# 第 3 章

# vi 文本编辑器

## 3.1 vi 文本编辑器概述

与 UNIX 相同，Linux 本质上是一个文本驱动(text-driven)的操作系统。文本文件就是全部由 ASCII 码字符及某种语言的编码字符构成的文件，不含有任何样式和格式信息。文本文件可以被任何文本编辑器解释，也可以被所有程序操作和使用。在 Linux 系统中，文本文件被广泛地用作系统配置文件和系统工具软件的操作对象。这使得用户可以在文本方式下完成几乎所有的工作，如编写程序、读写邮件、配置和管理系统等。而完成所有这些工作的基本工具就是文本编辑器。因此，Linux 的用户应当熟悉至少一种文本编辑器。

### 3.1.1 vi 文本编辑器介绍

Linux 下的文本编辑器有很多种，其中 vi 是最基本的文本编辑工具。vi(visual)诞生于 1978 年，由柏克莱大学的 Bill Joy 编写。从其诞生至今，vi 始终是所有 UNIX/Linux 系统上必配的编辑器。目前 Linux 系统上流行的 vi 版本是 vim。它是 vi 的增强版，在功能上有很多扩充，也更容易使用。关键是，vim 是个开源软件。

vi 是一个全屏幕文本编辑器，具有文本编辑所需的所有功能。vi 以高效和快捷著称，这是 vi 能够在编辑器领域中保持几十年领先地位的原因。以下介绍 vi 的几个突出特点。

**1. 编辑功能强大**

vi 的编辑功能十分强大，除通常的编辑功能外，vi 还支持一些高级编辑特性，如正则表达式、宏和命令脚本。利用这些特性可以完成非常复杂的编辑任务，实现编辑的智能化和自动化。另一方面，vi 的功能又十分专注，它只是一个编辑器，没有其他功能。Linux 系统提供了许多专门用途的工具，如排版、排序、流过滤、E-mail、编译等软件。vi 可以和这些工具软件协同工作，从而实现几乎所有的文件加工处理任务。用一些小而精悍、功能专一的工具结合起来完成复杂的处理功能，这正是 UNIX 的设计哲学。

**2. 适用于各种版本的 UNIX/Linux 系统**

vi 是 UNIX/Linux 系统的标准文本编辑器，几乎每一台 UNIX/Linux 系统上都会有 vi，甚至在 Windows、Macintosh，OS/2 乃至 IBM 大型机 S/390 系统上都能见到 vi 的某个版本。这是其他编辑器无法相比的。

**3. 适用于各种类型的终端**

vi 得以广泛应用的原因之一是它对终端设备的广泛适应性。不管是只有打字机键盘加 Esc 键的简单终端，还是受通信限制的远程终端，或是配有完备的功能键和鼠标的现代化终端，都可以很好地支持 vi 完成文本编辑工作。

**4. 使用灵活快捷**

广泛适应性带来的问题是繁多的命令。对于同一项编辑操作，vi 提供了许多不同的命令。vi 的命令都很简练，往往是单个字符或少数几个字符的组合。对初学者来说，使用这些命令并不方便。但对于熟练的用户来说，更多的选择意味着更大的自由，简单的命令意味着更少的击键次数。正因为如此，vi 被看作是 Linux 开发人员和系统管理员的编辑利器。初学者经过一段时间的使用，也会逐渐习惯 vi 的操作方式，并形成自己特有的操作风格。

### 3.1.2　vi 的工作模式

vi 是一个多模式的软件，它有 3 种基本工作模式。在不同的工作模式下，它对输入的内容有不同的解释。vi 的基本工作模式如下：

**1. 命令模式**

命令模式(normal mode)用于完成各种文本编辑工作。在命令模式下，输入的任何字符都作为命令来解释执行，屏幕上不显示输入内容。

**2. 输入模式**

输入模式(insert mode)用于完成文本录入工作。在输入模式下，输入的任何字符都将作为文件的内容被保存，并显示在屏幕上。

**3. 末行模式**

末行模式(last line mode)也称为 ex 模式。在末行模式下，光标停留在屏幕的最末行，在此接收输入的命令并执行。末行模式用于执行一些全局性操作，如文件操作、参数设置、查找替换、拷贝粘贴、执行 Shell 命令等。

在文本编辑过程中，用户可以控制 vi 在这 3 种工作模式之间进行切换，完成各种编辑工作。3 种模式之间的转换关系如图 3-1 所示。

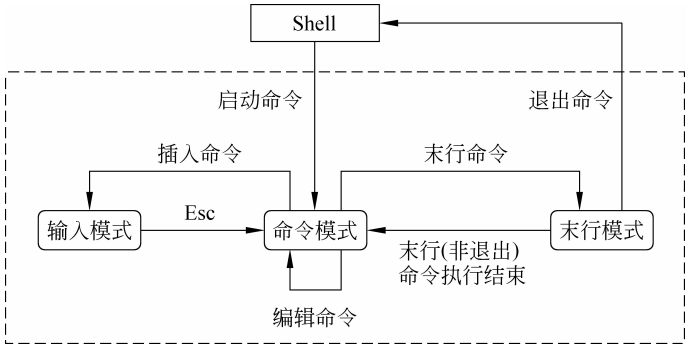



**图 3-1　vi 工作模式的转换**

### 3.1.3 vi 的基本工作流程

启动 vi 的方法是在 Shell 下输入 vi 命令，命令格式是：

```
vi [文件]
```

vi 的启动过程是：先建立一个编辑缓冲区，若指定了文件且该文件已存在，则将其内容读到编辑缓冲区中；若指定的文件不存在，则建立此文件。随后 vi 显示全屏幕编辑环境，将光标定位在第 1 行第 1 列的位置上。图 3-2(a)是 vi 启动后的初始界面。屏幕末行显示的是文件的名称、状态、行数和字符数、光标位置(行号，列号)等信息。光标位置的字符通常用反显方式或下划线方式显示。“～”表示编辑区的空行，它们不是文件的组成部分。

vi 启动后首先进入命令模式。此时，用户可以使用 vi 的编辑命令进行文本的输入和修改。进入输入模式的方法是按 Insert 键或插入命令字母，见图 3-2(b)。输入完成后按 Esc 键返回命令模式，见图 3-2(c)。此后可以使用各种编辑命令对已输入的文本进行修改，具体的用法在第 3.2 节介绍。注意：编辑命令只是修改调入编辑缓冲区中的文件的副本，文件本身不会被修改。所以，编辑完成后，要用末行命令：wq 将修改后的内容保存到文件中并退出 vi。若此次运行未对原文件做任何修改，则可用：q 命令退出。图 3-2(d)示意了退出命令的用法，更多的文件操作和退出命令在第 3.3 节介绍。

**例 3.1** vi 的基本用法如图 3-2 所示。

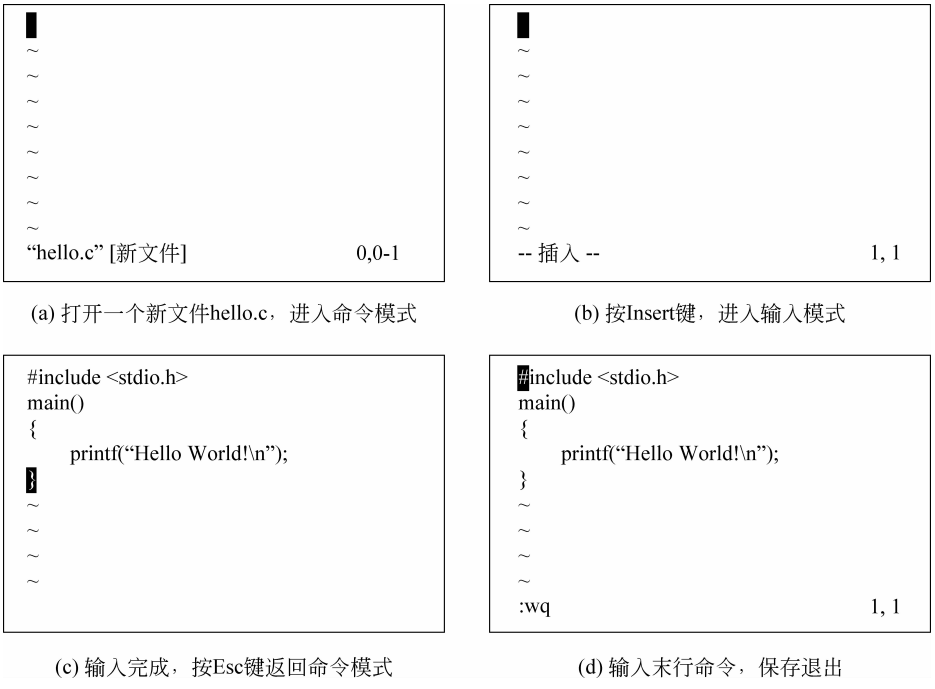



**图 3-2 vi 的基本工作流程**

## 3.2　vi 基本命令

vi 的命令繁多，但通常的编辑工作只需要掌握其中一小部分命令。而对于一些特殊的操作或编辑任务，总能够在 vi 手册中找到适当的命令。

vi 的常用编辑命令分为以下几类：

- 移动光标命令；
- 插入与删除命令；
- 修改与替换命令；
- 拷贝、粘贴与选择命令；
- 复原与重复命令。

vi 的命令通常是简单的字符(如 a、I、c)或是字符组合(如 dw、cc)。这就是说，仅仅通过普通键盘就可以实现所有编辑工作，完全不倚赖于鼠标和控制键。因此，熟练使用这些字符命令能够提高编辑的效率。注意，vi 的命令是区分大小写的。

尽管只用字符命令就可以完成所有编辑工作，vi 还是提供了对现代键盘上的编辑键的支持。适当地使用这些熟悉的按键将使编辑操作更加轻松。表 3-1 列出了这些键在不同模式下的作用。

**表 3-1　vi 按键功能说明**

| 按　键 | 命 令 模 式 | 输 入 模 式 | 末 行 模 式 |
|---|---|---|---|
| Home | 移动光标到行的最前面 | 同左 | 同左 |
| End | 移动光标到行的最后面 | 同左 | 同左 |
| PageDown | 向下翻一页 | 同左 | 向下翻找历史命令 |
| PageUp | 向上翻一页 | 同左 | 向上翻找历史命令 |
| Delete | 删除光标位置的字符 | 同左 | 同左，行尾时同 Backspace |
| Insert | 进入输入模式 | 替换－插入 | 无效 |
| Backspace | 光标前移一个字符 | 删除光标前的字符 | 同左 |
| Space | 光标后移一个字符 | 空格 | 同左 |
| Enter | 光标下移一个字符 | 换行 | 提交命令 |
| ←↑↓→ | 按箭头方向移动光标 | 同左 | ←、→前后移动光标，↑、↓上下翻找历史命令 |

### 3.2.1　光标定位与移动

在输入或修改文本前，应先将光标移到适当的位置。vi 不支持用鼠标移动光标的方式，但可以用命令来移动光标。以下是常用的移动光标命令：

**0**、**$**　　光标移至行首、行尾。同 Home、End 键。

^　　光标移至行首第 1 个非空格字符。

[*n*]G　光标移到第 *n* 行，未指定 *n* 时移到末行。

[*n*]|　光标移到第 *n* 列，未指定 *n* 时移到首列。

**h、j、k、l**　光标向左、下、上、右移一个字符。同箭头键。

**b、w、e**　光标移到上一个词、下一个词首、本词词尾。

(、)、{、}　光标移到句首、句尾、段首、段尾。

注意：以上光标移动命令前带数字 n 时，表示重复移动 n 次。如：2h 为左移 2 格，3e 为移到后面第 3 个词的词尾。

### 3.2.2　文本输入与删除

**1. 文本的输入**

在输入文本内容之前，应先将光标定位在要输入的位置上，然后执行插入命令，进入输入模式。处于输入模式时，屏幕底部会显示“—插入—”提示，表示后续的输入都作为文件的输入内容。输入完成后按 Esc 键就可返回命令模式。

插入(insert)命令都是单字符命令，可以灵活地实现在当前光标位置的前、后、行首、行尾、上一行、下一行开始输入。常用的插入命令如下：

**a、A**　在光标位置后、行尾后开始插入。

**i、I**　在光标位置前、行首前开始插入。i 的作用与 Insert 键相同。

**o、O**　在光标所在行之后、光标所在行之前的新行开始插入。

**2. 文本的删除**

删除(delete)文本的最简单方法是将光标移到要删除的字符上，然后，每按一下 x 键或 Delete 键就会删掉一个字符。若要删除的文本较多时，可以使用下面更加灵活的删除命令：

**x、X**　删除光标处、光标前的字符。x 的作用与 Delete 键相同。

**dd**　删除光标所在的行。

**J**　删除当前行尾的换行符，使当前行与下一行合并为一行。

**d**＋定位符　删除从光标位置到指定位置范围内的字符。常用的有：

- **d0、d ^**　删除光标左面的文本。0 或^代表行首。
- **d $**　删除光标右面的文本。$ 代表行尾。
- **dG**　删除光标所在行之后的所有行。G 代表最后一行。
- **db**　删除光标处前的字符直到词首。b 代表词首。
- **de**　删除光标处的字符直到词尾。e 代表词尾。
- **dw**　删除光标处的字符直到下一个词的词首。w 代表下一词词首。

注意：以上命令前带数字 n 时，表示删除的范围扩大 n 倍。如 3dd 为删除 3 行，2de 为删除从光标开始的两个词。

**例 3.2**　插入与删除命令的用法(下划线处为光标位置)。

| | |
|---|---|
| 原文本： | Yestoday is Thursday.<br>Today is Friday. |

```
                                    Tomorrow is Saturday.

执行命令：dd                        Yestoday is Thursday.
                                    Tomorrow is Saturday.

移动光标：2w 或 ww                  Yestoday is Thursday.
                                    Tomorrow is Saturday.

执行命令：5x 或 xxxxx               Yestoday is day.
                                    Tomorrow is Saturday.

执行命令：iSatur<Esc>               Yestoday is Saturday.
                                    Tomorrow is Saturday.

执行命令：oToday is Sunday.<Esc>    Yestoday is Saturday.
                                    Today is Sunday.
                                    Tomorrow is Saturday.

移动光标：jb                        Yestoday is Saturday.
                                    Today is Sunday.
                                    Tomorrow is Saturday.

执行命令：d$                        Yestoday is Saturday.
                                    Today is Sunday.
                                    Tomorrow is

执行命令：aMonday.<Esc>             Yestoday is Saturday.
                                    Today is Sunday.
                                    Tomorrow is Monday.

执行命令：ISo,<Esc>                 Yestoday is Saturday.
                                    Today is Sunday.
                                    So, Tomorrow is Monday.
```

### 3.2.3　文本修改与替换

**1. 文本的修改**

文本修改(correct)是指改写一部分文本的内容,修改的过程是：先删除指定范围内的文本,然后插入新文本,最后按 Esc 键结束插入。以下介绍几个常用的修改命令：

**cc**　　修改光标所在的行。

**C**　　修改光标处到行尾的文本。

**c**＋定位符　修改光标到指定范围内的文本。常用的有：

**c0**、**c^**　修改光标左面的文本。

**c$**　　修改光标右面的文本。

**cG**　　修改光标所在行之后的所有行。

**cb**　　修改光标处前的字符直到词首。

**cw**　　修改光标处的字符直到词尾。

**cl**　　修改光标处的字符。

注意：以上命令前带数字 n 时，表示修改的范围扩大 n 倍。如 5cc 为修改从光标所在行开始的 5 行，3cw 为修改从光标开始的 3 个词。

**2. 文本的替换与替代**

替换(replace)是指用一个字符替换另一个字符，这是一种覆盖操作，替换后文本的长度保持不变。替代(substitute)则是指用多个字符取代一个字符或一行，是一个先删除后插入的操作。通常情况下，替代后的文本长度会发生变化。以下介绍常用的替换与替代命令：

**r**　用输入的字符替换光标处的字符。

**R**　用输入的文本逐个替换从光标处开始的各个字符，直到按下 Esc 键。

**s**　用输入的文本替代光标处的字符，用 Esc 键结束输入，等同于 cl。

**S**　用输入的文本替代光标所在的行，用 Esc 键结束输入，等同于 cc。

注意：以上命令前带数字 n 时，表示替换或替代的范围扩大 n 倍。如 4r 为用输入的字符替换从光标处开始的 4 个字符，2s 为用输入的文本替代从光标处的开始的两个字符。3S 为用输入的文本替代从光标所在的行开始的 3 行。

**例 3.3**　修改、替换与替代命令的用法。

| | |
|---|---|
| 原文本行： | So, Tomorrow is Monday. |
| 执行命令：**r**t | So, tomorrow is Monday. |
| 移动光标：**2w** | So, tomorrow is Monday. |
| 执行命令：**cw**June 1<Esc> | So, tomorrow is June 1. |
| 移动光标：**2b** | So, tomorrow is June 1. |
| 执行命令：**2s**must be<Esc> | So, tomorrow must be June 1. |
| 移动光标：**2b** | So, tomorrow must be June 1. |
| 执行命令：**c$**is a holiday!<Esc> | So, tomorrow is a holiday ! |
| 移动光标：**4bh** | So, tomorrow is a holiday! |
| 执行命令：**c^**Great!<Esc> | Great ! tomorrow is a holiday! |
| 执行命令：**S**I like holiday.<Esc> | I like holiday . |

## 3.2.4　文本拷贝、粘贴与选择

为实现文本的拷贝粘贴，vi 中设置了专门的缓冲区，可称其为剪贴板。拷贝(copy)操作是将指定的文本复制到一个剪贴板中；粘贴(paste)操作是将剪贴板中的内容插入到文本中。此外，前面介绍的删除命令其实是剪切(cut)操作，被删除的文本并非真正消失，而是暂存到剪贴板中，可以再粘贴到文本中。以下是常用的拷贝粘贴命令：

**yy**　　拷贝光标所在行。

**y＋定位符**　拷贝光标到指定范围内的文本。常用的有：

**y0、c ^**　拷贝光标左面的文本。

**y $**　拷贝光标右面的文本。

**yG**　拷贝光标所在行之后的所有行。

**yb**　拷贝光标处的字符直到词首。

**yw**　拷贝光标处的字符直到词尾。

**p、P**　若剪贴板中的内容是完整的行，则将这些行插入到光标所在行之后、之前；若不是完整的行，则将这些文本插入到光标处之后、之前。

注意：以上命令前带数字 n 时，表示拷贝和粘贴的范围扩大 n 倍。如 2yy 为拷贝从光标所在行开始的 2 行，3yw 为拷贝从光标开始的 3 个词。

此外，vim 还支持鼠标粘贴与复制。在输入模式下，将光标移到要粘贴的位置，用鼠标选中要复制的文本，再按鼠标中键即可。

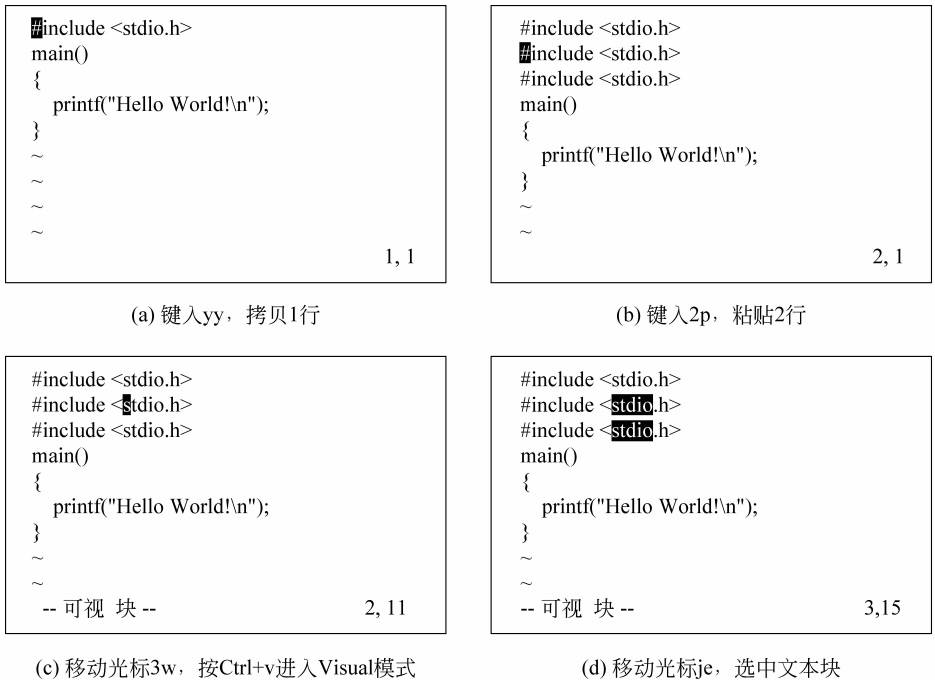
为了方便选择要处理的文本，vim 还提供了一种称为 Visual 的可视化文本选择模式，在此模式下，可以通过移动光标来直观地选择文本。对选中的文本可以执行某个编辑命令，如删除、拷贝、粘贴、修改等。若不做操作则按 Esc 键，放弃选择。选择文本的命令是：

**v**　以字符为单位选择连续的文本串。

**V**　以行为单位选择连续的文本行。

**Ctrl＋v**　按字符位置选择文本块。

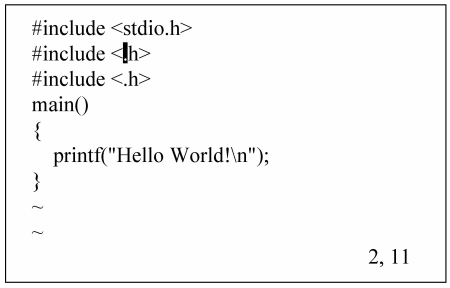
**例 3.4**　拷贝、粘贴与选择命令的用法如图 3-3 所示。



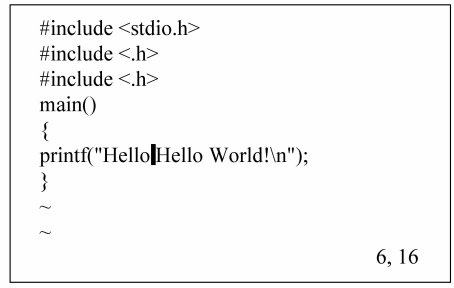

(a) 键入yy，拷贝1行　(b) 键入2p，粘贴2行

(c) 移动光标3w，按Ctrl+v进入Visual模式　(d) 移动光标je，选中文本块

**图 3-3　拷贝、粘贴和选择命令用法示意**




(e) 键入d，删除选中的文本




(f) 光标移到Hello字首，键入ywP，拷贝粘贴一词

**图 3-3　拷贝、粘贴和选择命令用法示意(续)**

### 3.2.5　撤销与重做

撤销(undo)即消除上一个命令所做的修改，恢复到命令执行前的样子。重做(redo)就是重复执行上一个命令。利用撤销和重做命令可以提高编辑的效率，减少击键次数。以下是常用的撤销与重做命令：

**u**　撤销上一个命令所做的修改。

**U**　撤销最近针对一行所做的全部修改。在对一行连续做了多处修改后，用此命令可以一次恢复全行。

.　重复执行前一个命令。

## 3.3　vi 常用末行命令

在命令模式下，输入“:”、“/”或“?”字符(称为 ex 转义字符)都将进入末行模式，随后的输入被解释为行命令，在屏幕末行显示。输入完成后按 Enter 键执行。末行命令执行结束后返回命令模式，或退出 vi。

末行命令主要有以下几类：

- 字符串搜索与替换命令；
- 文件操作与退出命令；
- 其他命令。

### 3.3.1　搜索与替换命令

**1. 字符串搜索**

要在一个大文件中查找某个字符串，可以用字符串搜索命令。执行搜索命令后，光标将停留在第一个匹配字符串的首字符处。按 n 键或 N 键则移到下一个匹配字符串之首。如果不存在匹配的字符串，则会在末行上显示“找不到模式”。搜索命令有以下两种：

/模式　从光标处向后搜索与指定模式匹配的字符串。按 n 键向后继续找。

?模式　从光标处向前搜索与指定模式匹配的字符串。按 N 键向前继续找。

例如，执行/and 命令，光标将从当前位置移到后面第一个“and”的字符 a 上。按 n 键

移到下一个“and”上。当搜索到文件尾时，再按 n 键则返回到文件头继续搜索。

**2. 字符串替换**

字符串替换使用 s 命令，它的功能是在指定的行中搜索与指定模式相匹配的字符串，并用另一个字符串替换它。

s 命令的一般格式是：

```
:[n1,n2]s/p1/p2/[g][c]
```

其中 n1、n2 表示目标行的行号范围，可以用“%”代表所有行；未指定范围时，目标行就是光标所在的当前行。p1 是用做搜索的字符串模式，p2 是用做替换的字符串模式。模式中可以用“^”代表行首，“$”代表行尾。s 命令可以带 g 和 c 选项。g 表示替换目标行中所有匹配的字符串，没有 g 的话则只替换目标行中第一个匹配的字符串。选项 c 表示替换前要求用户确认。

**例 3.5**　s 命令的用法。

```
:s/the/The/          将当前行中第 1 个 the 改为 The
:s/is/are/g          将当前行中所有 is 改为 are
:s/is a/has a/gc     将当前行中所有 is a 改为 has a。替换前提示用户确认
:1,6s/IF/if /g       将第 1 至 6 行中的所有 IF 用 if 替代
:%s/^/  /g           在所有行的行首处加 4 个空格
```

**3. 全局命令**

全局命令 g(global)的功能是在全文中搜索含有与指定模式相匹配的字符串的行，对匹配的行做标记。g 命令的格式是：

**:g**/*p1*　搜索所有包含 p1 字符串模式的行。

**:g!**/*p1*　搜索所有不包含 p1 字符串模式的行。

例如：:g/and 命令将找出所有含有“and”的行；:g!/and 命令找出所有不含“and”的行。

vi 的许多末行命令都是针对行的编辑命令(见 3.3.3 节)。g 命令可以与这些面向行的命令联合使用，它的作用是修饰这些命令，为其确定满足某个条件的目标行。在 g 命令的修饰下，这些行编辑命令就可用来完成面向全文的、按模式筛选的编辑操作。g 命令与其他命令联合使用的格式是：

**:g**/*p1*/命令　对所有包含 p1 的行执行指定的命令。

**:g!**/*p1*/命令　对所有不包含 p1 的行执行指定的命令。

例如，p 命令的功能是显示行，:g/and/p 命令将显示所有含有“and”的行；d 命令的功能是删除行，:g!/Note/d 命令将删除所有不含“Note”的行。

**4. 全局替换**

s 命令是面向行的字符串替换命令。s 命令经常与 g 命令联合使用，实现更灵活更细致的全局替换功能。

全局替换命令的一般格式是：

g 命令/s 命令

其含义是：先用 g 命令在文件中搜索含有某个模式的行，并做标记，然后用 s 命令对所有有标记的行执行搜索和替换。

常用的全局替换命令的格式有：

**:g**/*p1*/**s**/*p2*/*p3*/**g**　将文件中所有含有 p1 的行中的 p2 用 p3 替换。

**:g!**/*p1*/**s**/*p2*/*p3*/**g**　将文件中所有不含有 p1 的行中的 p2 用 p3 替换。

**:g**/*p1*/**s**//*p2*/**g**　将文件中所有的 p1 用 p2 替换。这里：g/p1/s//p2/g 是：g/p1/s/p1/p2/g 的简写，即，当 s 命令的搜索模式与 g 命令的搜索模式相同时，可以省略 s 中的搜索模式。注意，此处//之间没有空格。

**例 3.6**　全局替换命令的用法：

| | |
|---|---|
| **:g**/the/**s**//The/ | 将文中所有行的第 1 个 the 改为 The |
| **:g**/is/**s**//are/**g** | 将文中所有 is 改为 are |
| **:g**/Mary/**s**/1988//**g** | 将所有含有 Mary 的行中的所有 1988 去掉 |
| **:g**/printf/**s**/val/sum/**gc** | 将所有含有 printf 的行中的所有 val 改为 sum。换前先确认 |
| **:g!**/*/**s**/IF/if/**g** | 将所有不包含'*'的行中的所有 IF 用 if 替代 |

### 3.3.2　文件操作与退出命令

文件操作命令包括读文件和写文件操作。读文件就是将文件的内容读入编辑缓冲区中，写文件就是将编辑缓冲区的内容保存到文件中。在退出 vi 时，可以选择是否保存文件。以下是常用的退出和文件操作命令：

**:w**［文件］　写入指定文件。若未指定文件则写入当前文件。

**:q**　未修改原文件，不保存文件，直接退出。

**:q!**　修改了原文件，不保存文件，退出。

**:wq**、**:x**　保存文件并退出。

**:e!**　放弃修改，编辑区恢复为文件原样。

**:e** 文件　打开指定的文件，调入编辑区。

**:r** 文件　读入指定的文件，将文件内容插入到光标位置。

### 3.3.3　其他常用命令

**1. 行编辑命令**

行编辑命令用于对指定的行进行编辑。在指定行范围时，可以用“.”代表当前行，用“$”代表最后一行，用“%”代表所有行。常用的行编辑命令如下：

:*n*　跳至第 n 行。

:*n1*,*n2***co***n3*　将第 *n*1 至 *n*2 行之间的内容拷贝到第 *n*3 行下。

:*n1*,*n2***m***n3*　将第 *n*1 至 *n*2 行之间的内容移至第 *n*3 行下。

:*n1*,*n2***d**　将第 *n*1 至 *n*2 行之间的内容删除。

**:p**　　显示当前行的内容。

**2. 执行 Shell 命令**

用 vi 编辑文件时，可以在不退出 vi 的情况下执行 Shell 命令。执行命令期间 vi 暂时挂起，待命令执行结束后返回 vi 继续运行。执行 Shell 命令的格式是：

**:!***command*　执行 Shell 命令，command 为命令行。

**3. 设定 vi 选项**

vi 是一个高度可定制的编辑器，用户可以通过设置 vi 的选项来规定 vi 的一些外观和行为特性，使其满足特定的需求。设定 vi 选项的方法之一是使用 set 命令。常用的选项如下：

**:set all**　　显示所有选项。
**:set ai**、**:set noai**　设定、取消自动缩进。
**:set nu**、**:set nonu**　设定、取消行号显示。
**:help**　　显示 vi 的帮助手册。

## 习　　题

3-1　vi 编辑器的工作方式有哪些？相互之间如何转换？

3-2　用 vi --help 命令查看如何用 vi 打开一个文件，并将光标定位在第 10 行上。

3-3　解释下述 vi 命令的功能：
20G　18|　dM　x　cw　10cc　3rk　5s　7S　/this　:g/int/

3-4　要将文件中的所有字符串 str1 全部用字符串 str2 替换，应使用什么命令？若只替换每行中的第一个 str1，应使用什么命令？

3-5　在 vi 中拷贝一行文字并粘贴到另一位置用什么命令？

3-6　如何在 vi 中显示文本的行号？

3-7　如何恢复对一行文本所作过的修改？如何重复上一次修改操作？

3-8　如何放弃对一个文件的修改并退出？如何将编辑过的文件用不同的文件名保存？

# 第 4 章 Linux C 编程基础

C 语言是 Linux 系统的标准编程语言，绝大多数 Linux 的系统程序都是用 C 语言开发的。因此，每一位 Linux 程序员都需要掌握 C 编程技能。本章主要介绍在 Linux 系统下进行 C 程序开发的基础知识，包括 C 编译的基本方法、步骤和工具，并不涉及 C 语言编程的基础知识。有关 Linux 开发环境及辅助工具的介绍请参看附录 B。

## 4.1 Linux C 编程方法概述

开发一个 C 应用程序需要经过编辑、编译、调试、运行等多个步骤。在不同的开发环境中，实现这些步骤的方法有所不同。Windows 系统的开发环境是集成式的，即所有的开发活动都是在一个单一的应用界面中完成的；而 Linux 的主流开发环境是由一个个独立的工具构成的工具集，每个工具用于完成一项特定的工作。因此，从事 Linux C 程序的开发首先需要了解这些工具及其使用方法。

在 Linux 系统上进行 C 程序开发的基本方法和工具如下：

**1. 编辑源代码**

在 Linux 系统中，任何一款文本编辑器都可以用来编写程序源代码，其中尤以 vi/vim 和 Emacs 最为强大。除了一般编辑功能外，这两款编辑器还具有强大的定制功能，可以根据需要进行定制以适合编程的各种需要和习惯，因而被誉为程序员的编辑神器。关于 vi/vim 的介绍见第 3 章。此外，图形界面的编辑器（如 gedit 等）因便于使用而得到初学者的青睐。

**2. 编译可执行代码**

编译器用于将程序源代码转换为目标系统的可执行机器代码。Linux 系统上默认的 C 编译器是 gcc。关于 gcc 的介绍见 4.2 节。

对于具有一定规模的软件项目来说，由于源程序数量较多，单独使用 gcc 生成可执行文件并不方便。此时还需要借助工具来辅助构造软件。Linux 系统上默认的软件构造工具是 make。关于 make 的介绍见 B.2 节。

**3. 调试代码**

可执行代码中可能存在有运行时（runtime）错误。查找和修改运行时错误的过程称为调试（debug）。最简单的调试手段是在程序代码的适当位置加入 fprintf()函数，输出程序的动态运行信息。通过分析这些信息来判断出错的位置和原因。更为便利有效的手段是利用调试工具来辅助调试。Linux 系统上最常用的调试器是 gdb。关于 gdb 的用法

介绍见B.3节。

**4. 运行程序**

Linux系统将用户的可执行程序与系统自带的命令一样看待，它们的区别仅在于所处的目录位置可能不同。系统命令对应的可执行文件位于系统默认的标准路径下（包括/usr/bin、/bin等），而用户自己编写的程序则可以位于任何地方。Shell在执行一个命令时，需要首先查找并加载其对应的可执行文件，若命令行中没有指明文件的路径名，则默认地在系统标准路径下查找。因此，执行一个系统命令时可以省略路径，而执行用户自己编写的程序时，如果其没有位于标准目录中的话，则需要指明程序的路径名。例如，要执行当前目录下的一个可执行文件myprog，命令行应是：./myprog。若省略了路径前缀./的话，系统会因找不到可执行文件而报错。

**5. 寻求帮助**

对编程的全过程均有帮助的工具是系统自带的联机手册，它可以取代C语言的参考手册，供程序员随时查询。关于联机手册的介绍见4.3节。

## 4.2 gcc编译基础

gcc是Linux系统上的C编译器。它是一个完全免费的、符合ANSI C/C++标准的多平台编译系统，广泛使用在UNIX/Linux平台上。与其他编译工具相比，gcc的性能表现十分优越，用gcc编译的目标代码具有非常高的运行效率。本节介绍gcc的编译原理和基本用法。

### 4.2.1 gcc编译过程

编译器的工作是将源代码翻译成可执行代码。gcc编译的全过程分为4个阶段进行，包括预处理、编译、汇编和连接，如图4-1所示。

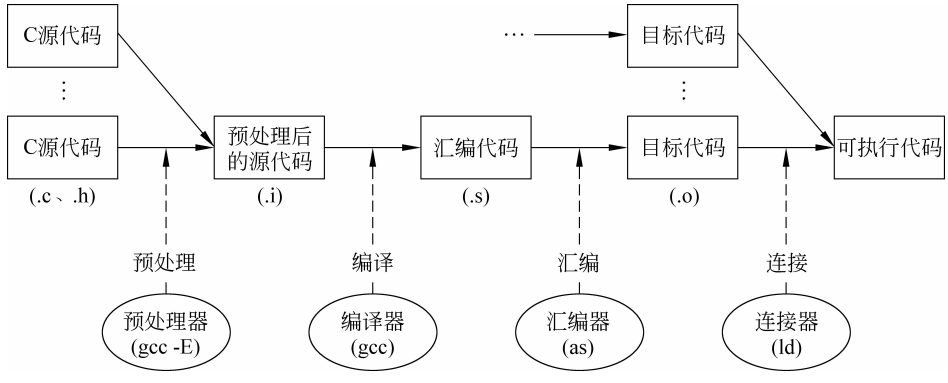



**图4-1 C程序的编译过程**

以上编译过程可以一次完成，也可以分阶段进行。程序员可以通过gcc命令的选项来灵活地控制整个编译过程。

**1. 预处理**

源代码中包含着一些预处理语句，如#include、#define 等。预处理(preprocessing)的任务是解析和处理源码中的预处理语句，执行文件包含、宏替换、条件编译等预处理工作。预处理的输入是若干个源代码文件(1 个.c 文件和 0 至多个.h 文件)，预处理的结果是生成一个后缀为“.i”的不含有预处理语句的源代码文件。在旧版本的 gcc 中，预处理是由 gcc 调用预处理器 cpp 来完成的。而在新版本的 gcc 中，预处理是作为编译的一个阶段，由 gcc 自己完成的。不过多数系统仍保留了 cpp 命令。

编程者需要了解 gcc 对于 include 预处理语句的处理方式，使 gcc 能够正确地找到头文件的存放位置。C 语言定义了两种头文件的说明方式，一种是系统标准头文件，用尖括号括起，如 #include <***.h>；另一种是用户自定义的头文件，用双引号括起，如 #include "***.h"。两者的区别在于 gcc 搜索头文件的默认路径有所不同。对于标准头文件，gcc 将在系统默认的头文件目录(通常是/usr/include)中搜寻；对于用户自定义的头文件，gcc 将先在被编译的.c 源文件所在的目录中搜寻，如果没有再到系统默认的头文件目录中搜寻。如果所需的头文件放在默认路径之外的其他目录中，就需要使用选项指示 gcc 增加头文件搜索路径。

**2. 编译与汇编**

编译(compilation)的工作是对预处理后的源代码进行词法和语法分析，生成目标系统的汇编代码文件，后缀名为“.s”。这步工作由 gcc 完成。汇编(assembly)的工作是对汇编代码进行优化，生成目标代码文件，后缀名为“.o”。这步工作由 gcc 调用汇编器 as 完成。

在默认情况下，gcc 会按照源代码中的语句直接编译生成目标代码。也就是说，编译后的代码的执行次序与源代码完全相同，没有经过优化处理。这种目标代码易于调试，且编译的时间也最短。若要生成更紧凑和更快速的目标代码，可以在编译时使用代码优化选项。对于大型程序来说，优化可以大幅度提高运行速度，减小代码的尺寸。不过，代码优化是以牺牲代码的易调试性和编译时间为代价的，所以通常只用于生成最终产品。

另外，若要使用调试工具对生成的代码进行调试，就需要在编译时加入调试选项，指示 gcc 在生成的代码中加入额外的调试信息。

**3. 连接**

目标代码是机器语言的代码，但还不是可执行的代码，因为模块化程序通常会有多个.c 源文件，每个都对应一个目标文件。另外，程序中还要引用一些库函数，它们的目标代码存放在系统库的目录下。这些目标模块之间存在着某种引用关系，需要连接在一起才可以工作。连接(linking)的任务就是解析目标代码中的外部引用，将多个目标代码文件连接为一个可执行文件。可执行文件是计算机可以直接运行的程序，如同 Windows 系统的.exe 文件。默认的可执行文件名是 a.out。不过为了避免重名，通常的做法是为它指定一个名字。与 Windows 系统不同，Linux 不限定可执行文件的后缀名，通常是不带后缀名。

连接工作由 gcc 调用连接器 ld 完成。连接程序 ld 在进行连接时，在系统默认的函数库目录(/lib、/usr/lib)中寻找并加载所需要的库文件。如果要使用放在其他目录下的库

文件，需要在gcc命令行中用选项指示ld到指定的目录中去寻找。

对每个C程序，ld都将自动加载C标准库libc，它包含了ANSI C所定义的所有标准C函数。如果程序中用到了其他库，如数学库、图形库、线程库、套接字库等，需要用连接库选项显式地指示ld加载该库。

C函数库分为静态库（后缀名为.a）和共享库（后缀名为.so）两种，两者都是C库函数的.o目标代码的集合，在连接时被连入到可执行文件中。两者的区别在于：当程序与静态库连接时，所有程序中用到的库函数的目标代码都被复制到最终的可执行文件中；而当程序与共享库连接时，可执行文件中只包含程序中用到的函数的引用表，而不是函数的目标代码。这些函数的目标代码只有在有程序调用它们时才被调入内存，并且可以被多个程序共享。因此，连接共享库的可执行文件比较小，节省磁盘空间和内存空间。由于共享库的优点，在两种版本的库都存在的情况下，ld将优先使用共享库进行连接。如需要使用静态库的话，必须在命令行中用选项指定。

### 4.2.2 gcc命令

gcc命令的格式是：

```
gcc [选项] 文件列表
```

gcc命令用于实现c程序编译的全过程。文件列表参数指定了gcc的输入文件，选项用于定制gcc的行为。gcc根据选项的规定将输入文件编译生成适当的输出文件。

gcc的选项非常多，这里只介绍一些常用的选项，它们大致可以分为以下几类：

**1. 过程控制选项**

过程控制选项用于控制gcc的编译过程。无过程控制选项时，gcc将默认执行全部编译过程，产生可执行代码。常用的过程控制选项有：

-E　预处理，产生预处理过的源代码，不编译。

-S　预处理＋编译，产生汇编代码，不汇编。

-c　预处理＋编译＋汇编，产生目标代码，不连接。

**2. 输出选项**

输出选项用于指定gcc的输出特性等，常用的有：

-o *filename*　指定生成文件的文件名为*filename*。无此选项时使用默认的文件名。各编译阶段有各自的默认文件名。可执行文件的默认名为a.out，其他阶段的默认输出文件名是由输入文件名更换相应的后缀名后得到的。如输入文件为foo.c，则-E过程输出的默认文件名为foo.i，-S过程输出的默认文件名为foo.s，-c过程输出的默认文件名为foo.o。

-Wall　显示所有的警告信息，而不是只显示默认类型的警告。建议使用。

**3. 头文件选项**

与头文件相关的选项是：

-I*dirname*　将*dirname*目录加入到头文件搜索目录列表中。当gcc在默认的路径中没有找到头文件时，就到本选项指定的目录中去找。

**4. 连接库选项**

与连接库相关的选项是：

-L*dirname*　将 *dirname* 目录加入到库文件的搜索目录列表中。

-l*name*　加载名为 lib*name*.a 或 lib*name*.so 的函数库。例如，-lm 表示连接名为 libm.so 的数学函数库。

-static　使用静态库。注意：在命令行中，静态加载的库必须位于调用该库的目标文件之后。

**5. 代码优化选项**

gcc 提供了几种不同级别的代码优化方案，分别是 0、1、2、3 和 s 级，用-O*level* 选项表示。默认是 0 级，即是不进行优化。典型的优化选项是：

-O　对代码进行基本优化(1 级优化)。在大多数情况下，这种优化会使程序执行的更快。

-O2　对代码进行深度优化，产生尽可能小和快的代码。这是 GNU 发布软件的默认优化级别。如无特殊要求，不建议使用 O2 以上级别的优化。

-Os　生成最小的可执行文件，适合用于嵌入式软件。

**6. 调试选项**

gcc 支持数种调试选项，最常用的是：

-g　产生能被 GDB 调试器使用的调试信息。-g 可以和-O 和-O2 连用，以便在与最终产品尽可能相近的情况下调试代码。

-pg　在程序里加入额外的代码，执行时产生供性能分析工具 gprof 使用的剖析信息，以便了解程序的耗时情况。

### 4.2.3 gcc 应用举例

以下用几个简单的“say hello”程序来说明 gcc 的用法。

**例 4.1**　最简单的 hello1 程序。

```
$ls
  hello.c
$cat hello.c                          #源文件
  #include<stdio.h>
  main()
  { printf("Hello world!\n");
  }
$gcc -o hello1 hello.c                #编译,生成可执行文件 hello1
$ls
  hello1  hello.c
$./hello1                             #运行
  Hello world!
$_
```

此例中,gcc的输入文件为源文件hello.c,选项-o hello1指定了输出文件的名称。

**例4.2** 有多个源文件和自定义头文件的hello2程序。

```
$ls
  hello.c    print.c    print.h
$cat print.h                                  #print.h源文件
  #define borderchar  '*'
  void my_print(char *);
$cat hello.c                                  #hello.c源文件
  #include "print.h"
  main()
  { char my_string[]="Hello world!";
    my_print(my_string);
  }
$cat print.c                                  #print.c源文件
  #include<stdio.h>
  #include<string.h>
  #include "print.h"
  void my_print(char *str)
  { int i;
    for (i=0; i<strlen(str)+4; i++) printf("%c", borderchar);
    printf("\n");
    printf ("%c %s %c\n", borderchar, str, borderchar);
    for (i=0; i<strlen(str)+4; i++) printf("%c", borderchar);
    printf("\n");
  }
$gcc -c hello.c                               #编译hello.c,生成hollo.o
$gcc -c print.c                               #编译print.c,生成print.o
$gcc -o hello2 hello.o print.o                #生成hello2
$ls
  hello2   hello.c  hello.o  print.c   print.h   print.o
$./hello2
  *****************
  * Hello world! *
  *****************
$mkdir include                                #创建include目录
$mv print.h include/                          #移动print.h文件到include目录下
$gcc -o hello2 hello.c print.c                #编译,输出错误信息
  hello.c:1:19:  print.h:  No such file or directory
  print.c:3:19:  print.h:  No such file or directory
  …
$gcc -o hello2 -Iinclude hello.c print.c      #编译,指定头文件目录
$ls
  hello2    hello.c  hello.o   include  print.c   print.o
$_
```

此例中，编译过程分为 3 步完成，先分别编译两个源文件，再将它们的目标代码连接起来。这 3 个 gcc 命令可以合并写为一个命令，即 gcc -o hello2 hello.c print.c。在最初的编译过程（前 3 个 gcc 命令）中，hello.c 文件与 print.h 文件都在当前目录中，因此执行正确；将 print.h 文件移到子目录中后，gcc 命令因找不到头文件而报错。使用-I 选项指定头文件目录后，gcc 得以正确进行编译。

**例 4.3** 使用特定的函数库的 hello3 程序。

```
$cat hello.c                          #hello.c 源文件
  #include "print.h"
  main ()
  { char my_string[]="Hello world!";
    my_print(my_string);
  }
$cat print.h                          #print.h 源文件
  void my_print(char *);
$cat print.c                          #print.c 源文件,全屏显示字符串
  #include<stdio.h>
  #include<curses.h>
  void my_print(char *str)
  { initscr();                        /*进入 curses 全屏显示模式,清屏幕*/
    move(5,15);                       /*移动光标到屏幕(5,15)坐标处*/
    printw ("%s", str);               /*向 curses 屏幕输出字符串*/
    refresh();                        /*刷新物理屏幕,显示出字符串*/
    sleep(5);                         /*程序暂停 5 秒*/
    endwin();                         /*结束 curses 全屏显示模式,恢复行模式显示*/
  }
$gcc -o hello3 hello.c print.c                #编译,ld 报错
  /tmp/cc0Nrf8Y.o(.text+0x7): In function 'my_print':
  : undefined reference to 'initscr'
  …
  /tmp/cc0Nrf8Y.o(.text+0x52): In function 'my_print':
  : undefined reference to 'endwin'
  collect2: ld returned 1 exit status
$gcc -o hello3 hello.c print.c -lcurses       #编译,使用 libcurses.so 库
$./hello3
  (全屏显示 Hello world!字符串,如图 4-2 所示。5 秒后屏幕复原)
$_
```

此例中使用了 curses 函数库，用于实现字符方式的全屏幕输入/输出（注：当运行在软件虚拟的“终端”中时，全屏就是全窗口）。在第 1 个 gcc 命令中，连接时 ld 因无法找到 curses 函数的目标代码而报错。加入-lcurses 选项后，ld 加载 libcurses.so 库使连接成功。


**图 4-2 hello3 执行时的屏幕显示**

## 4.3 C联机手册

联机手册(manual)是Linux系统的标准联机技术文档,每个Shell命令、系统调用、C标准库函数、配置文件等都有相应的手册页(manual page)。查看手册页的工具是man命令。2.2.4节介绍了用man命令查询Linux命令的手册页的方法,本节进一步介绍如何从联机手册中获取编程的技术支持。

联机手册按内容分为9节(section),每节对应一种类型的手册页。其中第1节是Linux命令手册,涵盖了所有Linux命令;第2、3节是Linux程序员手册,分别包含了所有的系统调用和C库函数的手册页。这些手册页提供了开发C程序所需的最准确、最完整的资料,因而是编程者的有力工具。

man命令的格式是:

```
man [选项] 名称
```

名称是要查询的对象的名称,选项指定查询的方式。默认情况下,man命令按节号从前向后顺序查找全部手册,并显示第1个与名称相匹配的手册页。若要查询某特定的节则需要使用选项来指定。常用的选项有以下几个:

*i*[p] 指定在第i节中查找。1为系统命令,2为系统调用,3为C库函数。p表示POSIX手册页,即:1p为POSIX命令,3p为POSIX库函数。

-k 以给定的名称参数作为关键字,查询与之相匹配的所有手册页及其简短说明。如果记不清命令或函数的完整名字,用此选项。

-f 查询全部与名称相符的手册页。如果想了解一个名称是什么,用此选项。

**例 4.4** 使用man命令查询联机手册:

```
$man exit              #显示 exit 命令的手册页,等同于 man 1 exit
$man 2 exit            #显示 exit()系统调用的手册页
$man 3 exit            #显示 exit()函数的手册页
$man -k exit           #显示所有名称中含有 exit 的手册页及其说明
  _Exit (2)            -terminate the calling process
  _Exit (3p)           -terminate a process
  …
```