，然后按【Enter】键。

（2） **“:ZZ”** 的功能是仅当对所编辑的内容做过修改时，才将缓冲区的内容写到指定文件上。

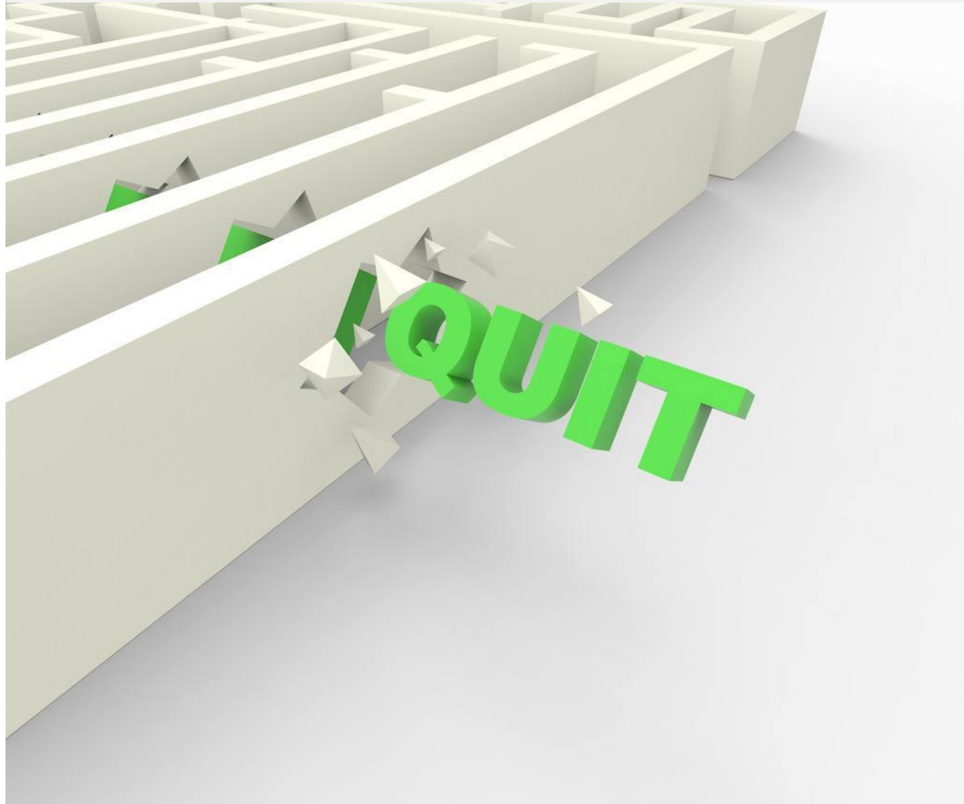
（3） **“:x”** 的功能与“:ZZ”相同。

（4） **“:q！”** 的功能是强行退出vi。感叹号“！”，通知vi，无条件退出，不将缓冲区中的内容写到文件中。

说明

以下命令的操作方式均与其相同。

4.2.2 退出vi



应该强调的是，当利用vi编辑器编辑文本时，输入或修改的内容都存放在**编辑缓冲区**中，并没有存放在磁盘的文件中。如果没有使用写盘的命令而直接退出vi，那么编辑缓冲区中的内容就被**丢弃了**，即在此之前所做的编辑工作也就白费了。所以，退出vi时，应该考虑是否需要保存所编辑的内容，再执行合适的退出命令。

目录

本章要点

4.1 vi的工作方式

4.2 vi的启动和退出

4.3 文本输入

4.4 打开文件

4.5 保存文件

4.6 移动光标

4.7 文本修改

4.8 ex命令

如果想新建一个文件或者想对已存在的文件进行添加或者要做较多修改，就要在输入方式下**输入新的文本**。文本插入命令总是进入输入方式。以下命令是纯粹的**插入命令**，使用时不会删除文本。

4.3.1 插入命令

插入命令有两个，即i和I。

（1）在i命令之后输入的内容都插在光标位置之前，光标后的文本相应向右移动。如按**【Enter】键**，就插入新的一行或者换行。

（2）输入I命令后，光标所在行的行首插入新增文本，行首是该行的第一个非空白字符。

例如，屏幕初始显示为：

```
/*this is an example */
main()
{
 a,b=10;_
printf("%d\n",a=b*2);
}
~
```

4.3.1 插入命令

输入I命令后，光标就移到行首，屏幕显示为：

```
/* this  is  an example */
main()
{
a,b=10;
printf("%d\n",a=b*2);
}
~
```

此时，光标移到该行的首字符“a”上。接着输入int和一个“空格”，显示为：

```
/*this  is  an  example */
main()
{
int a,b=10;
printf("%d\n",a=b*2);
}
~
```

4.3.2 附加命令

附加命令有两个，即a和A。

注意

A命令是将文本添加到行尾的唯一方法

（1）a命令：

该命令之后输入的字符都插入到光标之后，光标可在一行的任何位置。

（2）A命令：

在光标所在行的行尾添加文本。当输入A命令后，光标自动移到该行的行尾。

4.3.3 打开命令

打开命令有两个，即o和O。

（1）o命令：在光标所在行的下面新开辟一行，随后输入的文本就插入在这一行上。

（2）O命令：在光标所在行的上面新开辟一行，随后输入的文本就插入在这一行上。

在新行被打开之后，光标停在新行的行首，等待输入文本。

光标位于“{”的位置。

例如，原来屏幕显示为：

```
/* this is an example */
{
printf("ok!");
}
~
```

光标所在位置。

输入O命令（大写字母）后，屏幕显示为：

```
/* this is an example */

{
printf("ok!");
}
~
```

4.3.4 输入方式下光标移动

在键盘的右下方有4个表示方向的方向键，利用它们可以在输入方式下移动光标。每按一次“上”“下”方向键，光标即相应地移动一行；每按一次“左”“右”方向键，光标即在当前行上相应地移动一个字符位置。当光标位于行首（或行尾）时，按下“左”方向键（或“右”方向键），系统会发出“嘟嘟”声，并且返回到命令方式。

4.3.4 输入方式下光标移动

例如，屏幕显示的正文行（刚插入的）是：

praem

连续按三次Backspace键后，显示为：

praem

按Esc键后，显示为：

pr

利用【Backspace】键（退格键）可将光标从当前行新插入的字符上回退一个字符。在按【Backspace】键（退格键）之后，字符从输入缓冲区中删除，但仍留在当前屏幕上，写入文件时被删除的字符就不存在了。

4.3.4 输入方式下光标移动

还可用下列一些组合键来移动光标。

（1）**【Ctrl+U】键**：将光标回退到刚插入字符串的第一个字符，删除刚插入的字符串，并重新开始插入。

例如，新插入的正文行是：

```
/* this is a program*/
```

按【Ctrl+U】键后，显示为：

```
/* this is a program*/
```

接着输入“int a,abc;”，按【Esc】键后，显示为：

```
int a,abc;
```

原来插入的内容消失了。

4.3.4 输入方式下光标移动

（2）【Ctrl+W】键：将光标移到后插入单词的首字符。

例如，新插入的正文如下：

/* this is a program*/

按【Ctrl+W】键后，显示为：

/* this is a program*/

4.3.4 输入方式下光标移动

（3）**【Ctrl+T】键：**在插入正文时，如果光标在当前行的开头，并且设置了自动缩进选项，那么这个命令就插入缩进所对应的空格。如果光标在新插入词的中间，设从该词开头至光标位的位移为k，缩进空格为n，那么这个命令就在光标插入**（n-k）个空格**；如果k大于n，则n扩大1倍。

例如，插入的行是：

```
{
doublevarl;
```

按【Ctrl+T】键后，显示为：

```
{
    double varl;
```

目录

本章要点

4.1 vi的工作方式

4.2 vi的启动和退出

4.3 文本输入

4.4 打开文件

4.5 保存文件

4.6 移动光标

4.7 文本修改

4.8 ex命令

4.4.1 打开一个文件

在vi编辑器下，打开文件是指将从磁盘中调入内存，以供vi使用。vi可以一次打开一个文件，也可以一次打开多个文件。

用vi打开文件的方法很简单，即在vi命令后面接上路径及文件名，然后按**【Enter】键**，就像下面这样：

```
vi  vi_test
```

4.4.1 打开一个文件

vi程序就会在指定的路径里寻找文件。如果没有指定路径，就默认为当前目录。

上面的操作可打开如下屏幕显示：

```
How  are  you!
This is thr first text made by vi!
It's very simple. Don’t you think so?
~
~
~
……
~
"vi_test" [已转换]   3L,85C                    1,1          全部
```

4.4.1 打开一个文件

vi将vi_test读入缓冲区中，并在屏幕上显示出来。

在底部的状态上显示：

"vi_test" [已转换] 3L,85C 1,1

这时，如果按a进入输入模式，则底部的状态行如下

~

~

—插入—

它标志着当前正在做的事情。

4.4.1 打开一个文件

注意

刚进入vi时，光标在第一个字母“H”的下面，按a进入输入模式后，此时的光标位于第二个字母“o”的下面。如果用户按的是i进入输入状态，则光标位于第一个字母“H”的下面。这就是用a与i的区别：a在光标所在的字母后面插入，而i在这个字母的前面插入。

在vi里怎么实现“另存为”呢？其实使用**“:w”命令**就可以了。在“:w”后面给定一个与用户打开的文件名字不一样的文件名，然后按**【Enter】键**即可。

4.4.2 打开多个文件

vi能够一次打开多个文件。打开多个文件的语法如下：

vi 文件1 文件2

在输入上述的命令之后，vi将第一个文件读入缓冲区，并将光标定位在左上角。用户可以输入 **“:next”命令**来编辑下一个文件。

前面所讲文件的操作对打开多个文件的情况仍然适用。事实上，在任何时刻，只有一个文件被读入到缓冲区，所以可以对该文件进行编辑、保存及删除，所有的这些操作对其他的文件并没有影响。

4.4.2 打开多个文件

例如：

```
[root@localhost root]#vi  file1 file2
```

按上面的操作打开如下屏幕显示：

```
#now I'm testing the multi-file operation of vi.
It's the first file-file1
Hello!
~
~
……
~
~
"file 1" [已转换]   4L,66C                 1,1          全部
```

vi立即读入了第一个文件，屏幕显示的信息就如同单独编辑file1一样。屏幕**没有**任何关于多个文件的**提示**。

4.4.2 打开多个文件

怎么样能够编辑后面的文件file2呢？在命令模式下，输入命令 **“:next” 命令**。

随即出现如下屏幕显示：

```
#
  This is the second file---file2.
  Ha, Ha
  Hello
~
~
~
……
~
"file2" [已转换]  5L,58C                1,1        全部
```

◆ **在屏幕的底部是关于file2的信息：5行，58个字符。**

4.4.2 打开多个文件

这时，同单独编辑file2完全一样。刚刚编辑过的file1在这里没有任何痕迹和提示。

vi作为一个**字符界面的编辑器**，其根本的缺点就是不可视性，无论它多么优秀，这也是没法避免的。

为了回到file1，再输入 **":next"** 试试。结果并没有见到file1，而在屏幕底部的状态行上出现 **"Cannot go beyond last file"** 的信息，意思是不能超过最后的文件。此时，输入 **":previous" 回车**，成功了。在vi编辑多个文件时，用 **":previous"** 回到前一个文件。

说明

为了简化输入，":previous"可以用":prev"代替。

4.4.2 打开多个文件

在打开多个文件时，仍然可以用上面讲的那些命令退出vi，但是会稍微有点不一样。

假设没有对原文做过什么改动，试一下 **“:q”** 退出。屏幕底部的状态行显示 **“1 more files to edit”** 信息，意思是还有一个文件等待编辑。用户关闭的这个文件并没有从屏幕上消失，下一个文件也没有被调入缓冲区。按 **【A】键**，仍然可以在光标处进行文本的插入。这是为什么？这要从vi操作多个文件的机理去寻找原因。

4.4.2 打开多个文件

vi的操作流程如下：

（1）vi在参数行上找到了**多个参数**就将参数记录下来。

（2）vi读入第一个参数指向的文件，如果这个文件不存在，就建立一个**空文档**。

（3）在用户对一个文件操作完毕，并发出正常退出的信息时，vi就检查各个参数所指的文件是否已经被编辑过。如果没有，就表示还有几个文件等待编辑，并忽略用户的这个操作。

（4）在显示**“E173：还有1个文件未编辑”**信息时，如果用户再次传递退出命令给vi，vi就认为一定要退出，就执行**退出命令**。

这就不难理解为什么会出现上面的情况了，即vi给出提示信息并将用户的输入命令忽略了。

4.4.2 打开多个文件

再次输入“:q”，退出了vi。

此时，屏幕应显示如下：

```
[root@localhost root]# vi  file1  file2
```

还有两个文件等待编辑

```
[root@localhost root]#
```

在用vi编辑单个文件时，退出后并没有看见什么信息。而在编辑多个文件时，vi输出“还有两个文件等待编辑”的信息，然后退出编辑。

4.4.3 用vi打开多个窗口

上面已经看到了如何用vi打开多个文件，但那是在一个窗口里打开。任何时刻只能有一个文件在窗口里显示。

vi也可以用多个窗口打开多个文件，只需给vi传递一个参数即可，其语法如下：

```
vi –o 文件1 文件2
```

前面所编辑的两个文件可以在两个窗口中进行编辑，命令如下：

```
[root@localhost root]# vi –o file1 file2
```

在用vi编辑单个文件时，退出后并没有看见什么信息。而在编辑多个文件时，vi输出“还有两个文件等待编辑”的信息，然后退出编辑。

4.4.3 用vi打开多个窗口

屏幕显示如下：

```
#now I'm testing the multi-file operation of vi.
It's the first file-file1 Hello!
~
~ ~
file1                    1,1          全部
#
  This is the second file---file2.
  Ha, Ha
  Hello
~
~ ~
file2                    1,1          底端
```

有两个亮条将屏幕分成两半。上一半是文件 **file1**，下一半是文件**file2**。亮条上写着对应的**文件名**。

如果这时输入命令，如“:w”，它就会显示在第二个亮条下面，即底部的状态行上。现在用户可以对文件进行任何前面讲到的操作了。

目录

本章要点

4.1 vi的工作方式

4.2 vi的启动和退出

4.3 文本输入

4.4 打开文件

4.5 保存文件

4.6 移动光标

4.7 文本修改

4.8 ex命令

经常保存文件绝对是个好主意。对于很重要的文件更应该多存几个文件。在用户启动vi时，vi将指定的文件读入**缓冲区**。用户随后的所有的录入、修改和删除等操作都是对缓冲区中的内容进行的。用户的输入内容不会被保存到磁盘上。如果在用户正操作时，突然计算机停电了，那么缓冲区作为内存的一部分，其中的内容也必然全部丢失。在下一次开机时，用户能够得到的只是上一次存盘时的文件。

在用户录入文本时，要不断地存盘，3个存盘的命令如下：

将缓冲区中的内容写到上一次指定的文件（上一次保存时的文件，如未保存，则是打开时的文件）中；

将缓冲区中的内容写到名为file的文件中；

强制将缓冲区中的内容写到file中。

1．保存

在一般情况下，用户应该不断地用**“:w”命令**来保存文件。在万一断电时，保存的频率越高，用户的损失就越小。不要担心存入了不想要的内容，vi不像Windows下的notepad（记事本）那样，**一次只能撤销一次操作**，很可能就得不到以前的内容了，vi可以很多次地撤销，一直到原始的状态。

2．另存为

在用户编辑很重要的文件时，用户就应该常常地**另存文件**。例如，用户在修改一个很复杂的shell脚本，每一次修改，用户都要保存用户的成果。但是，每一次修改都有可能是无用的，不能够执行。因此，用户应该保存原来的文件。用户还应该将每一步修改存成不同的文件。

2．另存为

用“:w file”**命令**很容易实现另存为的功能。只需要将file设为与原来的文件不一样的文件名，就另存为指定的文件了。

例如，用户打开的文件叫做play，用户使用“:w play1”**命令**就可将缓冲区中的内容另存为了play1文件。

如果用户另存为时指定的文件已经存在，则在vi的状态行会出现“**file exits(use！ to override)**”的信息，提示用户文件已存在，请用“**!** ”覆盖。

处理这种情况的方法是：首先，用户要确定原来的文件确实不想保留；然后，再输入“**w! file2**”**命令**来强制覆盖。

目录

本章要点

4.1 vi的工作方式

4.2 vi的启动和退出

4.3 文本输入

4.4 打开文件

4.5 保存文件

4.6 移动光标

4.7 文本修改

4.8 ex命令

1．用4个方向键移动光标

方向键是基本的移动方法。大多数的系统都是支持方向键的。应该说，方向键就是为了移动光标而诞生的。

方向键箭头的方向就是用户按下此键后光标**移动的趋势**。之所以说是“趋势”，是因为用户按了方向键，光标未必就会移动。在光标已经到了尽头时，用户按下光标只会听到“报警”的声音，光标不可能移动到屏幕外去，它就在原地不动。

2．用命令移动光标

在命令模式里，vi还可以使用命令来**移动光标**。这是因为在很早的时候，很多终端没有方向键，因此vi提供了一些命令来移动光标。常用的移动光标命令如表4-1所示。

表4-1 移动光标的命令

命令	作用
h	向左移动一个字符
j	向下移动一行
k	向上移动一行
l	向右移动一个字符

2．用命令移动光标

命令	作用
b	将光标移动到当前单词的第一个字母
e	将光标移动到当前单词的后一个字母
空格	向右移动一个字符
回车或+	将光标移动到下一行行首
－（减号）	将光标移动到上一行行首
0	将光标移动到行首
$	将光标移动到行尾
w	将光标向前移动一个字符（如果已到行首，就移动到上一行的行尾）
Backspace（退格）键	将光标向左移动一个字符

2．用命令移动光标

命令	作用
Shift+h	将光标移动到本屏幕的第一行
Shift+m	将光标移动到本屏幕上中间的一行
Shift+l	将光标移动到本屏幕的后一行
Ctrl+b	向下移动一屏
Ctrl+f	向上移动一屏

目录

本章要点

4.1 vi的工作方式

4.2 vi的启动和退出

4.3 文本输入

4.4 打开文件

4.5 保存文件

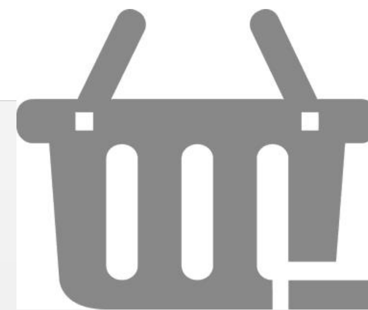
4.6 移动光标

4.7 文本修改

4.8 ex命令

4.7.1 删除与替换

1．删除



在输入模式下，可以用【Backspace】键来删除前面的字符，还可以用【Delete】键来删除当前的字符。

如果要删除一行，则只靠上面说的两个键显然是不够的。按n次“退格”键来删除一行的文字，显然是太累了。在vi的命令模式下还提供了几个命令来删除一个字符、一个单词或进行整行删除。

4.7.1 删除与替换

常用的删除命令如表4-2所示。

表4-2 常用删除命令

命令	作用
x	删除当前光标所在的字符
d w	删除当前光标所在单词字符至下一个单词开始的几个字符
d $（或 shift+d）	删除从当前光标至行尾的所有字符
d d	整行删除

4.7.1 删除与替换

两个字符排列时，如**dw**，表示先按下d键，然后再按下w键，而不需要两个键一起按。当按下某个键之后，vi会记下它，并等待继续输入其他的键。当连续输入的几个键与vi指定的某操作的命令键相符合时，vi就执行这个操作。

表4-2中的命令可以删除字符或是整行删除。用户还可以为它们指定要删除的字符或是行的数量。其用法是：

N "键组合"

其中，N是一个数字。

例如：

3 x **表示删除从当前光标向后共3个字符；**

4 dd **表示连着删除4行。**

4.7.1 删除与替换

为了方便更大范围的删除，vi还提供了一个以冒号开头的命令删除方式，用来删除整个一块区域的内容。其命令如下：

N1,N2 d

说明

在输入这些键组合时在屏幕上是没有任何显示的。

N1、N2是两个数字。N1是要删除的块区域的起始行的行号。N2是要删除的块区域的末尾行的行号。**d表示删除。**

例如：要删除从第1行至第12行的内容，输入命令如下所示：

:1,12 d

执行此命令后，从第1行至第12行的内容就全部被删除了，状态行上会显示总共删除了多少行。

4.7.1 删除与替换

在vi的命令模式下还提供了几个命令来替换一个字符、一个单词或是进行整行替换。常用的替换命令如表4-3所示。

表4-3 常用替换命令

命令	作用
r	替换光标所在的字符
R	替换字符序列
cw	替换一个单词
ce	同 cw
cb	替换光标所在的前一字符
c$	替换自光标位置至行尾的所有字符
cc	替换当前行
cb	替换光标所在的前一字符

4.7.1 删除与替换

例如，屏幕显示为：

/*this is abcd*/

输入rA后，屏幕显示为：

/*this is Abcd*/

对于表4-3中的命令，可以在vi编辑器中反复练习使用，从而达到熟练掌握的程度。

4.7.2 查找

查找显然也是一个成功的字处理软件必需提供的功能。vi提供的**字符串查找功能**，可使用户向前或向后查找，还可以继续上一次查找。当vi向后查找，找到文本的最后时，它就从文本开头的地方继续查找；同样，如果向前查找，找到了文本的开头，它就到文本的末尾继续查找。 **vi提供的几种查找命令如表4-4所示。**

表4-4 查找命令

命令	作用
?字符串	向后查找字符串
/字符串	向前查找字符串
n	继续上一次查找
N	以与上一次相反的方向查找

在用户输入“?”或“/”之后，字符就会显示在屏幕底部的状态行上。

4.7.1 删除与替换

光标停在Hello单词的第一个字母H上。

注意

在查找时，要注意字母的大小写，因为在vi编辑器中，大写字母与小写字母不是一个字母。

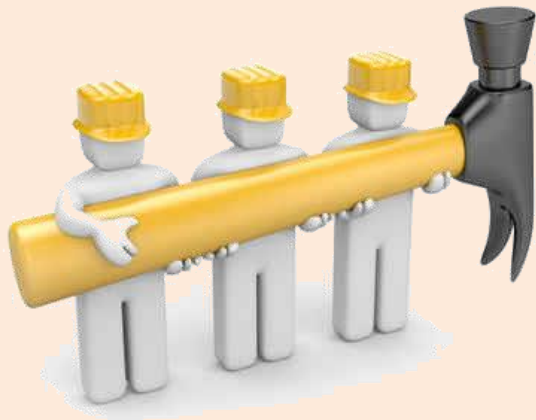
例如，在file1里查找Hello，输入“?Hello”并回车，则屏幕显示如下：

```
#now I'm testing the multi-file operation of vi.

It's the first file-file1
Hello!
~
~
```

4.7.3 复制、粘贴、剪切

如果要在vi编辑器里移动大块的文字，就不得不借助于复制、粘贴或者剪切功能了。



有过Windows使用经验的人都知道，Windows为所有的程序均提供了一个剪贴板。复制的动作就是将选定的内容送到了剪贴板；粘贴就是将剪贴板里的内容送到编辑器；剪切就是不仅将选项的内容送到剪贴板，而且将选定的文本在编辑器里删除。

在vi编辑器里的这些操作与Windows的**机理是相同的**，不同的是，Windows中的剪贴板由系统提供，剪贴板里的内容可以被其他的程序利用；而在vi编辑器中，**剪贴板是vi自己划出来的一块内存，其内容不能被其他的程序所利用。**

4.7.3 复制、粘贴、剪切

1．复制

在vi编辑器中，复制的方式有两种：**鼠标方式和命令方式。**

如果用户安装了鼠标，并且安装了鼠标驱动程序gpm，那么在vi编辑器里，用户可以**用鼠标来复制和粘贴文件**。在用户要复制的文本开始处按下鼠标的左键并拖动它，用户将看到被鼠标扫过的文字都由原来的黑底白字变成了白底黑字。亮条标志了用户选定的文本。在用户要复制文本的最后一个字上放开鼠标左键，就完成了对选定文本的复制。选定的文本就被送到了缓冲区。在用户松开鼠标的时候，白色亮条并不消失。

注意

但是要注意的是，用鼠标复制的功能是由系统提供的，与vi无关。所以，复制的内容并不是送到了vi的剪贴板上，而是在鼠标驱动程序gpm的剪贴板上，这种复制必须用鼠标来粘贴。

4.7.3 复制、粘贴、剪切

vi也提供了命令方式来复制文字，而且复制的对象也分为**单词和行**，其复制命令如表4-5所示。

表4-5 复制命令

命令	作用
yw	复制当前光标至下一个单词开始的内容
y$	复制当前光标至行尾的内容
yy 或 Y	复制整行

注意

在用组合键进行复制的时候，屏幕上不会有用鼠标复制时的白色亮条来提示用户究竟复制了哪些内容。

4.7.3 复制、粘贴、剪切

2．粘贴

不同的复制方法对应不同的粘贴方法。

如果用户是用鼠标复制的，那么在粘贴的时候，首先应该**用键盘**来移动光标，将光标移动至用户要粘贴的**位置后面**。

用命令方式的复制就应该用vi提供的粘贴命令**组合键来粘贴**。

vi的粘贴命令很简单，只有以下两个：

- **p**：在当前光标后面粘贴。
- **shift+h**：在当前光标前面粘贴。

注意

记住，一定要将光标放在那个位置的后面，因为鼠标操作会粘贴到光标所在位置的前面。

使用命令方式粘贴时，先将光标移至粘贴目标的位置，然后再按下相应的命令键就可以完成。

4.7.3 复制、粘贴、剪切

3．剪切

在vi编辑器中，所有的删除动作都是剪切。因为删除的内容都被送到了剪贴板。如果用户是用**组合键删除**，那么被删除的内容就能够被完整地粘贴。参见4.7.1节的删除内容，这里不再赘述。

4.7.4 重复

vi编辑器会记录上一次的操作，在用户需要重复这样的操作的时候，只需在命令模式下按一下【.】键就可以了。

例如，屏幕显示为：

```
#include <stdio.h>
main()
{
}
```

输入o命令，并插入一行正文printf()，按【Esc】键后，屏幕显示为：

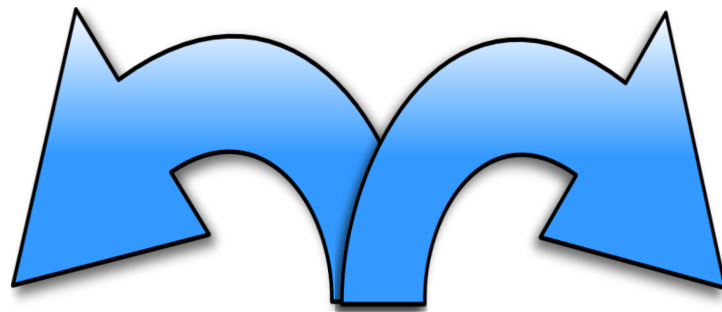
```
#include <stdio.h>
main()
{  printf();
}
```

连续输入两个“.”命令，屏幕显示为：

```
#include <stdio.h>
main()
{  printf();  printf();  printf();
}
```

这一个功能对于要通篇地进行某一重复操作的人来说，是非常有用的。

4.7.5 撤销



在vi编辑器中，可以取消后一次的操作。这对于一个编辑器来说是必需的。用户很可能不小心删除掉了不想删除的内容或者输入了一些错误的内容。如果没有撤销功能，将使用户非常被动。

撤销命令很简单，有以下两种形式：

- **u命令**：取消上次的操作。
- **U命令**：可以恢复对光标所在行的所有改变。

4.7.5 撤销

可以这样来记住这个命令：u就是undo的第一个字母。

例如，屏幕显示为：

```
#include  <stdio.h>
main()
{  printf("OK!");
}
```

输入dd命令后，屏幕显示为：

```
#include  <stdio.h>
{
}
```

接着输入u，屏幕显示为：

```
#include  <stdio.h>
main()
{
  printf("OK!");
}
```

目录

本章要点

4.1 vi的工作方式

4.2 vi的启动和退出

4.3 文本输入

4.4 打开文件

4.5 保存文件

4.6 移动光标

4.7 文本修改

4.8 ex命令

vi编辑器有三种工作方式，除命令方式、输入方式外，还有ex命令方式。进入ex命令方式后，就可以使用ex命令执行编辑处理了。进入ex命令的方法是，在命令方式下输入**冒号“：”**，即在状态行上出现冒号提示符，随后就输入**ex命令**。通常，ex命令用来写文件、读文件、跳到shell状态或者切换正在编辑的文件。

在默认情况下，很多ex命令完成的动作可影响到当前的文件，即影响vi命令之后输入的文件名。可以用**f命令**或者**n命令**更改当前的文件。所有的ex命令都可以用**【Enter】键**或者“**中断**”**键**予以中止。

4.8.1 命令定位

ex是面向行的编辑器，经常要将光标移到指定行。

其中的一种办法是**指定行号**，如输入**“:20+Enter键”**，即可将光标移到第20行的行首。

另一种办法是**给定模式**，向前或向后查找。例如，输入**“:/this?”**再按**【Enter】键**，从光标所在的行向前查找给定模式this，光标停在第一个与this匹配的行的行首。

输入“:?/this?”再按**【Enter】键**，则从光标所在的行向后查找给定模式this，光标停在首先找到的匹配行的行首。

4.8.1 命令定位

此外，ex命令还用下述字符指定行的地址：

- .：当前行，多数命令的默认地址是当前行。
- N：编辑器缓冲区中的第n行，行号从1开始顺序编排。
- $：缓冲区中的后一行。
- % 1,$：从第1行到后一行的缩写。
- +n或者-n：n表示相对当前行的位移。“.+3”，“+3”与“+++”三种形式等价。如果当前行是第100行，那么这三种形式都是定位在第103行。而-5定位在第95行。

4.8.1 命令定位

如果命令地址由一系列地址组成，如表示某个范围的正文行，那么各地址间就用**逗号“,”**或者**分号“;”**隔开。这种地址表是从左至右计算的。

例如：

:15,100d	**表示将删除第15行至第100行的正文。**
:.,+5d	**表示将删除当前行和它后面的5行。**

如果给出的地址多于命令所需的地址，那么除后一个或两个地址以外，其余的地址将全部被忽略。空地址通常以当前行代替，如“,100”等价于“.,100”。

4.8.2 常用ex命令

1．e命令

利用e命令可以在编辑当前文件时编辑另外的文件。当前文件名总是由vi记住，并用**百分号“%”**表示；而编辑缓冲区中的上一个文件名是用**“#”号**表示的。

e命令常用的形式如下：

（1）e文件名

它编辑由文件名指定的文件，不同于前面正在编辑的文件。编辑器首先检查自上次执行写（w）**命令**以来缓冲区内容是否被修改过。如果改过，则发出告警信息，并终止该命令。如未改过，那么就删除缓冲区中的全部内容，将指定的文件当作当前文件，并加以显示。确定该文件是可见文件之后（即它不是二进制、目录或设备文件），编辑器就将它读入缓冲区中。如果读文件过程中没有错误，就在状态行上显示所读的行数和字符数，然后就可以对这个文件进行编辑了，且光标停在文件的第一行上。

e命令

4.8.2 常用ex命令

1．e命令

e命令常用的形式如下：

它不将修改过的当前文件从编辑缓冲区中写出去，从而忽略在编辑新文件之前所做的全部修改。

(2)
e!文件名

(3)
e+n文件名

它从第n行开始编辑指定的文件。参数n也可以是不包含空格的编辑命令，如+/模式。

注意

按【Ctrl+ ^】键将返回到上一个编辑文件的先前位置，等价于“:e#”再按【Enter】键。

e命令

4.8.2 常用ex命令

2．w命令

w（写）命令可将编辑缓冲区中的全部或者部分内容写到当前文件或者另外某个文件中。它有以下几种常用的形式：

（1）w文件名

将所做的修改写回到指定的文件中，并显示所写的行数和字符数。通常，忽略文件名，缓冲区中的内容就写到了当前文件中。如果使用变种形式w!，就强行写出去。如果文件不存在，就创建它。

（2）w>>文件名

将缓冲区中的内容附加到现有文件的末尾，先前文件内容并不被破坏。

w命令

4.8.2 常用ex命令

2．w命令

（3）w! 文件名

可以跳过通常写命令对文件的检查，将缓冲区内容写到系统允许的任何文件上。

注意

感叹号“！”之后有空格。

（4）w !命令

将缓冲区中的内容附加到现有文件的末尾，先前文件内容并不被破坏。

注意

感叹号“！”之后有空格。

w命令

4.8.2 常用ex命令

3．r命令

r（读）命令将文本读入编辑缓冲区中的任意指定位置。所读入的文本必须至少有一行长，可以是一个文件或者命令的输出。它有以下几种常用的形式：

（1）r文件名

将指定文件的文本副本放入缓冲区中的指定行之后。如果没有指定文件，则使用当前文件名。如果没有当前文件名，则指定文件名就成为当前文件名。r命令前面可给出地址0，将文件读到缓冲区的开头。

（2）r !命令

将命令的输出读到缓冲区指定的行之后。

注意

感叹号“！”之后有空格。

r命令

4.8.2 常用ex命令

4．q命令

用q（退出）命令可实现从vi中退出来。它有下述几种使用方式：

（1）q

它的功能是退出vi。编辑器缓冲区中的内容并不会自动写到文件中。因此，输入q命令后，如果从上次利用w命令写文件以来该文件又做过修改，则vi就显示告警信息，并且不从vi中退出。vi也会显示在参数表中是否还有多个文件要进行编辑的诊断信息。

通常，用户会希望保存修改过的内容，那么应该在q之后输入w命令。如果不想保留所做的修改，那么就输入q!命令。

q命令

4.8.2 常用ex命令

4．q命令

（2）q!	（3）wq文件名	（4）wq!文件名	（5）x文件名
它的功能是立即从vi中退出，不保留所做的修改，也不显示任何提示信息。	它的功能等价于执行w命令后又执行q命令。	它的功能是忽略执行w命令之前通常所做的检查。例如，如果用户有一个文件，但没有打开它的写权限，那么wq!就允许用户用任何方式修改该文件。	它的功能是如果该文件做过修改，并且尚未写出去，那么这个命令就将缓冲区中的内容写出去，然后退出vi编辑器；否则，只是退出vi编辑器。

》 问题

1.进入和退出vi编辑器的方法有哪些?

2．vi编辑器的工作方式有哪些？相互间如何转换?

3．建立一个文本文件，将光标移至第5行。分别用c、C和cc命令进行修改。

4．在vi编辑器之下，将光标上、下、左、右移动的方式有哪些?

5．vi编辑器中复制一行文字并粘贴到另一位置用什么命令?

6．进入vi编辑器时，如果希望进入后光标位于文件中的第9行上，应该输入什么命令?

》 问题

7．不管文件中的某一行被编辑了多少次，总能将它恢复成被编辑之前的样子，应使用什么命令?

9．使用vi编辑器在目录里创建一个文本文件，然后输入一篇英文文章，并练习使用各种编辑命令。

7

8

9

8．要将编辑文件中的所有的字符串s1全部用字符串s2替换，包括在一行多次出现的字符串，应使用的命令格式是什么?

Linux操作系统及应用技术

shell编程

英文shell的本意是“壳”。它形象地说明了shell在Linux系统中的作用。shell就是紧紧包裹在Linux内核外面的一个**壳程序**。用户让操作系统做的所有任务，都是通过shell与系统内核的交互来完成的。shell所处的地位，就相当于DOS中的command.com程序，但比command.com的功能更加强大。

shell是用户与操作系统的内核之间的**接口**，是系统的**用户界面**，并且具有相当丰富的功能。利用shell可以编写出代码简捷，但功能很强的脚本文件。

目录

本章要点

5.1 shell概述

5.2 创建和执行shell脚本

5.3 shell特殊字符

5.4 shell变量

5.5 正则表达式与算术运算

5.6 控制结构

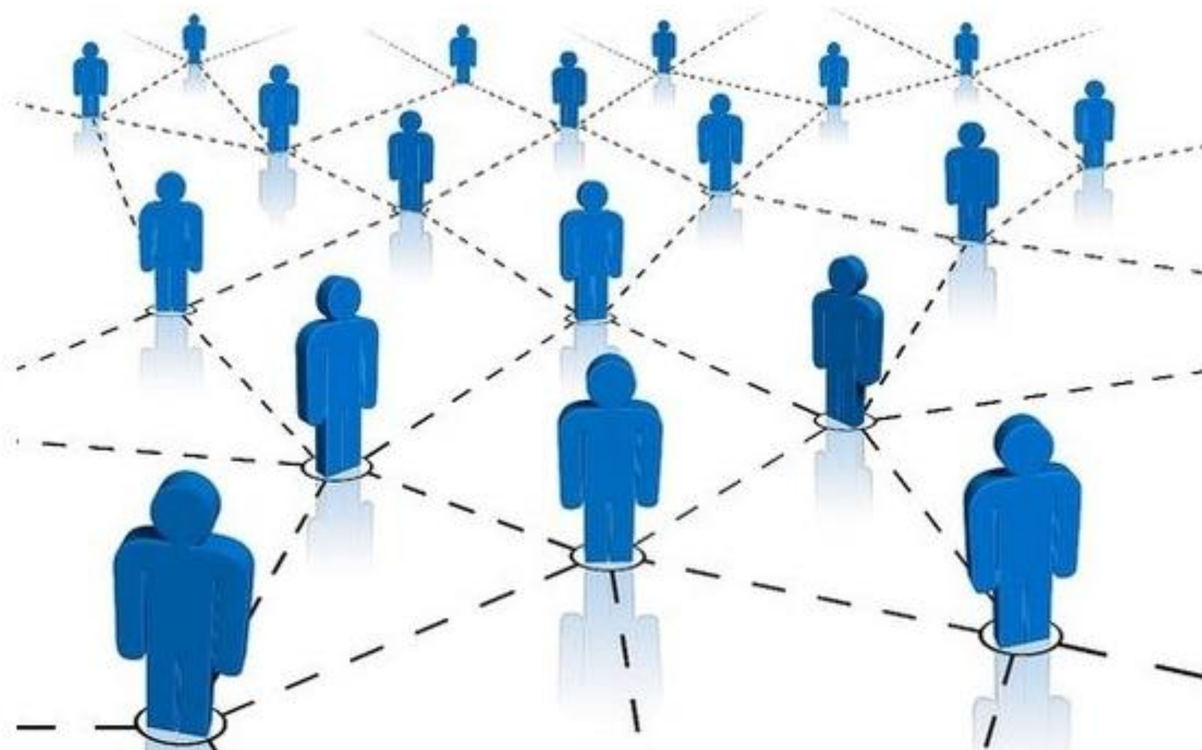
5.7 其他语句

5.8 函数

5.9 调试shell脚本

5.10 实例

1．shell的特点



Linux系统为用户提供了shell高级程序设计语言，大大方便了管理人员对系统的维护和普通用户的应用开发，提高了编程效率。

shell具有如下特点：

（1）对已有命令进行适当组合，构成**新的命令**，而且组合方式很简单。

（2）提供了**文件名扩展字符**（通配符，如*，?，[]），使得用单一的字符串可以匹配多个文件名，省去了键入一长串文件名的麻烦。

（3）可以直接使用shell的**内置命令**，而不需创建新的进程，如shell中提供的cd、help、kill、pwd、echo、alias、unalias、bg、fg、exit、export、fc、read、readonly等命令。为防止因某些shell不支持这类命令而出现麻烦，许多命令都提供了对应的二进制代码，从而也可以在新进程中运算。

（4）允许灵活地使用**数据流**，提供通配符、输入输出重定向、管道线等机制，方便了模式匹配、I/O处理及数据传输。

shell具有如下特点：

5

（5）**结构化的程序模块**，提供了顺序流程控制、分支流程控制及循环流程控制等。

6

（6）提供了在后台（&）执行命令的能力。

7

（7）提供了**可配置的环境**，允许用户**创建和修改**命令、命令提示符及其他的系统行为。

8

（8）提供一个**高级的命令语言**，允许用户能创建**从简单到复杂**的程序。这些shell程序被称为shell脚本。利用shell脚本，可将用户编写的可执行程序与Linux命令结合在一起，可以作为新的命令使用，从而便于用户开发新的命令。

2．shell的主要版本

在Linux系统中，常见的shell版本有以下几种：

（1）Bourne shell（sh）：

它是UNIX最初使用的shell，并且在每种UNIX上都可以使用。它在shell编程方面相当优秀，但处理与用户的交互方面不如其他几种shell。

（2）C shell（csh）：

它最初由Bill Joy编写，更多地考虑了用户界面的友好性，支持命令补齐等一些Bourne shell所不支持的特性，但其编程接口做得不如Bourne shell。Cshell被很多C程序员使用，因为C shell的语法和C语言的很相似，故C shell也由此得名。

2．shell的主要版本

（3）Korn shell（ksh）：

它集合了C shell和Bourne shell的优点，并且与Bourne shell完全兼容。

说明

在Red Hat Linux中，系统默认提供给每个用户的shell是bash。

（4）Bourne Again shell（bash）：

bash是大多数Linux系统的默认shell。它是Bourne shell的扩展，并与Bourne shell完全向后兼容，而且在Bourne shell的基础上增加和增强了很多特性。Bash放在/bin/bash中，可以提供自动补全命令行、命令行编辑及命令历史列表等功能。它还包含了很多Cshell和Korn shell中的优点，有灵活和强大的编程接口，同时又有很友好的用户界面。

2．shell的主要版本

（5）tcsh

它是**Cshell**的一个**扩展版本**，与csh完全向后兼容，包含了更多方便用户使用的新特性，在命令行编辑和历史浏览方面有更大的提高。它不仅与Bash shell提示符兼容，而且还提供比Bash shell更多的提示符参数。

（6）pdksh：

pdksh：它是一个专门为Linux系统编写的**Korn shell（ksh）的扩展版本**。Ksh是一个商用shell，不能免费提供，而pdksh是免费的。

此外，其他常见的shell还有**ash**、**zsh**等。

目录

本章要点

5.1 shell概述

5.2 创建和执行shell脚本

5.3 shell特殊字符

5.4 shell变量

5.5 正则表达式与算术运算

5.6 控制结构

5.7 其他语句

5.8 函数

5.9 调试shell脚本

5.10 实例

1．建立shell脚本

shell脚本（shell script）是指使用用户环境shell提供的语句所编写的**命令文件**。shell脚本可以包含任意从键盘输入的Linux命令。

建立shell脚本的步骤与建立普通文本文件的方式相同，利用**文字编辑器（如vi）**进行程序录入和编辑加工。例如，建立一个名为examplel的shell脚本，可在提示符后输入命令：

```
$vi  example1
```

进入vi的**插入方式**后，就可录入程序行。完成编辑之后，将编辑缓冲区中的内空写入文件中，返回到shell命令状态。

如果经常用到相同执行顺序的操作命令，便可以将这些命令写成**脚本文件**。这样以后要做同样的事情时，只要在命令行输入其文件名即可。

【例5-1】显示当前的日期时间、执行路径、用户账号及所在的目录位置。

在命令行中输入： $ vi example1

在vi编辑器中输入下列内容：

```
#!/bin/bash
#This script is a test!
echo –n "Date and time is :"
date
echo  –n  "The executable path is :"$PATH
echo  "Your name is :`whoami`"
echo  -n  "Your current directory is :" pwd
#end
```

文件中以“#”**开头**的行是注释行，在执行时会被忽略。特别是其中的第1行“#!/bin/bash”，用来指定脚本以bash执行。如果要设置以tcsh执行，则应设成“#!/bin/tcsh”。要指定执行的shell时，一定要将它写在第1行；如果没有指定，则以当前正在执行的shell来解释。

echo命令用来显示提示信息，参数“-n”表示在显示信息时不自动换行（默认会自动换行），如第3行的“echo -n……”，表示此行输出后不换行。下一行的date命令执行结果就会接在“Date and time is:”之后显示。

第6行的`whoami`字符串左右的倒引号（`）用于**命令转换**，也就是将它所括起来的字符串视为**命令执行**，并将其输出字符串在原地展开。第3行也可以改写成与第6行相同的写法“echo Date and time is : `date`”。

2．执行shell脚本

执行shell脚本的方式基本上有下述3种：

◆（1）输入重定向到shell脚本

这种方式是用输入重定向方式让shell从给定的文件中**读入命令行**，并进行相应处理。

其一般形式是： `$bash<脚本名`　　**例如：** `$bash <example1`

shell从文件example1中读取**命令符**，并执行它们。当shell到达文件末尾，就终止执行并将控制返回到**shell命令状态**。此时，脚本名后面不能带参数。

2．执行shell脚本

（2）以脚本名作为参数

其一般形式是：`$bash脚本名 [参数]`　　例如：`$bash example1`

其执行过程与上一种方式一样。这种方式的好处是，能在脚本名后面**带有参数**，从而将参数值传递给程序中的命令，使一个shell脚本可以处理多种情况，就如同函数调用时可根据具体问题给定相应的实参。

如果以当前shell执行一个shell脚本时，则可以使用如下简便形式：

`$. 脚本名 [参数]`

以shell脚本作为shell的命令行参数，这种方式可用来进行**脚本调试**。

2．执行shell脚本

◆（3）将shell脚本的权限设置为可执行，然后在提示符下直接执行它

通常，用户是不能直接执行由**文本编辑器**（如vi）建立的shell脚本的，因为直接编辑生成的脚本文件没有“执行”权限。如果要将shell脚本直接当作命令执行，就需要利用**命令chmod**将它置为有“执行”权限。

例如,

“$chmod a+x example1”就把shell脚本example1设置为对所有用户都有“执行”权限。然后，在提示符后输入脚本名example1就可直接执行该文件了。

例如：

```
$example1
```

2．执行shell脚本

◆（3）将shell脚本的权限设置为可执行，然后在提示符下直接执行它

shell**接收用户**输入的命令（脚本名），并进行**分析**。如果文件被标记为可执行的，但不是被编译过的程序，shell就认为它是一个shell脚本。shell将读取其中的内容，并加以解释执行。所以，从用户的观点来看，执行shell脚本的方式与执行一般的**可执行文件的方式相似**。因此，用户开发的shell脚本可以驻留在命令搜索路径的目录下（通常是“/bin”，“/usr/bin”等），像普通命令一样使用。

目录

本章要点

5.1 shell概述

5.2 创建和执行shell脚本

5.3 shell特殊字符

5.4 shell变量

5.5 正则表达式与算术运算

5.6 控制结构

5.7 其他语句

5.8 函数

5.9 调试shell脚本

5.10 实例

在Linux shell中，某些字符具有特殊的含义，所以利用它们作为变量名或者字符串的一部分时，会导致程序出现错误。当字符串含有这类特殊字符时，必须使用**转义字符（反叙杠）**来指明这些特殊字符不作为特殊字符来处理。在使用时应注意它们表示的意义和作用范围。

5.3.1 引号

在shell中的引号分为3种，即双引号、单引号及倒引号。

1．双引号

说明

转义字符告诉shell，不要对其后面的那个字符进行特殊处理，只将其当作普通字符。

由双引号括起来的字符，除**$、倒引号（`）及反斜杠（\）**仍保留其特殊功能外，其余字符均作为**普通字符**对待。

$表示变量**替换**，即用其后指定的变量的值来代替$和变量；倒引号表示**命令替换**；仅当“\”后面的字符是下述字符之一时，“\”才是转义字符。这些字符是$、倒引号（`）、”、反斜杠（\）或换行符。

【例5-2】双引号的作用。

```
[Stu@localhost wuxxy]$cat example2
echo  "current directory is `pwd`"
echo  "home directory is $HOME"
echo  "file*.?"
echo  "directory '$HOME'"
[Stu@localhost  Stu]$bash example2
current directory is /home/Stu/pro
home directory is /home/Stu
file*.?
directory '/home/Stu'
```

由脚本example2看出，在第1个echo语句中，在用双引号括起来的字符串中包含**`pwd`**。执行该语句时，先做用倒引号括起来的命令pwd，并将执行结果**代替**`pwd`。所以，得到第一行的输出结果。

第2个echo语句中，在双引号中有**$HOME**，执行时，先以HOME环境变量的值代替$HOME，然后显示整个参数字符串。

第3个echo语句中，在双引号中的字符都作为**普通字符**出现，所以执行结果出现所示的第3行输出。

第4个echo语句中，在双引号中有‘$HOME’，此时，单引号仍作为**普通字符**出现，而$HOME表示引用HOME的值。因而有执行结果的第4行输出。

5.3.1 引号

2．单引号

用单引号括起来的字符都可作为普通字符出现。

例如：

```
[Stu@localhost  Stu]$str='echo "directory is $HOME"'
[Stu@localhost  Stu]$echo  $str
echo "directory is $HOME"
```

其结果是将字符串“echo "directory is $HOME"”作为整体**赋给变量str**。由于使用了单引号，所以命令名echo及$HOME都可作为普通字符，失去了原有的特殊意义。

又如：

```
[Stu@localhost  Stu]$echo 'The time is 'date', the file is $HOME/abc'
The time is 'date', the file is $HOME/abc
```

可见，echo命令行中被用单引号括进来的所有字符都照原样显示出来，特殊字符也失去原来的意义。

5.3.1 引号

3．倒引号

利用倒引号的这种功能可以进行命令置换，即把用倒引号括起来命令的执行结果赋给指定变量。

例如：

```
[Stu@localhost pro]$today=`date`
[Stu@localhost pro]$echo  Today is $today
Today  is  一  1月  31   10:20:23  CST  2005
```

又如：

```
[Stu@localhost pro]$user=`who|wc –l`
[Stu@localhost pro]$echo  The number of users is $suers
The number of users is 3
```

5.3.1 引号

3．倒引号

可以看出，进行命令置换时，用倒引号括起来的可以是**单个命令**，也可以是**多个命令的组合**，如管道线等。另外，倒引号还可以嵌套使用。但应注意，嵌套使用时，内层的倒引号必须用反斜线（\）将其转义。

用倒引号括起来的字符串被shell解释为命令行。在执行时，shell会先执行该命令行，并以它的标准输出结果取代整个倒引号部分。这在前面的示例中已经见过。例如：

```
[Stu@localhost pro]$Nuser=`echo The number of users is \`who|wc -l\``
[Stu@localhost pro]$echo  $Nuser
The number of users is 3
```

5.3.1 引号

3．倒引号

如果内层倒引号不用其转义形式，而直接以原型出现在该字符串中，则成为以下形式：

```
[Stu@localhost pro]$Nuser=`echo The number of users is `who|wc -l``
```

按【Enter】键后，将出现：

```
0
```

接着输入：

```
[Stu@localhost pro]$echo  $Nuser
```

显示一个空行。这表明，它没有按想象的情况执行。

5.3.1 引号

4．反斜杠

反斜杠是转义字符。它能将特殊字符变成普通字符。在某个字符前面，利用反斜杠（\）能够阻止shell将后面的字符解释为特殊字符。例如：

```
[Stu@localhost pro]$echo  “Filename is N0\$\*”
```

则显示：

```
Filename is N0$*
```

如果想在字符串中使用特殊字符，如\、$等，则必须采用“\特殊字符”的形式。第一个反斜杠作为转义字符，将第二个字符变为普通字符。

5.3.2 输入/输出重定向

在Linux系统中，执行一个shell命令时，通常会自动打开**3个标准文件**：标准输入文件，通常对应终端的键盘；标准输出文件和标准出错输出文件，这两个文件都对应终端的屏幕。由父进程创建子进程时，子进程就**继承了**父进程打开的这3个文件，因而可以利用键盘输入数据，从屏幕上显示计算结果及各种信息。

在shell中，这三个文件都可以通过重定向符进行重新定向。

1．输入重定向符

输入重定向符“<”的作用是将命令（或可执行程序）的标准输入**重新定向**到指定文件。

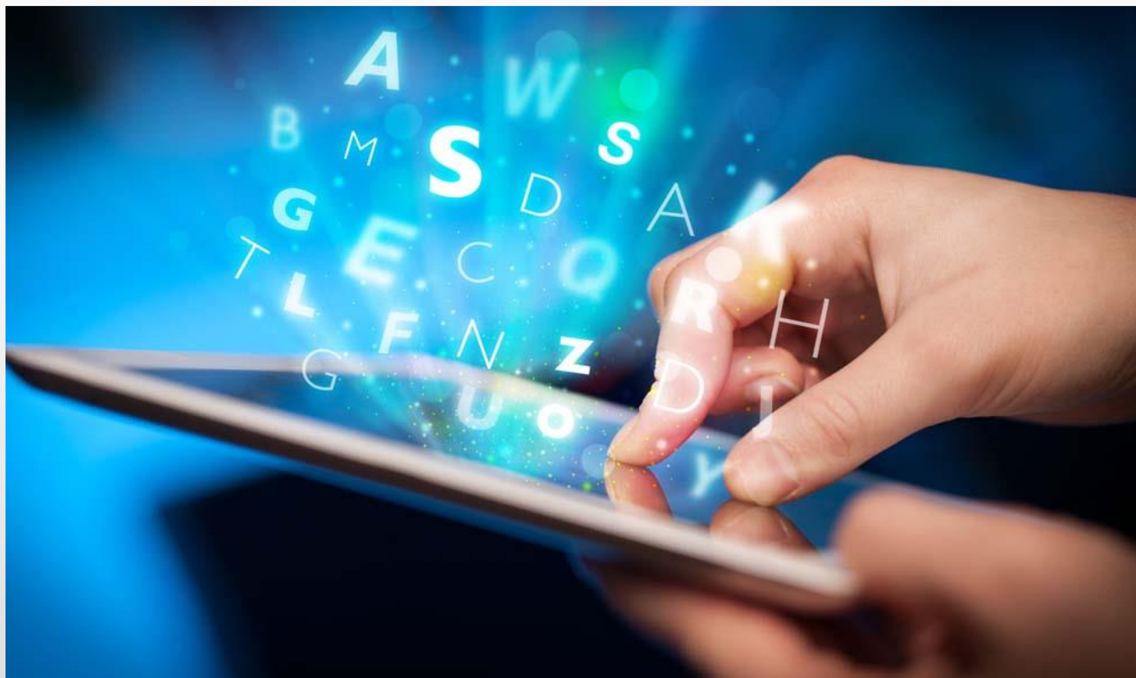
输入重定向的一般形式： **命令<文件名**

5.3.2 输入/输出重定向

1．输入重定向符

执行score的命令行可以是：

[Stu@localhost pro]$score <infile



例如，有一个可执行的程序score，其源程序用C语言编写，为了输入数据，使用了scanf函数调用语句。如果所需数据（如成绩表）预先已录入到一个文件infile中，那么就可以让score执行时直接从文件infile中读取相应的数据，而不必交互式地从键盘上输入。

5.3.2 输入/输出重定向

1．输入重定向符

如果程序所需输入数据较多或者会**被反复执行**，那么采用输入重定向方式就很用。另外，需要经常执行的shell命令也可放进一个文件，并且让shell从该文件中读取这些命令。

例如，在一个文件example3中包含以下命令：

```
echo  "your working directory is `pwd`"
echo  "your name is `logname`"
echo  "your time is `date`"
```

然后输入：

```
[Stu@localhost pro]$bash < example3
```

shell命令解释程序将从文件example3中读取命令行，并加以执行。这正是前面所说的执行shell脚本的一种方法。

5.3.2 输入/输出重定向

2．输出重定向符

输出重定向的一般形式：**命令>文件名**

输出重定向符“>”的作用是将命令（或可执行程序）的标准输出**重新定向到指定文件**。这样，该命令的输出就不在屏幕上显示，而是写入到指定文件中。

其中，文件名可以是普通文件名，也可以是对应于I/O设备的特别文件名，如打印机。例如：

```
[Stu@localhost pro]$who >outfile
```

命令who的输出重新定向到outfile文件中，在屏幕上看不到who的执行结果。查看outfile文件的内容，就可以得到who命令的输出信息。

```
[Stu@localhost pro]$cat  outfile
Stu tty1      jan 31 10:06
root tty2     jan 31 10:25
wxy tty3      jan 31 10:25
test tty5     jan 31 10:26
root :0       jan 31 09:46
root pts/0    jan 31 09:06
root pts/1    jan 31 10:12
```

5.3.2 输入/输出重定向

2．输出重定向符

shell脚本的输出也可重新定向到指定文件。例如，有一可执行shell脚本example4，其内容如下：

```
echo  "The time is `date`"
echo  "Your name is `logname`"
echo  "Working directory is `pwd`"
```

执行这个脚本：

```
[Stu@localhost pro]$example4 > tmp1
[Stu@localhost pro]$cat  tmp1
The time is  一  1月 31  16:30:00  CST  2005
Your name is Stu
Working directory is /home/Stu/pro
```

执行第一个命令行后，屏幕上没有显示任何信息。执行第二个命令行，将**重定向**的目标文件显示出来，正是example4脚本的执行结果。

5.3.2 输入/输出重定向

3．输出附加重定向符

输出附加重定向符“>>”的作用是将命令（或可执行程序）的输出附加到指定文件的后面，而该文件原有的内容**不被破坏**。

输出附加重定向的一般形式： **命令>>文件名**

例如：

```
[Stu@localhost pro]$ ps –a >> psfile
```

将**ps命令**的输出附加到文件psfile的结尾处。利用**cat命令**就可以看到文件psfile的全部信息，包括原有内容和新添内容。

5.3.2 输入/输出重定向

4．即时文件定向符

即时文件是由重新定向符“<<”、一对标记符及其中的若干输入符组成。它允许将shell程序的输入行重新定向到一个命令中。

即时文件的形式为：

命令 [参数] <<标记符
……输入行
标记符

例如

```
[Stu@localhost pro]$mail $1
<< !!
Best wishes to you on your birthday.
!!
```

其中，**标记符**是！！，它要成对出现。从“<<”之后的“！！”标记输入行开始，到后面的“！！”标记时文件结束。标记符也可以是别的能明显识别的符号，如%，甚至可以是用双引号括起来的字符串。如果没有用第二个标记符作为结束符，则当遇到文件末尾，同样也可以结束即时文件

5.3.2 输入/输出重定向

4．即时文件定向符

即时文件能使相应命令的输入重新定向，使它的输入到自两个标记符之间的若干输入行。如执行上例时，命令mail就将一对“！！”之间的输入行送给$1所对应的收信人。

输入输出重新定向可以连在一起使用。例如：

```
[Stu@localhost pro]$wc  -l<infile  >outfile
```

其功能是，命令wc从文件infile中输入信息，将按“行”统计后的结果送到另一个文件outfile中，并不在屏幕上显示。

5.3.2 输入/输出重定向

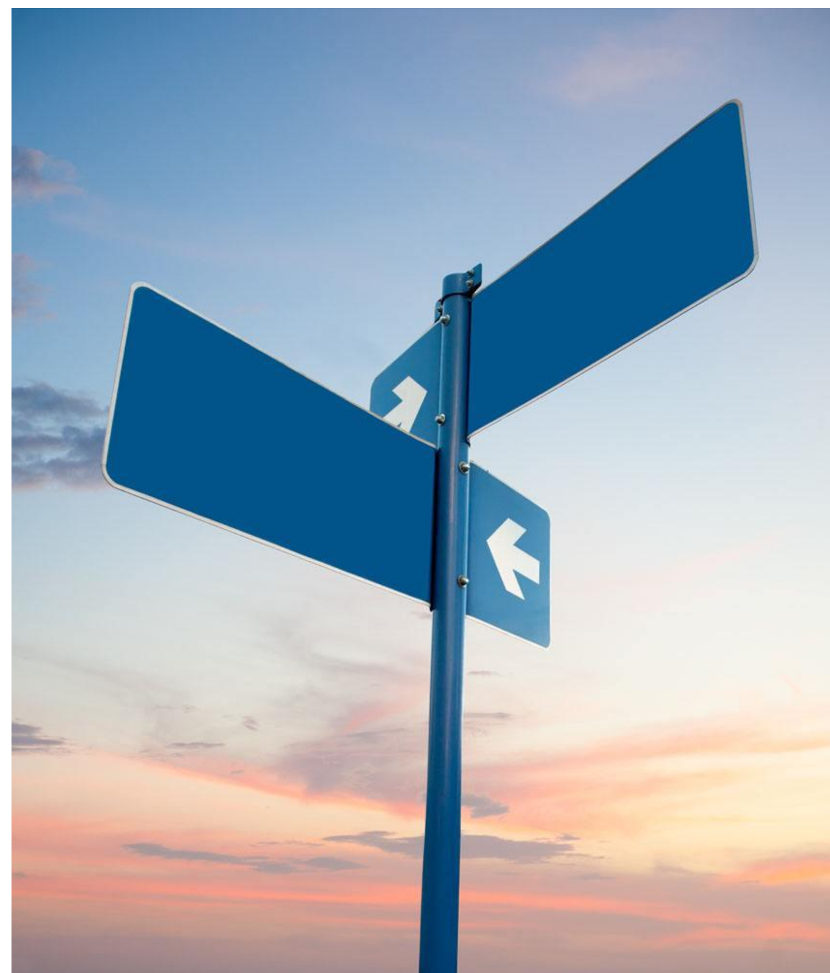
4．即时文件定向符

即时文件能使相应命令的输入重新定向，使它的输入到自两个标记符之间的若干输入行。如执行上例时，命令mail就将一对“！！”之间的输入行送给$1所对应的收信人。

输入输出重新定向可以连在一起使用。例如：

```
[Stu@localhost pro]$wc -l<infile >outfile
```

其功能是，命令wc从文件infile中输入信息，将按“行”统计后的结果送到另一个文件outfile中，并不在屏幕上显示。



5.3.2 输入/输出重定向

4．即时文件定向符

即时文件能使相应命令的输入重新定向，使它的输入到自两个标记符之间的若干输入行。如执行上例时，命令mail就将一对“！！”之间的输入行送给$1所对应的收信人。

其功能是，命令wc从文件infile中输入信息，将按“行”统计后的结果送到另一个文件outfile中，并不在屏幕上显示。

输入输出重新定向可以连在一起使用。例如：

```
[Stu@localhost pro]$wc  -l<infile  >outfile
```

5.3.3 注释和管道线

1．注释

在shell脚本中，以“**#**”**开头**的正文行表示注释。

如果shell脚本中的第一行是以“#！”开头的，则“#！”后面所跟的字符串就是所使用shell的**绝对路径**。

例如，对于C shell脚本，第一行通常是：

```
#! /bin/csh
```

对于bash脚本，第一行通常是：

```
#!/bin/bash
```

这一行说明该脚本是用哪一种shell编写的，从而可调用相应的解释程序予以执行。

5.3.3 注释和管道线

2．管道线

在Linux系统中，管道线是由竖杠（|）隔开的若干个命令组成的序列。在管道线中，每个命令执行时都有一个独立的进程，前一个命令的输出正是下一个命令的输入。

例如：

[Stu@localhost pro]$ls -l | wc -l

在管道线中，有一类命令也被称为**过滤器**。过滤器首先读取输入，然后将输入以某种简单的方式进行变换，再将处理结果输出，如grep、tail、sort和wc等命令就被称为过滤器。

一个管道线中可以包括多条命令，如：

[Stu@localhost pro]$ls
| grep m*.c | wc –l

5.3.4 命令执行操作符

多条命令可以在一行中出现。它们可以顺序执行，也可能在相邻命令之间存在逻辑关系，即逻辑与和逻辑或。

1．顺序执行

如前面所述，每条命令或管道线可单独占一行，如：

```
[Stu@localhost pro]$pwd
[Stu@localhost pro]$who|wc-l
[Stu@localhost pro]$cd  ./usr/bin
```

很显然，这些命令按其出现的**顺序依次执行**。

也可将这些命令在一行中输入，此时，各条命令之间应以分号“;”隔开，如：

```
[Stu@localhost pro]$pwd ; who | wc –l ; cd ./usr/bin
```

在执行时，以分号隔开的各条命令从左到右依次执行，即前面命令执行成功与否，并**不影响其后面**命令的执行。它与上面写成多行的形式是**等价的**。

5.3.4 命令执行操作符

2．逻辑与

逻辑与操作符“&&”可将两个命令联系在一起。

其一般形式为： **命令1 && 命令2**

功能是，先执行命令1，如果执行成功，才执行命令2；否则，若命令1执行不成功，则不执行命令2。

例如：

```
[Stu@localhost pro]$cp  example1  example11  &&  rm example1
```

功能是，先执行命令1，如果执行成功，才执行命令2；否则，若命令1执行不成功，则不执行命令2。

用“&&”可以将多个命令联系起来，格式如下：

命令1 &&命令2 &&命令3……&&命令n

5.3.4 命令执行操作符

3．逻辑或

逻辑或操作符“||”可将两个命令联系起来。

其一般形式为： **命令1 ||命令2**

功能是，先执行命令1，如果执行不成功，则执行命令2；否则，若命令1执行成功，则不执行命令2。

例如： **[Stu@localhost pro]$cat example11 || pwd**

表示如果不能将文件example11的内容显示出来，则显示当前工作目录的路径

同样，利用“||”也可以将多个命令联系起来，格式如下：

命令1||命令2||命令3……||命令n

目录

本章要点

5.1 shell概述

5.2 创建和执行shell脚本

5.3 shell特殊字符

5.4 shell变量

5.5 正则表达式与算术运算

5.6 控制结构

5.7 其他语句

5.8 函数

5.9 调试shell脚本

5.10 实例

Linux的shell编程是一种非常成熟的编程语言，它支持各种类型的变量，主要有3种，即用户变量、内部变量及环境变量。

- **用户变量**是在编写shell脚本时定义的。可以在shell程序内任意使用和修改它们。
- **内部变量**是由系统提供的。与环境变量不同，但用户不能修改它们。
- **环境变量**是系统环境的一部分，不必去定义它们。可以在shell程序中使用它们，某些变量（如PATH）还能在shell中加以修改。

5.4.1 用户变量

1．变量名

用户定义的变量是普通的shell变量。变量名是以字母或下线符开头的字母、数字及下线符序列，并且大小写字母意义不同。例如，dir与Dir是两个不同的变量名。变量名的长度不受限制。

2．变量赋值

给变量赋值的过程也是声明一个变量的过程。变量的赋值很简单。

其一般形式是：

```
变量名=字符串/数字
```

例如：

```
lcount=0
myname=wangtong yourname="li dong"
```

5.4.1 用户变量

3．访问变量值

可以通过给变量名加上**前缀$**（美元符）来访问变量的值。也就是说，假设变量名为myname，那么使用$myname就可以访问这个变量。

如果要将myname的值**分配给**变量yourname，那么可以执行下面的命令：

```
yourname=$myname
```

5.4.2 内部变量

内部变量是Linux所提供的一种**特殊类型的变量**。这类变量在程序中用来做出判断。在shell程序内，这类变量的值是不能修改的。

参数

- **$#:** 传送给shell程序的位置参数的**数量**。
- **$?:** 后命令的完成码或者在shell程序内部执行的shell程序（返回值）。
- **$0:** shell程序的**名称**。
- **$*:** 调用shell程序时所传送的全部参数组成的单字符串。

5.4.2 内部变量

【例5-3】有一个shell脚本mypgn2。其内容如下：

```
#!/bin/bash
#my test program
echo  "Number of parameters is"$#
echo  "Program name is "$0
echo  "Parameters as a single string is "$*
```

执行这个脚本：

```
[Stu@localhost pro]$bash  mypgm2  Sanjiv Guha
```

那么会得到下面的结果：

```
Number of parameters is 2
Program name is mypgn2
Parameters as a single string is Sanjiv Guha
```

5.4.3 环境变量

在用户注册过程（会话的建立过程）中，系统需要做的一件事就是建立**用户环境**。所有的Linux进程都**各自独立**，并且不同于程序本身的环境。Linux环境（也称为shell环境）由许多变量及这些变量的值组成。由这些变量和变量的值决定环境外观。这些变量就是**环境变量。**

shell环境变量名由**大写字母或数字**组成。有许多变量是在注册过程中定义的，一些为只读变量，意味着不能改变这些变量；而另外一些为非只读变量，可以由你随意增加或修改。

5.4.3 环境变量

下面简单介绍主要环境变量的使用。

1

HOME：

用户目录的**全路径名**。主目录是用户开始工作的位置。在一般情况下，如果用户的注册名为exuser，则HOME的值为/home/exuser。不管用户的当前路径在哪里，都可以通过下述命令返回到主目录：

2

LOGNAME：

用户的注册名，由Linux自动设置。它是系统与用户交互的名字或字符串。

5.4.3 环境变量

3 MAIL：

用户系统信箱的路径。无论何时邮件到达用户的系统中，它都会存在该变量指定的文件中。用户可以通过定时查询这个文件最近更新的时间来判断是否有新邮件到达。在一般情况下，如用户的注册名为pb，则MAIL的值为/var/spool/mail/pb。

4 SHELL：

shell从中查找命令的**目录列表**。这是一个非常重要的shell变量。PATH变量包含带冒号分界符的字符串。这些字符串指向含有用户所使用命令的目录。用户可以设置它，而且其中的字符串顺序决定了先从哪个目录查找。

5 PS1：

shell的**主提示符**。主提示符是在shell准备接收命令时显示的字符串。PS1定义用户的主提示符是如何构成的。

5.4.3 环境变量

6 **PWD：**

用户当前工作目录的路径。它指出用户目前在Linux文件系统中处在什么位置，是由Linux自动设置的。

7 **PATH：**

用户当前使用的shell。它也指出用户shell解释程序放在什么地方。

8 **TERM：**

用户终端类型。DEC公司制定的vt-100终端的特性，被许多厂商接受，也被许多终端软件仿真，成为广泛使用的标准设置。

5.4.4 位置参数

1．位置参数及引用

可以编写一个**shell脚本**，当从命令行或者从其他shell脚本中调用它时，这个脚本可以接收若干参数。这些选项是通过Linux作为位置参数（positional parameter）提供给shell脚本的。在shell脚本中应有变量，接收实参。

这类变量的名称很特别，分别是1，2，3，...称为**位置变量**。位置参数1存放在位置变量1中，位置参数2存放在位置变量2中，...在程序中可以使用$1，$2，...来访问，依此类推。

5.4.4 位置参数

【例5-4】位置参数的使用。

wm1.c的内容如下：

```
/* this is wm1.c */
main()
{
printf("Beigin\n");
}
```

wm2.c 的内容如下：

```
/* this is wm2.c */
main()
{
printf("OK!\n");
printf("End\n");
}
```

example4的内容如下：

```
#exam6:shell script to combine files and count lines
cat  $1  $2  $3  $4  $5  $6  $7  $8  $9  |wc  -l
# end
```

```
[Stu@localhost pro]$bash  example4  wm1.c  wm2.c
11
```

上面的shell脚本中使用了$1～$9共9个**位置参数**，因此执行时，可以接收**9个文件名**。而实际使用时只给出了两个文件名，分别对应$1和$2。这样，$3～$9等位置参数并没有被指定具体值，shell就将它们当作**空字符串**处理。

5.4.4 位置参数

2．用set命令为位置参数赋值

在shell程序中，可以利用set命令为位置参数赋值或重新赋值。

set命令的一般格式如下：

set ［参数表]

该命令后面无参数时，将显示系统中的系统变量值；如果有参数，将分别给位置参数赋值。

【例5-5】用set命令设置位置参数值，文件名为example5。

```
#! /bin/bash
#The file is example5
# example5:shell script to combine files and count lines
#using command set to set positional parameters
set  wm1.c  wm2.c
echo  $1  $2
#end
```

运行example5的结果如下：

```
[Stu@localhost pro]$bash  example5
wm1.c  wm2.c
```

5.4.4 位置参数

3．位置参数移动

当位置变量个数超出9个时，就不能直接引用位置大于9的位置变量了，必须用shift命令移动位置参数。。

shift命令的一般形式如下：

```
shift [n]
```

每次执行时，将位置参数向**左移动n位**。如果没有参数，则每次执行时，将位置参数向左移动1位。

【例5-6】使用shift命令，文件名为example6。

```
#! /bin/bash
#this file is example6
#example6:shell script to demonstrate the shift command
echo  $1 $2 $3 $4 $5 $6 $7 $8 $9
shift
echo  $1 $2 $3 $4 $5 $6 $7 $8 $9
shift 4
echo  $1 $2 $3 $4 $5 $6 $7 $8 $9
#end
```

运行example6脚本的结果如下：

```
[Stu@localhost pro]$bash  example6 A B C D E F G H I J K
A B C D E F G H I
B C D E F G H I J
F G H I J K
```

目录

本章要点

5.1 shell概述

5.2 创建和执行shell脚本

5.3 shell特殊字符

5.4 shell变量

5.5 正则表达式与算术运算

5.6 控制结构

5.7 其他语句

5.8 函数

5.9 调试shell脚本

5.10 实例

5.5.1 正则表达式

正则表达式是一种可以用于**模式匹配和替换的工具**，可以让用户通过使用一系列的特殊字符构建匹配模式，然后将匹配模式与待比较字符串或文件进行比较，根据比较对象中是否包含匹配模式，执行相应的程序。

※1．一般通配符

***（星号）**：可匹配任意字符的**0次或多次**出现。例如，f*可以匹配f、fa、fb、fa3、fff、ff.s等，即匹配以f打头的任意字符串。

但应注意，文件名前面的**圆点“.”**和路径中的**斜线“/”**必须显式匹配。例如，模式*file不能匹配.profile，而.*file才可匹配.profile。模式/etc*.c不能匹配在/etc目录下带有后缀.c的文件，而模式/etc/*.c会匹配这些文件。

5.5.1 正则表达式

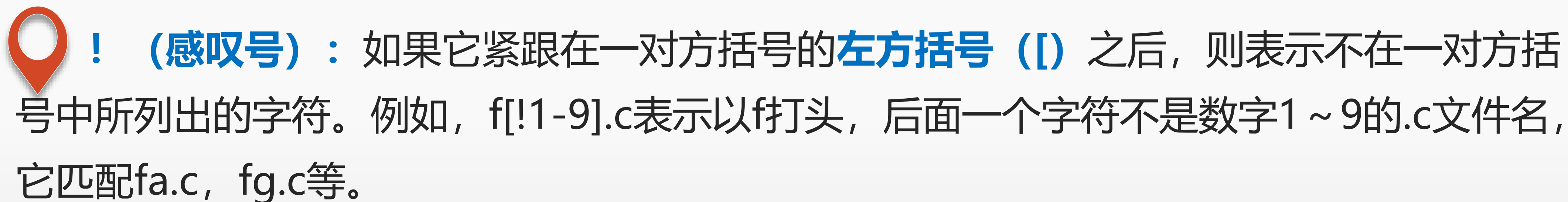
※1．一般通配符

?（问号）： 可匹配**任意一个字符**。例如，f?可以匹配f1、f4、fv等，但不能匹配f，facd，f34等。

[ ]（一对方括号）： 其中有一个**字符组**。其作用是匹配该字符组所限定的任何一个字符。例如，f[adefu]可以匹配fa，fd，fe，ff，fu，但不能匹配f1，fa3，fkk等。方括号中的字符组可以由直接给出的字符组成，如上面所示，也可以由表示限定范围的起始字符、终止字符及中间一个连字符（-）组成。例如，f[a-d]与f[abcd]的作用相同。又如，a[1-9]与a[123456789]的作用相同。前者的表示方式更简捷。

5.5.1 正则表达式

※1．一般通配符



!（感叹号）：如果它紧跟在一对方括号的**左方括号（[）**之后，则表示不在一对方括号中所列出的字符。例如，f[!1-9].c表示以f打头，后面一个字符不是数字1～9的.c文件名，它匹配fa.c，fg.c等。

在一个正则表达式中，可以**同时使用*（星号）和?（问号）**。例如，/usr/meng/t?/*匹配目录/usr/meng之下，子目录名是以t打头，后随一个任意字符的这些子目录下的所有文件名。

5.5.1 正则表达式

※1．一般通配符

又如，chapter[0-9]*表示chapter之后紧跟着1个0～9的数字，其后字符是**任意字符**，它可匹配chapter10，chapter0，chapter1，chapter29等。

^（幂次方号）： 只允许放在一行的开始匹配字符串。例如，^d表示所有以d开头的行。

**$（美元号）：** 只在行尾匹配字符串，它放在匹配单词的后面。例如，Linux$表示以单词Linux结尾的所有文件。

5.5.1 正则表达式

※2．模式表达式

模式表达式是那些包含一个或多个通配符的字符串。bash除支持Bourne shell中的*、?、[ ]、!、^及$通配符外，还提供了特有的扩展模式匹配表达式。

（1）*（模式表）：

匹配给定模式表中“模式”的0次或多次出现，各模式之间以竖线（|）分开。例如，file*(.c|.o)匹配文件file.c、file.o、file.c.o、file.c.c、file.o.c、file等，但不匹配file.h、file.s等。

（2）+（模式表）：

匹配给定模式表中“模式”的1次或多次出现，各模式之间以竖线（|）分开。例如，file+(.c|.o)匹配文件file.c、file.o、file.o.c、file.c.o等，但不匹配file。

5.5.1 正则表达式

※2．模式表达式

（3）？（模式表）：

匹配模式表中任何一种“模式”的0次或1次出现，各模式之间以竖线（|）分开。例如，file?(.c|.o)只匹配file、file.c、file.o等，不匹配file.c.c、file.c.o等。

（4）@（模式表）：

仅匹配模式表中给定“模式”的1次出现，各模式之间以竖线（|）分开。例如，file@(.c|.o)只匹配file.c和file.o，但不匹配file、file.c.c、file.c.o等。

（5）！（模式表）：

除给定模式表中的一个“模式”之外，它可以匹配其他任何的字符串。

在实际使用时，模式表达式可以递归，即每个表达式中都可以包含一个或多个模式。例如，file*(.[cho]|.sh)是合法的模式表达式。

5.5.2 算术运算

bash中执行整数算术运算的命令是let。

其语法格式为： **let 参数 ……**

其中，参数是单独的**算术表达式**。这里的算术表达式使用C语言中表达式的语法、优先级和结合性。除++、--和逗号之外，所有的整型运算符都得到支持。此外，还提供了方幂运算符**。

let命令的替代表示形式是： **（（算术表达式））**

例如，let"j=i*6+2"等价于（（j=i*6+2））

5.5.2 算术运算

表5-1列出了在算术表达式中可用的运算符及其优先级和结合性。

表5-1 bash中的算术运算符及其优先级和结合性

优先级	运算符	结合性	功能
1	- +	从右至左	取表达式的负值取表达式的正值
2	! ~	从右至左	逻辑非按位取反
3	**	从左至右	方幂
4	*/%	从左至右	乘除取模
5	+ -	从左至右	加减

5.5.2 算术运算

表5-1 bash中的算术运算符及其优先级和结合性

<table>
<tr><th>优先级</th><th>运算符</th><th>结合性</th><th>功能</th></tr>
<tr><td>6</td><td><< >></td><td>从左至右</td><td>左移若干二进制位右移若干二进制位</td></tr>
<tr><td>7</td><td>> >= < <=</td><td>从左至右</td><td>大于大于等于小于小于等于</td></tr>
<tr><td>8</td><td>= = !=</td><td>从左至右</td><td>等于不等于</td></tr>
<tr><td>9</td><td>&</td><td>从左至右</td><td>按位与</td></tr>
<tr><td>10</td><td>^</td><td>从左至右</td><td>按位异或</td></tr>
<tr><td>11</td><td>|</td><td>从左至右</td><td>按位或</td></tr>
<tr><td>12</td><td>&&</td><td>从左至右</td><td>逻辑与</td></tr>
<tr><td>13</td><td>||</td><td>从左至右</td><td>逻辑或</td></tr>
</table>

5.5.2 算术运算

表5-1 bash中的算术运算符及其优先级和结合性

优先级	运算符	结合性	功能
14	?:	**从右至左**	条件计算
15	= += -= *= /= %= &= ^= |= >>= <<=	**从右至左**	赋值 运算且赋值

表中运算符优先级是**由高到低排列**的，即1级高，15级低。同级运算符在同一个表达式中出现时，其执行顺序由结合性表示。

在**bash表达式**中可以使用括号来**改变运算符的操作顺序**，即在运算时先计算括号内的表达式。

当表达式中有shell的特殊字符时，必须用**双引号**将其括起来。例如，let"val=a|b"。如果不括起来，shell会将命令行let val=a|b中的“|”看成管道符，将其左右两边看成不同的命令，因而无法正确执行。

目录

本章要点

5.1 shell概述

5.2 创建和执行shell脚本

5.3 shell特殊字符

5.4 shell变量

5.5 正则表达式与算术运算

5.6 控制结构

5.7 其他语句

5.8 函数

5.9 调试shell脚本

5.10 实例

5.6.1 条件语句

shell具有一般高级语言程序设计所具有的控制结构和其他的复杂功能，如if语句、case语句、循环结构及函数等。

1．条件测试

在shell中，测试条件表达式中只能通过使用test命令来完成。

test命令的语法如下： **test 条件表达式** **或** **[条件表达式]**

test命令可以与多种系统运算符一起使用。这些运算符可以分为四类，即字符串比较、数字比较、文件操作符及逻辑操作符。

5.6.1 条件语句

1．条件测试

（1）字符串比较

s1=s2	如果s1等于s2，则测试条件为真。
s1!=s2	如果s1不等于s2，则测试条件为真。
-n s1	如果字符串s1的长度大于0，则测试条件为真。
-z s1	如果字符串s1的长度等于0，则测试条件为真。

5.6.1 条件语句

1．条件测试

（2）数字比较

n1 -eq n2	如果n1等于n2，则测试条件为真。
n1 -ne n2	如果n1不等于n2，则测试条件为真。
n1 -gt n2	如果n1大于n2，则测试条件为真。
n1 -ge n2	如果n1大于或等于n2，则测试条件为真。
n1 -lt n2	如果n1小于n2，则测试条件为真。
n1 -le n2	如果n1小于或者等于n2，则测试条件为真。

5.6.1 条件语句

1．条件测试

（3）文件操作符

-r 文件名 如果文件存在且是用户可读的，则测试条件为真。

-w 文件名 如果文件存在且是用户可写的，则测试条件为真。

-x 文件名 如果文件存在且是用户可执行的，则测试条件为真。

-d 文件名 如果文件存在且是目录文件，则测试条件为真。

-f 文件名 如果文件存在且是普通文件，则测试条件为真。

-b 文件名 如果文件存在且是块文件，则测试条件为真。

-c 文件名 如果文件存在且是字符文件，则测试条件为真。

-s 文件名 如果文件存在且长度大于0，则测试条件为真。

5.6.1 条件语句

1. 条件测试

（4）逻辑操作符

逻辑操作符用于根据逻辑规则比较表达式。下面这些字符表示NOT、AND和OR。

!逻辑表达式	对一个逻辑表达式求反。
逻辑表达式 -a逻辑表达式	两个逻辑表达式同时为真，则返回真；否则为假。
逻辑表达式 -o逻辑表达式	两个逻辑表达式同进为假时，则返回假；否则为真。

5.6.1 条件语句

1．条件测试

（5）特殊条件测试

除以上条件测试外，在if语句和循环语句中还常用下列三个特殊条件测试语句。

: 表示不做任何事情，其退出值为0。

true 表示总为真，其退出值总为0。

false 表示总为假，其退出值为255。

如前面所述，命令退出值若为0，则表示条件测试为真；若退出值不等于0，则为假。

5.6.1 条件语句

2．条件语句

（1）if语句

if语句通过判定条件表达式做出选择。

格式1

```
if   条件表达式
then
命令1
[else
命令2]
fi
```

格式1的执行过程是：先进行条件测试，如果测试结果为真，则执行then之后的命令；否则，执行else之后的命令2。

格式2

```
If   条件表达式1
then
     命令1
elif 条件表达式2
then
     命令2
……
else
     命令n
fi
```

格式2的执行过程是：先进行条件1测试，如果测试结果为真，则执行命令1；否则，进行条件2测试，如果测试结果为真，则执行命令2；…；如果条件测试都为假，则执行命令n。

5.6.1 条件语句

if条件语句是可以**嵌套的**，也就是说，一个if条件语句可以在其中包含另一个if条件语句。在if条件语句中，elif或者else部分并不是必需的。如果在if条件语句中所指定的表达式和随后可选的elif语句中的表达式都不为真时，则执行else部分。字符fi标志if条件语句的结束，在嵌套if条件语句的情况下，fi是很有用的。在这种情况下，应该让fi与if相匹配，从而确保所有的if语句都编写正确。

【例5-7】if语句应用，文件名为example7。

```
#!/bin/bash
#if statement aplication
if  [  $1 = "yes"  ]
then
   echo "value is yes"
elif  [  $1 = "no"  ]
then
   echo "value is no"
else
echo  "invalid value"
fi
#end
```

5.6.1 条件语句

运行这个脚本：

```
[Stu@localhost pro]$bash  example7  yes
value is yes
[Stu@localhost pro]$bash  example7  no
value is no
[Stu@localhost pro]$bash  example7
invalid value
```

通常，if的测试部分是由**test命令**实现的。其实，条件测试可以利用一般命令执行成功与否来做出判断。如果正常结束，则表示执行成功，其返回值为0，条件测试值为真；如果命令执行不成功，其返回值不等于0，条件测试就为假。

if语句更一般的格式是：

```
if        命令表1
then
命令表2
[else
命令表3]
fi
```

其中，各命令表可以**由一条或者多条命令**组成。如果命令表1是由多条命令组成的，那么测试条件是以其中后一条是否执行成功为准。

5.6.1 条件语句

【例5-8】命令在if语句中的应用，文件名为example8。

```
#!/bin/bash
#if user has logged in the system.
#then, copy a file to his or her file
#else, display an error information
echo  "Type in the user name. "
read  user
if   grep $user  /etc/passwd >tmp/null
    who |grep  $user
then
echo"$user has logged in the system."
cp./tmp/null tmp1
rm/tmp/null
else
    echo "$user has not logged in the system"
fi
#end
```

运行这个脚本：

```
[Stu@localhost pro]$bash  example8
Type in the user name.
Stu
Stu has logged in the system.
[Stu@localhost pro]$bash example8
Type in the user name.
dongdong
dongdong has not logged in the system
```

5.6.1 条件语句

2．条件语句

（2）case语句

case语句用来执行依赖于离散值或者与指定变量相匹配的一定数据范围的语句。在大多数情况下，如果存在很多条件，那么可以使用case语句来代替if语句。

case语句的格式如下：

```
case  字符串  in
模式字符串1）
        命令
        ……
        命令；；
模式字符串2）
        命令
        ……
        命令；；
    ……
模式字符串n）
        命令
        ……
        命令；；
      *）
        命令
        ……
        命令；；
esac
```

其执行过程是，用**字符串的值依次与各模式字符串进行比较**，如果发现同某一个模式字符串匹配，那么就执行该模式字符串之后的各个命令，直至遇到两个分号为止。如果没有任何模式字符串与该字符串的值相符合，则执行*)后面的命令。

5.6.1 条件语句

【例5-9】以月份数字作为参数，编写一个回显月份名的脚本，脚本文件名为example9。

```
#!/bin/bash
case  $1  in
1) echo "month is January";;
2) echo "month is February";;
3) echo "month is March";;
4) echo "month is April";;
5) echo "month is May";;
6) echo "month is June";;
7) echo "month is July";;
8) echo "month is August";;
9) echo "month is September";;
10) echo "month is October";;
11) echo "month is November";;
12) echo "month is December";;
*) echo "Invalid parameter.";;
esac
#end
```

运行这个脚本：

```
[Stu@localhost pro]$bash  example9  2
month is February
[Stu@localhost pro]$bash  example9  14
Invalid parameter.
```

5.6.1 条件语句

在使用case语句时应注意以下6个方面：

（1）每个模式字符串后面可以有一条或多条命令，其后一条命令必须以两个分号结束。

【例5-10】下面的脚本检查命令行的第一个参数是否为-b或-s。如果是-b，则计算由第二个参数指定的文件中以b开头的行数。如果是-s，则计算由第二个参数指定的文件中以s开头的行数。如果第一个参数不是-b或-s，则显示一条选择有错的信息。脚本文件名为example10。

```
#!/bin/bash
case  $1  in
-b) count='grep  ^b  $2 |wc  -l'
echo  "The number of lines in $2 that start with b is $count.";;
  -s) count='grep  ^s  $2|wc  -l'
echo  "The number of lines in $2 that start with s is $count.";;
*) echo  "That option is not recognized.";;
esac
#end
```

● 5.6.1 条件语句

（2）模式字符串中可以使用通配符。

【例5-11】在模式字符串中通配符的应用，脚本文件名为example11。

```
#!/bin/bash
case  $1  in
-u) echo  "Searching  /usr/'logname'  for:$2"
find/usr/'logname' -name  $2  -print;;
-[cs] echo  "statement  for  command:$2"
          find /bin  /usr/bin  /etc  -name  $2  -print;;
   *)  echo  "invalid  first  argument. ";;
esac
#end
```

5.6.1 条件语句

（3）如果一个模式字符串中包含多个模式，那么各模式之间应以竖线（|）隔开，表示各模式是“或”的关系，即只要给定字符串与其中一个模式相配，就会执行其后的命令表。

【例5-12】在模式字符串中“或”关系的应用。

```
#!  /bin/bash
case  $choice  in
 time|date)          echo "The time is 'date'.";;
  dir|path) echo "Current directory is 'pwd'.";;
*)   echo "Bad argument.";;
esac
#end
```

5.6.1 条件语句

（4）各模式字符串应是**唯一的**，不应重复出现。并且要合理安排它们的**出现顺序**。例如，不应将*作为头一个模式字符串。因为*可以与任何字符串匹配。它若第一个出现，就不会再检查其他模式了。

（5）case语句以关键字**case开头**，以关键字esac结束。

（6）case的退出值是整个结构中最后执行的那个命令的退出值。若没有执行任何命令，则退出值为0。

5.6.2 循环语句

shell中有3种用于循环的语句，即while语句、for语句及until语句。

1．while语句

while语句的一般形式如下：

```
while测试条件
do
        命令表
done
```

其执行过程是，先进行条件测试，如果结果为真，则进入循环体（do和done之间的部分），执行其中的命令；然后再做条件测试，……，直至测试条件为假时才终止while语句的执行。

5.6.2 循环语句

【例5-13】while语句的应用。

```
#!/bin/bash
while [ $1 ]
do
if   [ -f  $1  ]
then  echo "display  :$1"
cat  $1
else  echo  "$1 is not a file name."
fi
shift
done
```

程序的作用是，首先判断位置参数1是否为普通文件，若是，则显示其内容；否则，显示它不是文件名的信息。每次循环处理一个位置参数$1，利用shift命令可将后续位置参数左移。

【例5-14】使用命令做测试条件的应用。

```
#!/bin/bash
echo  "key in file name →\c"
read  filename
echo  "key in data"
while    read  x
do      echo  $x >>$filename
done
cat $filename
#end
```

程序的作用是，执行while语句时，每读入用户输入的一个数据，就将它添加到指定的文件中，直至用户按**【Enter】键**为止。最后利用**cat命令**将指定文件列出来。

5.6.2 循环语句

2．for语句

for语句是常用的建立循环结构的语句。其使用格式主要有3种，取决于循环变量的取值方式。

格式一

```
for 变量 in 值表
do
    命令表
done
```

其执行过程是，变量依次**取值表中的各个值**，即第一次取值表中的第一个值，然后进入循环体，执行其中的命令；第二次取值表中的第二个值，然后进入循环体，执行其中的命令；依次处理，直到变量将值表中的各个值都取一次之后，从而结束for循环。

5.6.2 循环语句

【例5-15】for语句格式一的应用举例，文件名为example15

```
#!/bin/bash
for day in Monday Wednesday Friday Sunday
do
echo $day
done
#end
[Stu@localhost pro]$bash example15
Monday
Wednesday
Friday
Sunday
```

5.6.2 循环语句

格式二

```
for 变量 in 文件正则表达式
do
    命令表
done
```

前目录下（或给定目录下）与**正则表达式相匹配**的文件名，每取一次，就进入循环命令表，直到所有匹配的文件名取完为止，退出循环。

【例5-16】for语句格式二的应用举例。

```
#!/bin/bash
for file in m*.c
do
cat $file|pr
done
#end
```

这个脚本的作用是将当前目录下所有以m打头的C程序文件都按分页格式显示出来。

5.6.2 循环语句

格式三

```
for i in $*
do
    命令表
done
```

或者

```
for i
do
    命令表
done
```

这种格式有两种形式。其执行过程是，变量i依次取位置参数的值，然后执行循环体中的命令表，直至所有位置参数被取完为止。

5.6.2 循环语句

【例5-17】for语句格式三的应用举例。

```
#/bin/bash
#display files under a given directory
#$1-the name of the directory
#$2-the name of files
dir=$1;shift
if  [ -d  $dir ]
then
   cd  $dir
   for  filename
   do
```

接左边

```
if   [ -f $filename ]
then
cat  $filename
          echo  "End of ${dir}/$filename"
      else
 echo  "Invalid file name:${dir}/$filename"
fi
done
else
    echo  "Bad directory name:$dir."
fi
#end
```

执行这个脚本时，如果第一个位置参数是合法的目录，那么就将后面给出的各个位置参数所对应的文件显示出来，若给出的文件名不正确，则显示出错信息。如果第一个位置参数不是合法的目录，则显示目录名不对。

5.6.2 循环语句

3．until语句

until语句可以用来执行一系列命令，直到所指定的条件为真时才终止循环。

until语句的一般格式为：

```
until
      测试条件
do
      命令表
done
```

可以看出，它与while语句很相似，只是测试条件不同，即当测试条件为假时，才进入循环体，直至测试条件为真时终止循环。

【例5-18】求前5个偶数之和，文件名为example18。

```
#!  /bin/bash
loopcount=0
result=0
until  [ $loopcount –ge 5 ]
do
   let  loopcount=$loopcount+1
   let  increment=$loopcount\*2
   let  result=$result+$increment
done
   echo  "result is $result"
#end
[Stu@localhost pro]$bash example18 Result is 30
```

目录

本章要点

5.1 shell概述

5.2 创建和执行shell脚本

5.3 shell特殊字符

5.4 shell变量

5.5 正则表达式与算术运算

5.6 控制结构

5.7 其他语句

5.8 函数

5.9 调试shell脚本

5.10 实例

5.7.1 break语句

break语句可以用来终止**一个重复执行的循环**。这种循环可以是for、until或者while语句构成的循环。

其语法格式为： break [n]

其中，n表示要跳出的几层循环。默认值是1，表示只跳出一层循环。如果n为3，则表示一次跳出3层循环。

【例5-19】按反向打印出命令行中给出的参数。

```
#!/bin/bash
count=$#
cmd=echo
while  true
do
cmd="$cmd \$$count"
let  count=$count –1
if   [ $count  -eq  0 ]
then
    break
fi
done
eval $cmd
#end
```

在该脚本中，while的测试条件总为真，它的唯一出口点是执行break语句。

5.7.2 continue语句

continue语句跳过循环体中在它之后的语句，回到本层循环的开头，进行下一次循环。

其语法格式为： continue [n]

其中，n表示从包含continue语句的内层循环体向外跳到第n层循环。默认值为1。循环层数是由内向外的编号。

【例5-20】continue语句的应用。

```
#!/bin/bash
for i in 1 2 3 4 5 6
do
if [ "$i" -eg 3 ]
then
    continue
else
    echo "$i"
fi
done
#end
```

5.7.3 exit语句

exit语句可以用来退出一个shell程序，并设置退出值。

其语法格式为：

```
exit [n]
```

其中，n是设定的**退出值**。如果未显示给出的n值，则退出值为后一个命令的执行状态。

目录

本章要点

5.1 shell概述

5.2 创建和执行shell脚本

5.3 shell特殊字符

5.4 shell变量

5.5 正则表达式与算术运算

5.6 控制结构

5.7 其他语句

5.8 函数

5.9 调试shell脚本

5.10 实例

与其他的编程语言一样，shell程序也**支持函数**。函数是shell程序中执行特殊过程的部件，并且在shell程序中可以被**重复调用**。编写函数将有助于编写没有重复代码的shell程序。

函数定义的格式为：

```
[function]函数名( )
{
   命令表
}
```

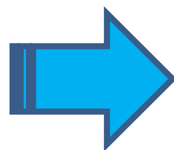
函数的调用形式为：

```
函数名 参数1  参数2  参数3
```

参数是可选的。

【例5-21】函数应用举例。displaymonth函数的作用是：在用户传送一个月份数字之后显示月份名或者一条错误消息。

```
#!/bin/bash
displaymonth(){
case $1 in
1)echo "month is January";;
2)echo "month is February";;
3)echo "month is March";;
4)echo "month is April";;
5)echo "month is May";;
6)echo "month is June";;
7)echo "month is July";;
8)echo "month is August";;
9)echo "month is September";;
10)echo "month is October";;
```



```
11)echo "month is November";;
12)echo "month is December";;
*) echo "Invalid parameter";;
esac
}
while  true
do
   echo  "input month's number; "
   read  mm
   displaymonth   $mm
   echo  "continue(y/n)? "
   read  answer
   if  [  ${answer}='n'  ]
   then
       break
   fi
done
#end
```

函数应**先定义，后使用**。调用函数时，直接用函数名，如displaymonth，不必带圆括号，就像一般命令那样使用。shell脚本与函数间的参数传递可利用位置参数和变量直接传递。

目录

本章要点

5.1 shell概述

5.2 创建和执行shell脚本

5.3 shell特殊字符

5.4 shell变量

5.5 正则表达式与算术运算

5.6 控制结构

5.7 其他语句

5.8 函数

5.9 调试shell脚本

5.10 实例

在编写shell脚本的过程中难免会出现错误，通过对shell脚本的调试，可以寻找和消除这些错误。在bash中，shell脚本的调试主要是利用bash命令解释程序的选项来实现。

其格式为：

```
bash  [选项] 脚本文件名
```

其中，主要利用bash命令解释程序的-v或-x选项来跟踪程序的执行。例如：

```
$/bin/bash  -x  脚本文件名
```

或

```
$/bin/bash  -v  脚本文件名
```

通常，**-v选项**允许用户观察一个shell程序的**读入和执行**。如果在读入命令行时发生错误，则**终止程序的执行**。每个命令行被读入后，shell按读入时的形式显示出该命令行，然后执行命令行。而**-x选项**也允许用户观察一个shell程序的执行，而它是在命令行执行前完成所有的替换之后，才显示出每一个被替换后的命令行，并且在行前加前缀符号“+”（变量赋值语句不加“+”符号），然后执行命令。两者的主要区别在于：**使用-v选项，则打印出命令行的原始内容；而使用-x选项，则打印出经过替换后的命令行的内容。**

上面的两个选项也可以在shell脚本的内部用**“set -选项”**的形式引用，而用**“set +选项”**禁止该选项起作用。如果只想对程序的某一部分进行调试时，则也可以将该部分用上面的两个语句包围起来。

目录

本章要点

- 5.1 shell概述
- 5.2 创建和执行shell脚本
- 5.3 shell特殊字符
- 5.4 shell变量
- 5.5 正则表达式与算术运算
- 5.6 控制结构
- 5.7 其他语句
- 5.8 函数
- 5.9 调试shell脚本
- **5.10 实例**

◆1．建立bash脚本

在编辑器，如vi中，编辑如下程序：

```
1   #!/bin/bash
2   # this script searches a file for a specified word
3   # you need to enter two arguments to this script："the filename" and "the word"
4   if  test  $ #  -ne 2
```

脚本第1行指定bash（绝对路径为/bin/bash）作为该脚本的**解释器**。第2、3行是注释，说明了**bash脚本的用途及如何调用这个脚本**（调用时需输入文件名及指定的单词两个参数）。

第4行用test命令来检验用户**是否传递了两个参数**。如果不是两个参数，则执行第6行的命令，输出Invalid argument！信息；如果是两个参数，则程序将继续执行。

◆1．建立bash脚本

在编辑器，如vi中，编辑如下程序：

```
5   then
6   echo  " Invalid argument! "
7   else
8   filename = $1
9   word = $2
```

第8、9行将第1个和第2个参数的值通过位置变量$1、$2分别赋给filename和word变量。

◆1．建立bash脚本

在编辑器，如vi中，编辑如下程序：

```
10  if  grep  $word  $filename >/home/Stu/temp
11  then
12  echo  "The word was found ! "
13  else
14  echo  "The word was not found ! "
15  fi
16  fi
17  exit 0
```

第10行利用grep在指定的文件中搜索单词，同时将输出重定向到/home/Stu/temp中。如果grep命令找到了这个单词，那么就执行第12行的命令，显示“The word was found！”的信息；否则，就执行第14行的命令，显示“The word was not found！”信息后，执行exit命令，退出。

◆2．调试bash脚本

调试该脚本时，还应给出两个参数，即被搜索的文件名和要查找的指定的单词。

通过命令

$/bin/bash -x /home/Stu/search 文件名指定的单词

或

$/bin/bash -v /home/Stu/search 文件名指定的单词

跟踪脚本程序的**执行过程**，对脚本进行**调试**。如有错误，则可以进一步修改，直到调试后，无错误出现，即可结束调试。使用这种命令形式，可以同时完成脚本的**调试和执行**。

◆3．执行bash脚本

方法一

```
$chmod  u+x  /home/Stu/search
$/home/Stu/search  文件名 指定的单词
```

首先，设置脚本的**访问权限为可执行**，然后执行该脚本（需传递两个参数，即文件名和指定的单词）。

```
$. search   文件名 指定的单词
```

方法二

```
$bash   /home/Stu/search  文件名 指定的单词
```

直接由bash执行该脚本。

1.什么是shell？Red Hat Linux系统默认的是哪一种shell?

2．shell的主要功能是什么？bash有什么特点?

3．执行shell脚本的方式主要是什么?

4．什么是重定向？什么是管道?

5．说明三种引号的作用，以及有什么区别?

6．shell有哪几种类型的变量？如何实现对变量的赋值和引用?

7．试说明下列命令的执行结果

（1）ls [a-h]?.c

（2）sort <text1&>>test2

（3）ls |wc -l

（4）假设当前目录为 /home，则以下程序结果。

```
string1="$PWD";
string2="\$PWD"
echo "$string1 and $string2"
```

8．分析下面shell脚本的功能

```
count=$# cmd=echo
while [ $count -gt 0 ]
do
   cmd="$cmd \$$count"
   let count=$count-1
done
```

9．编写一个shell脚本，它把第二个位置参数及其以后的各个参 数指定的文件拷贝到第一个位置参数指定的目录中。

10．编写一个shell脚本，显示当天日期，查找给定的某用户是否在系统中工作。如果在系统中工作，就发一个问候给用户。

11．打印给定目录下的某些文件，由第一个参数指出文件所在的目录，其余参数是要打印的文件名。

12．利用for循环将当前目录下的.c文件移到指定的目录下，并按文件大小排序，显示移动后指定目录的内容。

13．编写一个脚本，求斐波那契数列的前10项及其总和。

14．编写一个脚本，求前10个自然数之和

谢谢观看