第 5 章　Linux 的 vim 与 Bash

学习目标

(1) 掌握 vim 编辑器的功能。
(2) 了解 Bash Shell 的基本功能。
(3) 理解 Bash 环境变量。

vim 和 Bash 是进行 Shell Script 编写的基础，也是 Linux 系统编程比较重要的先导内容。本章首先讲解 Linux 下 vim 编辑器的功能和使用，然后介绍 Shell 与 Bash 之间的关系，以及 Shell 的相关内容和命令，接着详细讲解 Bash 的基本功能和环境变量的使用，从而为 Shell 编程打下坚实的基础。

5.1　vim 编辑器

每个系统管理员都应该至少要学会一种文字接口的编辑处理器，以方便系统日常的管理行为。在 Linux 上的文字处理软件非常多，vi 与 vim 编辑器是所有 UNIX 及 Linux 系统下标准的编辑器，相当于 Windows 系统中的记事本，它的强大不逊色于任何最新的文本编辑器，是人们使用 Linux 系统不能缺少的工具。对 UNIX 及 Linux 系统的任何版本，vi 编辑器都是完全相同的。所以 vi 编辑器是未来人们进行 Shell Script 程序的编写与服务器相关配置文件的首选编辑工具，学会使用它后，将在 Linux 的世界里畅行无阻。

5.1.1　vi、vim 与 gvim

诞生于 20 世纪 70 年代的 vi 编辑器是所有的类 UNIX 系统默认的文本编辑器，所以所有的类 UNIX 系统都会内置 vi 编辑器，其他的编辑器则不一定会存在。但是目前人们使用比较多的是 vim 编辑器。vim 具有程序编辑的能力，可以主动地以字体颜色辨别语法的正确性，方便程序设计。

vim 是从 vi 发展出来的一个文本编辑器，不过在 vi 的基础上增加了很多新的特性，vim 普遍被推崇为类 vi 编辑器中最好的一个，其代码补完、编译及错误跳转等方便编程的功能特别丰富，在程序员中被广泛使用，和 Emacs 并列成为类 UNIX 系统用户最喜欢的编辑器。vim 的第一个版本由 Bram Moolenaar 在 1991 年发布，最初的简称是 Vi IMitation，随着功能的不断增加，正式名称改成了 Vi IMproved。

gvim 的 g 指的是 GUI,也就是图形化界面,相当于在 vim 包了一层图形化界面。所以可以说 gvim 是 vim 的图形前端,一般运行在桌面环境中,而 vim 一般运行在命令行下。相比之下,gvim 拥有更丰富的颜色和字体,还有菜单和滚动条,以及更友好的鼠标操作等,除此之外与 vim 差异不大。gvim 是跨平台的编辑器,主流的 Linux 操作系统上面都有它的版本,并会根据安装的平台自动选择相应语言包,支持中文及其各种编码,这个极具 UNIX 特色和风格(simple is the best)的编辑器相信会带来不同的感受。

5.1.2 vim 和 gvim 的安装

目前 Fedora 29 默认只安装了 vi 命令,可以直接运行 vim 由系统自动帮忙完成这个命令的安装,当然也可以通过 **#dnf -y install vim-enhanced** 完成 vim 命令所属 rpm 包的安装。

```
[root@localhost ~]#vim
bash: vim: 未找到命令…
安装软件包 vim-enhanced 以提供命令 vim? [N/y]y

 * 正在队列中等待…
 * 装入软件包列表…
下列软件包必须安装:
gpm-libs-1.20.7-16.fc29.x86_64    Dynamic library for for the gpm
vim-common-2:8.1.1991-2.fc29.x86_64    The common files needed by any version
of the VIM editor
vim-enhanced-2:8.1.1991-2.fc29.x86_64    A version of the VIM editor which
includes recent enhancements
vim-filesystem-2:8.1.1991-2.fc29.noarch    VIM filesystem layout
继续更改? [N/y] y
…
 * 正在安装软件包…
```

而 gvim 可以采用 dnf 命令(参见 8.3 节)完成安装,安装完成后,输入命令 gvim,运行界面如图 5.1 所示,可以很方便地使用鼠标进行界面操作和内容编辑。

```
[root@localhost ~]#gvim
bash: gvim: 未找到命令…
相似命令是: 'vim'
[root@localhost ~]#dnf  -y install  vim-X11
…
vim-X11      x86_64       2:8.1.1991-2.fc29       updates       1.5 M

…
安装  1 软件包
```

```
总下载:1.5 M
…
已安装:
  vim-X11-2:8.1.1991-2.fc29.x86_64

完毕!
```

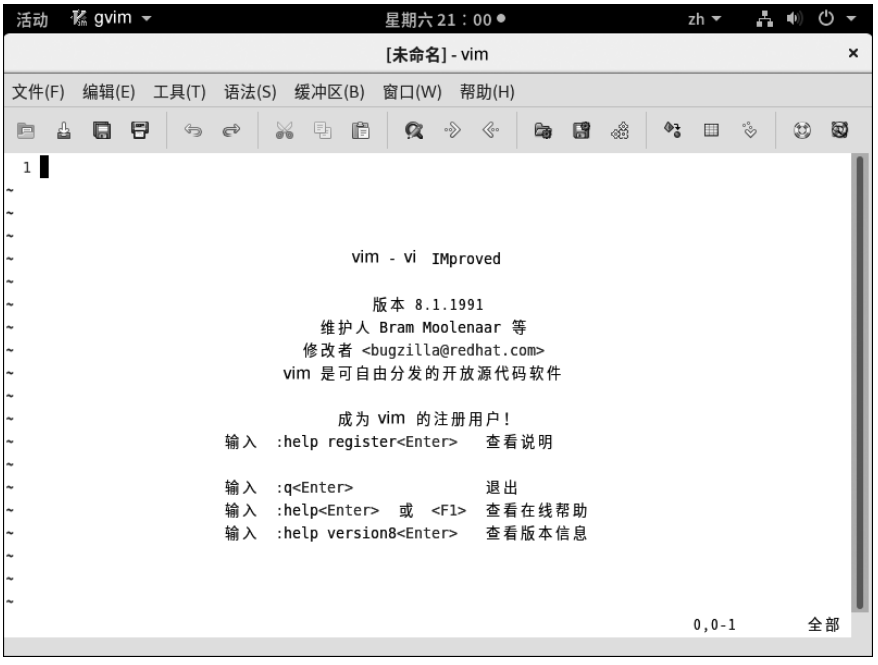

图 5.1 gvim 界面

5.1.3 vi/vim 的使用

如图 5.2 所示,基本上 vim 可以分为三种模式状态,分别是命令模式、输入模式和底线命令模式,各模式的功能区分如下。

操作演示
vim 的使用

1. 命令模式

以 vim 打开一个文件就直接进入命令模式(这是默认的模式)。在这个模式中,可以使用上、下、左、右按键来移动光标,可以使用删除字符或删除整行来处理文件内容,也可以使用复制、粘贴来处理文件的数据。

2. 输入模式

在命令模式中可以进行删除、复制、粘贴等操作,但是却无法编辑文件的内容,当按下

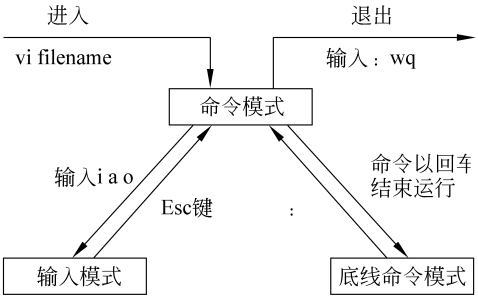

图 5.2　vi/vim 工作模式

i、I、o、O、a、A、r、R 等任何一个字母之后进入输入模式。这时候屏幕的左下方会出现 INSERT 或 REPLACE 的字样，此时才可以进行编辑。而如果要回到命令模式时，则必须要按下 Esc 键即可退出输入模式。

3. 底线命令模式

输入“：、/、?”三个中的任何一个，都可以将光标移动到最底下那一行。在这个模式中，可以提供查找、读取、存盘、替换字符、离开 vi、显示行号等功能。

使用 vim 建立一个文件的一般步骤如下。

(1) 输入“vim 文件名”，进入 vim 的命令模式，可以在左下角观察到这个文件目前的状态，如下所示，watch.txt 此时是一个新文件。

```
    [root@localhost ~]#vim watch.txt

~
~
~
"watch.txt" [新文件]                                    0,0-1         全部
```

(2) 按下 i(o 或 a)键之后，则进入输入模式，左下角出现“插入”，则可以开始编辑文字，可以输入任意字符。

(3) 按下 Esc 键可再次回到命令模式，此时“插入”已不再出现，内容编辑完毕。在命令模式下，vim 提供了文本整行的删除(dd)、复制(yy)与粘贴(p)，用法相同。以 dd 为例，可以输入 dd：连续按 d 键两次，删除当前行；也可以输入 dnd：如连续按 d、3、d，删除包括当前行在内的往下三行。

(4) 然后按下“：”键进入底线命令模式，可以输入 wq 保存退出。vim 退出编辑器的相关方式如下。

：w 表示将缓冲区写入文件，即保存修改，但不退出。

：wq 表示保存修改并退出。

：x 表示保存修改并退出。

：q 表示退出，如果对缓冲区进行过修改，则会提示。

:q! 表示强制退出,放弃修改。

```
    hello world!
I will study Linux
~
~
~
:wq
```

vim 还有很多实用的功能,大家可以在使用过程中查阅 vim 的官方网站(http://www.vim.org)加强对 vim 的深入了解。例如要想在 vim 中显示文件内容的行号有两种方式。

(1) 用 vim 打开文件时,按 Esc 键进入命令模式,输入 set nu,即可显示行号。

(2) 直接修改配置文件,#vim ~/.vimrc(注:.vimrc 文件原先可能没有,在此可以创建),在该文件中添加一行 set nu,然后保存,退出。

5.2 Shell 与 Bash

Shell 是 Linux 系统的用户界面,提供了用户与内核进行交互操作的一种接口。它接收用户输入的命令并把它送入内核去执行。Shell 的英文意思是外壳,在 Linux 系统,Shell 实际上也是一个程序,它是用户和操作系统间的命令解释器,负责接收用户输入的命令并将它翻译成操作系统能够理解的指令。如果把 Linux 内核想象成一个球体的中心,Shell 就是围绕内核的外层,从 Shell 向 Linux 操作系统传递命令时,内核就会做出相应的反应,如图 5.3 所示。例如,用户输入 **ls -l**,Shell 首先翻译这条命令;然后判断该命令是内部命令还是一个应用程序,Shell 的内部命令或应用程序将被分解为系统调用并传给 Linux 内核执行翻译后的指令;最后 Linux 内核将指令的执行结果返回给 Shell。

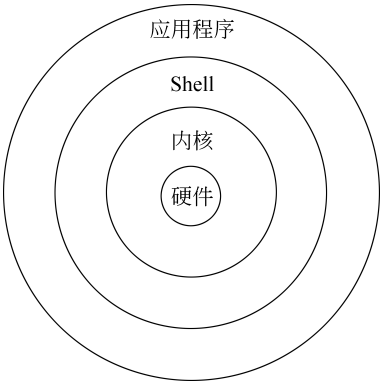

图 5.3 Linux 结构示意图

因为 Linux 就是以 Bash 为预设 Shell 的,在 Linux 下如果不懂 Bash,其他的内容就没有意义了,因为许多命令的执行是通过 Bash 的环境来处理的。

Bash(GNU Bourne-Again Shell)是许多 Linux 平台的内定 Shell,事实上,还有许多传统 UNIX 上用的 Shell,如 tcsh、csh、ash、bsh、ksh 等,Shell Script 大致都类同,当学会一种 Shell 以后,其他的 Shell 会很快就上手。多数情况下一个 Shell Script 通常可以在很多种 Shell 上使用。Bash 是大多数 Linux 系统以及 Mac OSX v10.4 默认的 Shell,它能运行于大多数 UNIX 风格的操作系统之上,甚至被移植到了 Microsoft Windows 上的 Cygwin 系统中,以实现 Windows 的 POSIX 虚拟接口。

5.3 Bash 的基本功能

Bash 是当前 Linux 版本的标准 Shell。Bash 与 Bourne Shell 完全向后兼容，并且在 Bourne Shell 的基础上增加和增强了很多特性。Bash 也包含了很多 csh 和 ksh 里的优点。Bash 有很灵活和强大的编程接口，同时又有很友好的用户界面。Bash 的主要优点是它有命令记忆功能，即历史命令 history，记录在/home/用户名/.bash_history 中，它还有命令补全功能，能够设置别名，能够随时结束终端进程，可以编写脚本程序，还有通配符能够帮助用户查询和命令执行。

5.3.1 解析命令行

当用户打开一个（虚拟）终端时，可以看到一个 **Shell** 提示符，标识了命令行的开始。用户可以在提示符后面输入任何命令及其选项与参数。请注意在命令行中选项先于参数输入。


```
#command  [选项][参数]
```

```
[root@localhost ~]#ls -l /root
总用量 116
…
```

在一个命令行中可以输入多个命令，用分号将多个命令隔开。

```
[root@localhost ~]#touch mydata
[root@localhost ~]#cp  /proc/cpuinfo  cpuinfo ;  ls  -a
…
```

如果一个命令太长，无法在一行中显示，可以使用反斜线来续行，在多个命令行上输入一个命令或多个命令。

```
[root@localhost ~]#cp  /proc/cpuinfo  cpuinfo ;\
>ls -a
…
```

查询某命令是否为 Bash 内部命令：#type -t 命令名，其中 file 为外部命令；alias 为命令别名；builtin 为内部命令。

```
[root@localhost ~]#type -t ll
alias
[root@localhost ~]#type -t cd
```

```
builtin
[root@localhost ~]#type -t mkdir
file
```

5.3.2 通配符

通配符是一种特殊符号，可以用来在引用文件名时简化命令的书写。在 Bash 中可以使用三种通配符：*、?、[]，用来模糊搜索文件。通配符的含义如表 5.1 所示。

表 5.1 通配符的含义

符号	含 义
*	任意的字符串(包括零个字符)
?	匹配任何单个字符
[ ]	创建一个字符列表，方括号中的字符用来匹配或不匹配单个字符。如 [xyz]匹配 x、y 或 z，但不能匹配 xx、xy 或者其他任意组合。 无论列表中有多少个字符，它只匹配一个字符。 [abcde]可以简写为 [a-e]。 另外，用感叹号作为列表的第一个字符可以起到反意作用，如 [!xyz]表示匹配 x、y、z 以外的任意一个字符

通配符的举例如下。

通配符 * 的常用方法就是查找具有相同扩展名的文件，例如，ls *.tgz 就是显示当前目录下后缀名为 tgz 的所有文件。

```
[root@localhost ~]#touch sun.tgz
[root@localhost ~]#touch sky.tgz
[root@localhost ~]#ls *.tgz
sky.tgz  sun.tgz
```

通配符 * 有时可以将几百条命令缩短成一个命令。例如，rm -f *.jpg 就是删除当前目录下后缀名为 jpg 的所有文件。

问号通配符? 必须匹配一个且只能匹配一个字符，通常用来查找比 * 更为精确的匹配。

```
[root@localhost ~]#ls *.???
anaconda-ks.cfg  hjk.dat  install.log  test.txt  tst.txt  watch.txt
//显示文件后缀名只有三个字符的所有文件
```

方括号通配符使用括号内的字符作为被匹配的字符，且只能匹配其中的一个字符。如列出以 a、b、c 开头，且以 dat 为扩展名的所有文件：ls [abc]*.dat。也可以在方括号

中使用连字符-来指定一个范围，如列出以字母开头、数字结尾的所有文件：#**ls [a-z A-Z]*[0-9]**。

5.3.3 命令别名

别名是 Bash 中用来节省时间的另一项重要功能，直接运行 alias 命令显示所有当前的别名，会发现其实 rm 别名也是 rm -i。要特别注意的是，别名既可以生造一个单词，也可以覆盖一个已有命令。

```
[root@localhost ~]#alias
alias cp='cp -i'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --
show-tilde'
```

Bash 允许用户按照自己喜欢的方式通过 alias 对相关的命令进行自定义创建；也可以通过 unalias 命令取消已创建的别名。

```
[root@localhost ~]#alias lm='ls'
[root@localhost ~]#lm
anaconda-ks.cfg  install.log.syslog  newdata     test.txt   公共的图片音乐
hjk.dat          list                sky.tar.qz  tst.txt    模板文件桌面
install.log      mydata              sun.tar.qz  watch.txt  视频下载
[root@localhost ~]#ls
anaconda-ks.cfg  install.log.syslog  newdata     test.txt   公共的图片音乐
hjk.dat          list                sky.tar.qz  tst.txt    模板文件桌面
install.log      mydata              sun.tar.qz  watch.txt  视频下载

[root@localhost ~]#unalias lm
[root@localhost ~]#lm
命令未找到。
```

5.3.4 命令行自动补齐

通常用户在 Bash 下输入命令时不必把命令输全，Shell 就能判断出所需要的命令。

该功能的核心思想：Bash 根据用户已输入的信息来查找以这些信息开头的命令，从而试图完成当前命令的输入工作。

用来执行这项功能的键是 Tab 键，按下 Tab 键后，Bash 就试图完成整个命令的输入，并将列出所有能够与当前输入字符相匹配的命令列表。例如在命令行输入 his＜Tab＞，Bash 就会自动将命令补全为查看用户历史命令为 history。这项功能同样适用于文件名的自动补齐，例如要进入目录：/etc/sysconfig/network-scripts/，不需要一个一个字符的输入，既浪费时间，又可能输错，可以直接输入如下命令。

```
[root@localhost ~]#cd  /e<Tab>sys<Tab>c<Tab>ne<Tab>-<Tab>
[root@localhost network-scripts]#pwd
/etc/sysconfig/network-scripts/
```

5.3.5 管道与 awk、cut 命令

管道是 Linux 从 UNIX 继承过来的进程间的通信机制，是把一个程序的输出直接连接到另一个程序的输入。管道是 UNIX 早期的一个重要通信方法，其思想是在内存中创建一个共享文件，从而使通信双方利用这个共享文件来传递信息。由于这种方式具有单向传递数据的特点，所以这个作为传递消息的共享文件就叫作“管道”。

我们在第三章介绍 head、tail 命令以及 grep 命令时已经使用了管道，下面先回顾一下这几个命令与管道的结合使用。

```
[root@localhost ~]#head  -n  20  /etc/passwd  |  tail  -n  10
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
gopher:x:13:30:gopher:/var/gopher:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
avahi-autoipd:x:170:170:Avahi IPv4LL Stack:/var/lib/avahi-autoipd:/
sbin/nologin
usbmuxd:x:113:113:usbmuxd user:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
rpc:x:32:32:Rpcbind Daemon:/var/lib/rpcbind:/sbin/nologin
rtkit:x:172:172:RealtimeKit:/proc:/sbin/nologin
/*该命令用到了管道文件，即先获取 passwd 文件的前 20 行内容，但是不直接打印而是把这个结果作为管道后面命令 tail 的输入，从而使得 tail 命令在这 20 行内容的基础上截取获得后 10 行*/
```

```
[root@localhost ~]#cat  /proc/meminfo  |  grep  Total
MemTotal:        2016276 kB
SwapTotal:       2097148 kB
VmallocTotal:   34359738367 kB
CmaTotal:              0 kB
HugePages_Total:       0

[root@localhost ~]#cat  /proc/meminfo  |  grep  Mem
MemTotal:        2016276 kB
MemFree:          381308 kB
MemAvailable:     922540 kB

[root@localhost ~]#cat  /proc/meminfo  |  grep  Total  |  grep Mem
MemTotal:        2016276 kB
//连续用到了两个管道,先过滤出含有 Total 内容的 5 行,在此基础上再次过滤含有 Mem 内容的 3 行
```

下面结合经常与管道一起使用的 awk 命令和 cut 命令进一步深入理解管道。

1. awk 命令

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

awk 有 3 个不同版本:awk、nawk 和 gawk,如果没有特别说明,一般指 gawk,gawk 是 awk 的 GNU 版本。awk 其名称得自于它的创始人 Alfred Aho、Peter Weinberger 和 Brian Kernighan 姓氏的首个字母。实际上 awk 的确拥有自己的语言:awk 程序设计语言,三位创建者已将它正式定义为“样式扫描和处理语言”。它允许创建简短的程序,这些程序读取输入文件、为数据排序、处理数据、对输入执行计算以及生成报表,还有无数其他的功能。

awk 命令的基本语法如下。

```
awk  [选项][脚本]文件名
```

在默认情况下,awk 会将如下变量分配给它在文本行中发现的数据字段。

$0:代表整个文本行。

$1:代表文本行中的第 1 个数据字段。

$2:代表文本行中的第 2 个数据字段。

$n:代表文本行中的第 n 个数据字段。

前面说过,在 awk 中,默认的字段分隔符是任意的空白字符(例如空格或制表符)。在文本行中,每个数据字段都是通过字段分隔符划分的。awk 在读取一行文本时,会用预定义的字段分隔符划分每个数据字段。