

```
[pi@raspberrypi pifun]$ cp -r moarpi moarpi2
[pi@raspberrypi pifun]$ ls
moarpi moarpi2 pi pi2 raspberry
[pi@raspberrypi pifun]$
```

`mv` 命令与 `cp` 命令在这点上稍有不同，即在移动目录时无需使用 `-r` 选项，显然我们是不可能抛开目录中的文件只移动目录本身的。

通过 `rm` 命令来删除文件

至此，我们已经了解了如何创建、复制以及移动文件和目录。现在我们将尝试使用 `rm` 命令进行删除操作。显然，`rm` 命令是一把锋利的双刃剑。不小心的话，`rm` 命令可以轻易地毁掉整个服务器上的文件。就我们所知，这样的意外确实发生过。

首先，我们来看如何删除一个目录。除了 `rm` 命令外，`rmdir` 命令（`remove directory` 的缩写）也可以进行目录的删除。使用 `rmdir` 命令时需要注意的是，它只能删除空目录，即便目录里只有一个文件，命令也会拒绝执行。这样做是出于安全方面的考虑，但很多情况下显然并不方便。

```
[pi@raspberrypi pifun]$ rmdir moarpi2
[pi@raspberrypi pifun]$ rm moarpi
rm: cannot remove 'moarpi': Is a directory
[pi@raspberrypi pifun]$
```

可以看到，用 `rmdir` 命令可以直接删除空目录 `moarpi2`，而使用 `rm` 命令时却出了点问题。这和 `cp` 命令的情况有点相似，删除整个目录的危险性比复制目录还要高得多，所以我们需要通过 `-r` 选项告知 `rm` 命令我们想要进行递归的（`recursively`）删除。

```
[pi@raspberrypi pifun]$ rm -r moarpi
[pi@raspberrypi pifun]$
```

执行成功了！在删除内容较多的目录时，有时你会看到 `rm` 命令给出大量的错误提示。例如，不具有目录里的某些文件的写权限。你可以使用 `-f` 选项（`force`，强制的）来忽略提示强制执行。这相当于告诉 `rm` 命令“去他的水雷，全速前进！”¹。一般情况下这或许不是个坏主意，但凡事总有例外（下面这条命令绝对不该被执行）。

```
[pi@raspberrypi pifun]$ rm -rf /
```

这条命令会删除文件系统上的所有文件，它甚至会清空连接在系统上的 USB 存储设备以及挂载在系统上的网络共享目录。为了避免这种的意外发生，我们应该坚持使用普通账号处理日常任务。普通账户没有删除系统文件的权限，不过即便如此仍需要小心，避免误删自己的文件。因为 Linux 认为你能够对你自己的行为负责，使用 `rm` 命令时不会再进行事前确认。所以，再三确认总是没有错的，小心驶得万年船。

`rm` 命令也可以通过文件路径删除单个文件，这样做时无需使用 `-r` 选项。

¹ 译者注：戴维·法拉格特将军的名言。在莫比尔湾战役中，时年 64 岁的法拉格特将军这一声怒吼拯救了整个舰队。<http://www.baik.com/wiki/莫比尔海战>。

```
[pi@raspberrypi pifun]$ rm pi2
[pi@raspberrypi pifun]$ ls
pi raspberry
[pi@raspberrypi pifun]$
```

至此我们已经学习了各种与文件系统有关的重要命令。

绝对路径和相对路径

在 Linux 中，我们可以通过两种方式定位一个文件或目录。

- 从根目录 (/) 开始，给出绝对路径。
- 从特定文件、目录或是 “.” 和 “..” 开头，给出相对路径。

虽然看起来有些不同，但这两种方式都可以实现对文件及目录的定位。当路径从根目录 (/) 这样的固定位置开始时，我们就称之为“绝对路径”，无论你当前的工作目录¹是什么，使用绝对路径可以唯一确定一个位置。这就像古时人们总喜欢用矗立于城镇中央的大钟楼作为原点指路，如此一来就不会存在歧义了。

另一方面，相对路径依赖于当前的工作目录。在相对路径中，./表示当前目录，../表示当前目录的上一级目录。像../test.txt 这样的路径可以简捷地表示上两级目录中的 test.txt 文件所在的位置。当然我们也可以用/home/pi/test.txt 这样的绝对路径来进行定位。而当目录层次比较多时，使用相对路径可能会带来些困扰。如果当前的工作目录为/home/pi/Documents/Work/Projects_2012/Spreadsheets/，如果我们需要访问上面提到的 test.txt 文件，使用相对路径定位：../../../test.txt。显然此处相对路径定位不如绝对路径定位来得直接。

可以看到，以上两种定位方式各有千秋。所以，因地制宜地使用以上两种定位方式，才能有效地提高操作效率。

4.5 用户和用户组

用户和用户组是 Linux 安全体系中不可或缺的重要一环。我们将粗略涉及用户和用户组的基础知识，为以后更好地理解文件权限相关内容打下基础。

在 UNIX 哲学中，可以通过用户名来唯一标识用户。在一个系统中，每个用户名都能对应到特定的人或是对象（比如 Web 服务器程序可能有独立的用户名）。前文中，我们基本在使用 pi 用户进行操作，而我们也尝试了使用 root 用户。用户名是 Linux 系统识别你身份的关键。

用户组的定义十分简单。每个用户只属于一个主用户组（primary group），但可以属于多个用户组。在一个大学的系统中，一个学生的用户名可能属于其私有用户组（这在 Red Hat 和 Debian 等发行版中十分常见），同时也是学生用户组、研究用户组以及其院系用户组的成员。用户组对系统管理员来说是一种十分有用的工具，它使管理员能够方便地以组为单位管理用户，从而简化了诸如文件权限管理等问题。

¹ 译者注：指当前所在的文件夹，可以通过 pwd 命令查看。

当你在树莓派上创建新用户时，系统会用相同的名字创建一个用户组。在另一些系统中，新用户会被默认加到一个已有用户组中，在接下来的章节中我们会看到，这样的做法存在着安全隐患。由于私有用户组是专属于单一用户的，不会有用户因为偶然被加到你的用户组中而能访问你的文件。这也是在较新的系统中文件的所有者和其用户组同名的原因。

4.6 文件权限

通过设置文件权限，你可以决定谁能够访问你的文件，以及他们具体能做什么。你可以设置 3 种不同的权限：文件读取权限、文件写入权限以及文件执行权限。

当然，对单一文件针对所有用户设置权限还是有些缺乏灵活性，有时我们希望能只让特定用户组具有访问权限。这时，前面提到的用户和用户组的知识就有用武之地了。在 Linux 中，每个用户对子文件的访问具有 3 种角色。

- 户（User）：文件的所有者。
- 用户组（Group）：文件的所有者所在组。
- 其他（World）：文件所有者为其他人。

不同的角色具有不同的权限，我们可以分别设置不同角色的读、写、执行权限。在尝试设置权限前我们首先来看一下如何查看一个文件当前的权限设置情况。这里我们需要使用 `ls` 命令的 `-lh` 选项。

```
[pi@raspberrypi pifun]$ ls -lh
-rw-r--r-- 1 pi pi 0 Oct 7 16:29 pi
-rw-r--r-- 1 pi pi 0 Oct 7 16:58 pi2
-rw-r--r-- 1 pi pi 0 Oct 7 16:29 raspberry
[pi@raspberrypi pifun]$
```

`-lh` 选项其实是 `-l` 选项和 `-h` 选项合并在一起得到的。`-l` 选项告诉 `ls` 命令我们想要通过列表的形式显示文件详细信息，而 `-h` 选项告诉 `ls` 命令使用更具可读性的方式显示文件大小。

我们可以看到在每行的行首显示了文件的权限设置信息，包括不同种类权限的具体设置以及该文件属于哪个用户和用户组。这里我们以 `raspberry` 文件为例，进行分析。

```
-rw-r--r-- 1 pi pi 0 Oct 7 16:29 raspberry
```

文件权限字段。

```
-rwxrwxrwx
```

可以看到该字段总共有 10 个标识位，当某一位所代表权限设置为空时，就显示为连字符“-”。需要注意的是对普通文件来说，首位永远显示为连字符“-”。相应的，目录的首位显示为“d”，连接或是快捷方式显示为“l”以示区别。余下的 9 个标识位 3 位一组分为 3 组，分别表示用户、用户组和其他人对应的读、写、执行 3 种权限。相应权限打开时，对应位会以字母显示，未打开时，显示连字符“-”。如果我们打开 `raspberry` 文件的所有权限，其文件权限部分会显示如下：

```
-rwxrwxrwx
```

简单地做个分组后就可清楚地看到其权限设置。

```
-   rwx   rwx   rwx
```

通过第一组 3 个字母我们可以知道，文件的所有者具有读、写、执行所有 3 种权限，并且同用户组的用户和其他用户也拥有所有的权限。为了理解这些权限的具体含义，我们需要确定文件的所有者是谁，我们可以在文件权限字段后面看到相关信息。

```
pi pi
```

很简单吧。还记得吗？在较新的 Linux 系统中，用户拥有以其用户名命名的私有用户组。上面就是这样的情况。第一个 pi 表示文件所有者，第二个 pi 表示文件所有者所在的用户组（一般情况下是用户所属的主用户组）。我们再来看下原始 raspberry 文件的信息。

```
-rw-r--r-- 1 pi pi 0 Oct 7 16:29 raspberry
```

我们可以看到，pi 用户具有读、写权限，pi 用户组具有读权限，其他用户具有读权限。Linux 会通过用户的身份来进行权限限定。

- 1. 如果你的用户名和文件所有者一致，将会适用用户权限。
- 2. 如果你不是文件所有者，但属于文件所标识的用户组，将会适用用户组权限。
- 3. 除了以上两种情况，将会适用其他用户权限。

在上面看到的例子中，用户组权限和其他用户权限相同。这意味着，除了文件所有者外的其他用户都具有一样的权限。当然，凡事总有例外。root 用户是超级用户，他不受文件权限制约，并可以随意修改文件权限和文件所有者。

文件执行权限可以控制一个文件是否可以作为程序执行，这是一种限制用户执行程序的安全机制。不过需要注意的是，具有该文件读取权限的用户可以复制文件，并执行复制后的文件。

执行权限标识位对于目录的含义有些不同。显然，我们不能执行目录，所以这个标识位表示是否可以浏览目录（比如使用 ls 命令）。如果用户具有目录的读取权限而没有执行权限，那么他们可以读取目录里文件的内容，但必须提前知道相应的文件名，因为他们没有权限浏览目录。这样的设置在图形界面程序中会带来一些困扰，毕竟大部分图形界面程序都没有考虑路径输入的问题。

还有一种称为“文件扩展属性”的机制，它更为灵活，也更加复杂，这种机制和 Windows 处理权限的方式有些相似，有兴趣的读者可以尝试搜索一下。

设置文件权限

我们可以使用 chmod 命令来改变文件权限，具体参数见表 4-1。

表 4-1 chmod 命令可用参数

参数种类	可选参数	参数种类	可选参数	参数种类	可选参数
用户角色	u（文件所有者）	操作种类	+（添加权限）	操作对象	r（读取权限）
	g（文件所属用户组）		-（删除权限）		w（写入权限）
	o（其他用户）		=（指定权限）		x（执行权限）
	a（所有用户）				

根据操作种类的不同，我们可以使用 `chmod` 命令对文件权限进行 3 种操作：添加、删除以及指定。前两种操作不会影响操作外的其他权限，而指定权限后，所有未指定的权限都会被清空。

首先我们来尝试清空 `pi` 文件的所有文件。

```
[pi@raspberrypi pifun]$ chmod a= pi
[pi@raspberrypi pifun]$ ls -lh
----- 1 pi pi 0 Oct 8 03:52 pi
[pi@raspberrypi pifun]$
```

可以看到，文件的权限已清空。那么 `chmod` 命令具体是如何使用的呢？权限的改变涉及 3 个部分（见表 4-1）：

- 改变哪些用户的权限
- 如何改变权限
- 改变哪种权限

在上面 `pi` 文件的例子中，用户角色部分我们使用 `a` 参数改变所有用户的权限，操作种类使用 `=` 参数来指定权限，操作对象部分留空，表示不指定任何一种权限。因为使用 `=` 参数所有未指定的权限都会被清空，所以所有权限都会被清空。

如果我们想为自己打开所有权限的话，可以这样做（此处执行权限还没有用处，具体参见第 7 章）：

```
[pi@raspberrypi pifun]$ chmod u+rwX pi
[pi@raspberrypi pifun]$ ls -lh
-rwx----- 1 pi pi 0 Oct 8 03:52 pi
[pi@raspberrypi pifun]$
```

将这条命令分解来看，使用 `u` 参数改变所有者权限，使用 `+` 参数来添加权限，`r`、`w`、`x` 3 个参数表示我们要改变的权限种类。接着让我们看看如何为文件所属用户组和其他用户添加读取权限。

```
[pi@raspberrypi pifun]$ chmod go+r pi
[pi@raspberrypi pifun]$ ls -lh
-rwxr--r-- 1 pi pi 0 Oct 8 03:52 pi
[pi@raspberrypi pifun]$
```

还有另一种用数字代替字母参数的权限设置方式，要想设置为上面两个例子最终结果中的权限，可以这样做：

```
[pi@raspberrypi pifun]$ chmod 744 pi
[pi@raspberrypi pifun]$ ls -lh
-rwxr--r-- 1 pi pi 0 Oct 8 03:52 pi
[pi@raspberrypi pifun]$
```

在这种设置方式中，不同的权限有不同的权值。

- 读取权限：4
- 写入权限：2
- 执行权限：1

3 个数字位上 3 种权值分别相加后对应了文件所有者、文件所属用户组和其他用户的权限。例如，如果打开所有权限的话是 $4+2+1=7$ ，只打开读取权限的话是 $4+0+0=4$ 。在上面的例子中，我们需要的数字为 744。使用数字的设置方式朴素直接，使用字母的方式简明易懂。它们在功能上并没有区别，用户可以按喜好选择。

现在你已经学会如何随意改变权限了，那么该如何改变文件的所有权呢？相比设置文件权限，改变文件所有权这类的操作并不算常见。事实上，除了 root 用户以外的其他用户都无法执行此类操作。这很好理解，如果你不小心将自己文件的所有权让出了，可没有后悔药吃。所以还是交给全知全能的 root 用户来做吧。

我们可以通过 sudo 命令临时获得 root 用户权限。前文中已经提及，这个命令可以使其后的命令以 root 用户身份执行。为了避免意外，系统中有一份可以使用 sudo 命令的认证用户列表，别担心，pi 用户就在其中。

使用 sudo 命令时，我们只需要将 sudo 加到我们需要执行的命令之前，就这么简单。首次使用这一命令时，系统会提示输入密码，注意这里的密码是当前用户的密码，而不是 root 用户的密码。这主要是为了确认当前用户身份，然后 sudo 命令再去检查当前用户可以执行哪些操作。这意味着，如果系统上的用户相当多时，你可以通过设置 sudo 命令使他们可以执行一些高阶命令，而不需要交出你手上的万能钥匙（root 用户）。

首先，我们先来尝试下使用 chown 命令将文件所有权转给 root 用户。

```
[pi@raspberrypi pifun]$ chown root pi
chown: changing ownership of 'pi': Operation not permitted
[pi@raspberrypi pifun]$
```

Linux 用教科书式的方式告诉我们：无法进行操作。就像之前提到的一样，只有 root 用户可以进行此类操作。所以我们需要使用 sudo 命令临时变成 root 用户。

```
[pi@raspberrypi pifun]$ sudo chown root pi
[pi@raspberrypi pifun]$ ls -lh
-rwxr--r-- 1 root pi 0 Oct  8 03:52 pi
[pi@raspberrypi pifun]$
```

成功了！当然，你也可以使用上面的命令将文件所有权转交给其他用户。此外，你还可以通过 chgrp 命令改变文件的所属用户组。chgrp 命令也需要使用 sudo 命令执行，使用方法与 chown 十分相似。

```
[pi@raspberrypi pifun]$ sudo chgrp root pi
[pi@raspberrypi pifun]$ ls -lh
-rwxr--r-- 1 root root 0 Oct  8 03:52 pi
[pi@raspberrypi pifun]$
```

可以看到，现在 pi 文件属于 root 用户和 root 用户组了。在大多数情况下我们需要同时改变文件所有者和文件所属用户组，为此 chown 命令提供了一种便捷的方法。这里我们来尝试将文件还给 pi 用户和用户组。

```
[pi@raspberrypi pifun]$ sudo chown pi:pi pi
[pi@raspberrypi pifun]$ ls -lh
-rwxr--r-- 1 pi pi 0 Oct  8 03:52 pi
[pi@raspberrypi pifun]$
```

在上面的命令中，使用冒号将所指定的用户和用户组隔开。对于 `chown` 或者 `chgrp` 命令还有一点需要注意，它们只会改变命令中所指定文件的所有权。如果指定的是个目录，那么它们会改变目录的所有权，但不涉及其中的文件。有时这不是我们想要的，和 `cp` 和 `rm` 命令略有不同，这里我们需要使用 `-R` 参数（这里参数必须大写）。使用 `chown` 或者 `chgrp` 命令时，按下回车键前请务必再三确认，如果错误执行，可没有后悔药吃。

4.7 软链接和硬链接

Linux 下可以使用 `ln` 命令创建链接。链接可以分为两类。

- 软链接：它与 Window 中的“快捷方式”十分相似。软链接本质上是一个标明特定文件真实地址的指向性文件。我们可以使用 `ln` 命令的 `-s` 参数创建软链接。
- 硬链接：与软链接相比硬链接的情况有些复杂，使用硬链接相当于为一个文件创建了两个名字。这听来多少有些奇怪，随着软链接的普及，硬链接的使用也日益稀少。此外，硬链接还必须使用在存在相应支持的文件系统上。硬链接的主要优势就在于可以为同一个位置提供两个不同的名字，不带参数直接使用 `ln` 命令就可以创建硬链接。

注意 为了可移植性以及避免歧义，建议尽量使用软链接。

我们来看下面这个例子。

```
[pi@raspberrypi pifun]$ ln pi pi1
[pi@raspberrypi pifun]$ ln -s pi pi2
[pi@raspberrypi pifun]$ ls -lh
-rw-rw-r-- 2 miggyx miggyx  0 Oct  8 08:14 pi
-rw-rw-r-- 2 miggyx miggyx  0 Oct  8 08:14 pi1
lrwxrwxrwx 1 miggyx miggyx  3 Oct  8 08:33 pi2 -> pi
[pi@raspberrypi pifun]$
```

使用硬链接创建的 `pi1` 可以和原来的文件名 `pi` 无差别地使用。可以看到在文件权限部分后面有一个数字，`pi` 和 `pi1` 的条目上数字都为 2。这表示当前文件存在两个不同的文件名。`pi2` 这个软链接的情况更为有趣，可以看到它的所有权限都被打开了，我们可以根据需要进行相应的权限设置。还可以发现，在原来显示文件名的位置标明了软链接指向的文件。

链接在日常操作中非常实用。当你想把某个程序使用的数据目录移到一个更大硬盘时，移动目录后只需要在原来的路径创建一个软链接就可以了，这样的操作对程序不会有任何影响。有些时候，一个软链接就可以解决各种头痛的问题，为你节省大把的时间。

本章小结

在这一章中我们了解了文件系统的方方面面。我们回顾了文件系统的发展历程，了解了 Linux 文件系统的结构，学习了使用命令行进行创建、复制、移动和删除等基本文件操作。我们讨论了文件权限的构成以及如何按需设定文件权限，以及如何改变文件的所有权。在章节的最后我们介绍了软链接和硬链接相关的内容。

在下一章中，我们会涉及在树莓派上最为常用的命令。它们就像是十八般武器，工欲善其事，必先利其器。用好了它们，我们才能在日常工作中披荆斩棘，化险为夷。让我们开始第 5 章吧！

重要命令

现在，我们手上的树莓派已经完成了安装和配置，整装待发。在之前的章节里我们已经逐步熟悉了整套系统，了解了各种基础操作，是时候更进一步了。

本章会介绍各种重要命令，帮助你成为一名 Linux 进阶用户。我们会涉及如何提权成为 root 用户、检查磁盘空间、终止进程、配置用户空间以及监视环境变量等方方面面的内容。这些都是系统管理员的日常操作，对系统的运维十分重要。闲话少叙，让我们开始吧。

5.1 掌控全局

“根”(root)作为本源，支持着树木的生长。在类 UNIX 系统（包括 Linux）中，root 有两个含义：系统管理员（即 root 用户，参见第 3 章）以及文件系统中的顶层目录（即根目录，参见第 4 章）。为了进行系统管理，我们需要使用 root 用户来改变系统的核心设置。也正因为如此，root 用户是一把双刃剑，更大的能力意味着更大的责任，过于锋利的剑也更容易伤到使用者。root 用户的意外操作导致各种令人头疼的问题甚至必须重装系统的情况屡见不鲜。因此，我们建议尽量不要直接使用 root 用户登录 Shell。

在前面两章中，我们已经接触过使用 sudo 命令来避免直接使用 root 用户。你可以将 sudo 命令作为任何命令的前缀使用（例如：sudo ls），这样这些命令就将以 root 用户身份运行。虽说这样做多少有些麻烦，但却要安全得多。甚至有一部分 Linux 发行版已经不再默认为 root 用户设置密码，而是为创建的用户授予 sudo 命令的权限。如此一来，所有的系统管理都可以由普通用户通过 sudo 命令来执行。

在上一章中我们介绍过，执行 sudo 命令时，为了安全起见需要输入当前用户密码以验证身份。并且，sudo 命令还会检查用户认证列表，以确认哪些用户可以使用该命令及其具体的操作权限。我们可以通过 visudo 命令来编辑这份认证用户列表，如此重要的操作显然需要使用 sudo 命令来运行。

```
$ sudo visudo
```

这一命令会使用系统默认的文本编辑器打开用户认证列表（一般是 vi 或 nano，参见第 6 章）。在这份列表中我们可以看到 sudo 的各种设置。最为重要的内容在 userauth 部分。

```
root    ALL=(ALL:ALL) ALL
```

上面这行内容表示 root 用户可以在任意主机上向任意其他主机上发送执行命令。看起来有点晕不是吗，简单来说就是表示使用 root 用户登录便可以在系统中畅行无阻。文件中的另一行是这样的：

```
%sudo  ALL=(ALL:ALL) ALL
```

这行表示，属于 sudo 认证列表的用户可以通过 sudo 命令获得和 root 用户完全一致的权限。行首的 % 表示 sudo 表示的是一个用户组的名字。不难发现每行的结构都是这样的：

```
<用户> <从哪台主机>=(<访问哪台主机>:<使用什么命令>) <使用什么命令和选项>
```

简单起见，我们先以 pi 用户为例，比照 root 用户的设置。

```
pi      ALL=ALL  ALL
```

为了方便理解，我们用 ALL 代替了 (ALL:ALL)。那么如何让 pi 用户在使用 sudo 命令时不需要输入密码呢？我们可以在最后一个 ALL 的前面添加 NOPASSWD。

```
pi      ALL=ALL  NOPASSWD: ALL
```

这样我们就可以省略 pi 用户使用 sudo 命令时输入密码的环节。不过，这样设置存在安全风险，不妨限定只有在执行 visudo 命令时不需要密码。将最后一个 ALL 替换为我们需要指定的命令 visudo。这里我们不能直接输入 visudo，而需要给出绝对路径/usr/sbin/visudo。

```
pi      ALL=ALL  NOPASSWD: /usr/sbin/visudo
```

这样我们基本了解了这份认证列表文件的设置方法，通过添加更多的条目我们可以进行更细致的配置。

下面我们来介绍 su 命令 (switch user, 切换用户)。顾名思义，这个命令是用来切换当前用户的。如果当前用户不是 root 用户，系统会提示你输入你所要切换到的用户的密码。这里有一个小技巧，即我们可以通过 sudo 命令和 su 命令的组合来快速切换到 root 用户，只需输入自己的用户密码而不需要 root 用户的密码。如此一来我们就可以解决在某些 Linux 发行版中“root 用户密码未知”的问题了。

```
$ sudo su -
```

- 表示你想要登录到系统中，因为没有提供其他参数，默认使用 root 用户登录。在 - 后面再以用户名为参数就可以切换到相应用户。切换到 root 用户还有一种更为简捷的办法。

```
$ sudo -i
```

5.2 快去读那该死的文档

我们最喜欢的一句技术俚语是：RTFM (Read The Flaming Manual, 快去读那该死的文档)。当然，我们可以找人帮忙解决问题。但总有一份文档——一个安静的助手，在那儿等你去翻阅，

不离不弃。绝大多数的工具附带的文档都可以通过 `man` 命令查阅，如果你想要查阅 `sudoers` 的文档，使用如下命令。

```
$ man sudoers
```

就这么简单，文档就会浮现眼前。`man` 命令显示的文档实际上是使用 `less` 命令（稍后我们会做介绍）打开的，可以使用键盘上的方向键导航。你也可以按 `/` 键进行搜索，按 `N` 键翻页。好了，快试试用 `man` 命令查阅各种文档吧！

5.3 系统资源监控

监控系统资源的使用情况是系统管理员的日常必修功课。下面要介绍的命令可能对普通用户来说不是很常用，但借助它们我们可以轻而易举地了解系统运行状况。

首先我们来看下 `top` 命令，它可以显示当前各种系统资源的使用情况以及各进程的系统资源占用情况。`top` 命令十分强大，可以事无巨细地显示各种有用的信息，图 5-1 是其运行截图。

```
top - 22:22:11 up 6 days, 20 min, 1 user, load average: 0.11, 0.08, 0.06
Tasks: 66 total, 1 running, 64 sleeping, 1 stopped, 0 zombie
%Cpu(s): 0.0 us, 0.0 sy, 0.0 ni, 99.9 id, 0.0 wa, 0.0 hi, 0.1 si, 0.0 st
KiB Mem: 123156 total, 109784 used, 13372 free, 30340 buffers
KiB Swap: 0 total, 0 used, 0 free, 41608 cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
7445	raspbian	20	0	4608	1312	960	R	16.0	1.1	0:00.08	top
1	root	20	0	1832	680	592	S	0.0	0.6	0:22.11	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.45	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/0
5	root	20	0	0	0	0	S	0.0	0.0	0:59.85	kworker/u:0
6	root	-2	0	0	0	0	S	0.0	0.0	0:03.20	rcu_kthread
7	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	khelper
8	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kdevtmpfs
9	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
10	root	20	0	0	0	0	S	0.0	0.0	0:04.19	sync_supers
11	root	20	0	0	0	0	S	0.0	0.0	0:00.18	bdi-default
12	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kblockd
13	root	20	0	0	0	0	S	0.0	0.0	0:00.28	khubd
14	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	rpciod
16	root	20	0	0	0	0	S	0.0	0.0	0:00.42	khungtaskd
17	root	20	0	0	0	0	S	0.0	0.0	0:00.51	kswapd0
18	root	20	0	0	0	0	S	0.0	0.0	0:00.00	fsnotify_mark
19	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	nfsiod
20	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	crypto
27	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kthrotld
28	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	VCHIQ-0
29	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	VCHIQR-0
30	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	dwc_otg
31	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	DWC Notificatio
32	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kworker/u:1
33	root	20	0	0	0	0	S	0.0	0.0	2:11.51	mmcgq/0
34	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	ext4-dio-unwrit
125	root	20	0	2904	1324	728	S	0.0	1.1	0:00.65	udevd
190	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	bcm2708_spi.0
951	root	20	0	0	0	0	S	0.0	0.0	1:53.66	RIKTHREAd
956	root	20	0	5588	1204	848	S	0.0	1.0	0:19.18	wpa_supplicant
983	root	20	0	6332	2148	396	S	0.0	1.7	0:00.01	dhclient
1102	root	20	0	2304	796	584	S	0.0	0.6	0:02.77	rpcbind
1133	statd	20	0	4112	1728	1148	S	0.0	1.4	0:00.06	rpc.statd
1145	root	20	0	2736	444	264	S	0.0	0.4	0:00.00	rpc.idmapd
1452	root	20	0	27928	1452	1008	S	0.0	1.2	0:03.53	rsyslogd
1482	daemon	20	0	2084	324	200	S	0.0	0.3	0:00.02	atd
1524	root	20	0	3744	784	628	S	0.0	0.6	0:02.29	cron

图 5-1 运行中的 `top` 命令

`top` 命令显示的内容不止数量众多而且更新频繁，多少有些让人眼花缭乱。别着急，我们会逐一解释，一步步帮助你理解这些信息。

系统运行时间和负载

`top` 命令显示的第一行内容给出了当前系统时间和系统已经启动了多久，之后给出了当前登录到系统的用户数以及系统负载。负载是一个比较难理解的概念，它是一个“元”变量，和当前系统的性能相关。负载部分由 3 个数字组成，分别表示了近 1 分钟、近 5 分钟以及近 15 分钟的平均负载情况，负载这一概念对系统计算资源的使用和需求给出了数值统计。

我们不妨使用高速公路来进行类比，CPU 使用率就像高速公路上用很多车在行驶，不过除了要知道有多少正在行驶以外，还需要知道有多少车正在高速公路入口排队。所以除了 CPU 使用率之外，应用程序等待 CPU 的时间也很重要。就像繁忙但是通畅的高速公路和有大量车正在排队的高速公路是截然不同的。

负载可以让你直观地看到系统计算资源的实时需求，这就是为什么它是系统资源的一项重要指标。通过负载数值的变化，系统管理员就可以掌握计算资源需求何时达到峰值，并及时地降低需求。

任务

`top` 命令显示的第二行内容列出了系统当前执行的任务，包括：

- 进程总数
- 活跃进程数
- 休眠进程数
- 中止进程数
- 僵尸进程数

以上大部分的数字可以用来了解当前系统运行了哪些程序，唯一的例外是僵尸进程。它代表的是程序已经结束运行但仍驻留在系统内存中的进程。通常，产生僵尸进程的原因是子进程未能向父进程发送返回值（或者父进程没有很好地处理返回值），所以它们就像行尸走肉一般徘徊在内存里，苦苦等待最终能够逃出升天。

如果系统中存在大量僵尸进程，这表明产生这些进程的父进程出现了问题。因此，我们可以通过检测父进程的运行状态和日志文件（一般程序会将日志文件保存在硬盘上，方便系统管理员进行分析），来推断系统上是否存在更严重的问题。

CPU 利用率

`top` 命令显示的第三行内容是 CPU 利用率相关数值。通过这些数值我们就可以知道系统各部分当前对计算资源的占用情况。具体数值如下：

- `us`（表示用户程序占用百分比）

- sy (表示系统程序占用百分比)
- ni (表示系统中 nice 程序占用百分比)
- id (表示空闲百分比)
- wa (表示等待 I/O 操作的进程百分比)
- hi (表示等待硬件中断进程百分比)
- si (表示等待软件中断进程百分比)
- st (表示虚拟机管理程序“偷”走的时间)

在这些数值中,我们往往发现 id 这个值会比其他值大得多,这表示系统的计算资源比较空闲。如果 wa、hi、si 以及 st 这些值较高的话可不是个好消息,这表明系统在等待进程执行。如果这些值一直保持在很高的数字上的话,我们就应该进行检测,看看到底是哪个进程出了问题还是硬件出了故障。

内存使用量

第 4 行和第 5 行表明了系统内存的占用情况。第 4 行显示了物理内存的占用,第 5 行则显示了虚拟内存的占用。这两行分别会显示内存总数(total)、已用内存空间(used)以及空闲内存空间(free)这 3 部分内容。在这两行的最后分别显示了 buffers 和 cached。这两个互有关联的概念往往使 Linux 新手十分困惑。要想区分这两个概念我们需要了解一下内存分配机制, Linux 有时会将一部分内存用作缓存,这些被用作缓存的内存在别的应用程序需要时仍然可以释放出来使用 Linux 会把这部分内存标记为已被占用,但它们随时可以再次提供给应用程序使用。这里显示的缓存使用量的数据比较难理解,不过可以通过别的命令来了解这些资源究竟是如何使用的,我们会在后续的内容中讲到。

进程列表

在第 5 行下面我们可以看到一张很大的进程列表,这张表显示了所有进程的准实时运行情况,可以清楚地看到每个进程的资源占用。下面我们来看看这张表中的每一列数值分别代表了什么。

- PID: 进程编号。
- USER: 进程所属用户。
- PR: 进程优先级。
- NI: 进程的 nice 值¹。
- VIRT: 进程所消耗的虚拟内存大小。
- RES: 实际驻留虚拟内存大小。
- SHR: 进程所用的共享内存大小。
- S: 进程状态(比如:休眠中、运行中、僵尸进程)。
- %CPU: CPU 占用率。
- %MEM: 内存占用率。

¹ 译者注: nice 值是 Unix 操作系统表示中表示进程优先级的数值, nice 值越大的进程的优先级越小。

- TIME+: 进程启动后占用的 CPU 时间
- COMMAND: 进程对应的启动命令行

top 命令还可以通过一系列的热键来定制进程列表的显示顺序，在 top 命令的手册中对这部分内容有着详尽的描述。

至此我们已经对 top 命令有了大致的了解。如果你想了解系统资源的使用情况，它是一个很棒的工具，通过它你还可以深入了解系统当前的工作状态。

使用 free 命令查看内存分配情况

前文提到，相比使用 top 命令，查看内存分配情况有一种更为简单的方法，说的就是 free 命令。使用 free 命令你可以指定所需的容量单位（-k 表示 KB，-m 表示 MB，-g 表示 GB，以此类推）。这里我们使用 MB 为单位。

```
$ free -m
```

输出如图 5-2 所示。

	total	used	free	shared	buffers	cached
Mem:	231	157	74	0	18	79
-/+ buffers/cache:		59	172			
Swap:	99	0	99			

图 5-2 简单的 free

-/+ buffers/cache 这行已用内存和空闲内存的值与上一行并不相同。当缓存的（buffers/cache）内存数据相当多时，我们就可以知道内存已经接近峰值。通过 top 和 free 这两个工具我们便可以对系统的运行状况洞察秋毫。

磁盘使用

在看过 CPU、内存使用情况这类无时无刻不在变化的数据后，让我们再来看看磁盘占用这样相对稳定的数据。绝大多数的操作系统都可以方便地查看当前磁盘的使用情况，Linux 当然也不例外。我们可以使用 df 命令（disk free，磁盘空闲）来查看磁盘使用情况，不需要再使用额外的参数就可以获得所需的基本数据。df 命令的输出如图 5-3 所示。

```
df: '/root/.gvfs': Permission denied
```

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
rootfs	1828696	1757188	0	100%	/
/dev/root	1828696	1757188	0	100%	/
tmpfs	23688	504	23184	3%	/run
tmpfs	5120	0	5120	0%	/run/lock
tmpfs	47376	4	47372	1%	/tmp
tmpfs	10240	0	10240	0%	/dev
tmpfs	47376	0	47376	0%	/run/shm
/dev/mmcblk0p1	57288	34784	22504	61%	/boot

图 5-3 糟糕，没有空间啦

从图中可以看到，挂载在根目录下的磁盘空间已经基本满了。这里使用的容量单位是 KB，显然以现在的磁盘大小来看，使用这样的单位并不方便查阅。另外我们可以看到，输出的最上方

df 命令给出了一个报错信息。

幸运的是，这两个问题都有解决的办法。使用 `sudo` 命令运行可以解决报错的问题。通过添加 `-h` 参数，我们就可以看到可读性更高的容量单位了。`-h` 这一参数一般有两个含义：1、表示帮助信息（help），2、表示请求使用更便于阅读的单位。

```
$ sudo df -h
```

图 5-4 中的输出是不是看起来清爽多了？

Filesystem	Size	Used	Avail	Use%	Mounted on
rootfs	1.8G	1.7G	0	100%	/
/dev/root	1.8G	1.7G	0	100%	/
tmpfs	24M	504K	23M	3%	/run
tmpfs	5.0M	0	5.0M	0%	/run/lock
tmpfs	47M	4.0K	47M	1%	/tmp
tmpfs	10M	0	10M	0%	/dev
tmpfs	47M	0	47M	0%	/run/shm
/dev/mmcblk0p1	56M	34M	22M	61%	/boot

图 5-4 可读性更高的 df 命令输出

可以看到，加上 `-h` 参数后，df 命令会根据容量的实际大小选择合适的单位。查阅 df 命令的手册可以发现更多有用的参数。

有时候我们需要了解更为详细的磁盘使用情况，这时我们就需要用到 `du` 命令（disk usage，磁盘使用）了。它可以显示每个文件，甚至递归地显示目录占用的磁盘大小。du 命令同样也可以使用 `-h` 参数。

```
$ du -h
4.0K  ./cache/dconf
12K   ./cache/menus
576K  ./cache/midori/web
4.0K  ./cache/midori/thumbnails
24K   ./cache/midori/icons
608K  ./cache/midori
4.0K  ./cache/openbox/sessions
12K   ./cache/openbox
640K  ./cache
12K   ./dbus/session-bus
16K   ./dbus
5.0M  ./tor
12K   ./ssh
1.8M  ./python_games
8.0K  ./dillo
20K   ./vnc
4.0K  ./gvfs
124K  ./gnupg
8.0K  ./fltk/fltk.org
12K   ./fltk
48K   ./Desktop
8.0K  ./config/lxpanel/LXDE/panels
16K   ./config/lxpanel/LXDE
```

```

20K    ./config/lxpanel
8.0K   ./config/pcmanfm/LXDE
12K    ./config/pcmanfm
8.0K   ./config/lxterminal
28K    ./config/midori
4.0K   ./config/enchant
28K    ./config/openbox
104K   ./config
76K    ./fontconfig
16K    ./netsurf
40K    ./local/share/gvfs-metadata
36K    ./local/share/webkit/icondatabase
40K    ./local/share/webkit
88K    ./local/share
92K    ./local
9.0M   .

```

一段很长的输出不是吗？这是因为 `du` 命令默认递归地显示目录的磁盘占用。我们可以通过 `-s` 参数（summary）来简化输出结果，只显示当前目录的磁盘占用总值。

```

$ du -sh
9.0M   .

```

这样看起来就简洁多了。我们看到在输出里有表示当前目录的符号，这是因为当没有指定目录时，`du` 命令会使用当前目录作为默认参数。下面我们尝试对 `/var/log` 目录使用 `du` 命令。这个目录主要用来保存系统日志文件，我们需要使用 `sudo` 命令来执行，以保证命令不会报错。

```

$ sudo du -sh /var/log
9.8M   /var/log

```

好了，通过上面两个工具，我们已经可以事无巨细地了解系统中的磁盘占用情况，大到整块磁盘，小到一个文件，一切尽在掌握中。

管理进程

学习使用 Linux 系统最重要的一件事可能就是了解如何管理进程。计算机的所有工作都是由进程执行的，如果不能有效地管理进程，你就很难充分地利用的计算机性能。所以本节我们介绍的第一个命令是 `ps`，该命令的名称是 **process snapshot**（进程快照）的缩写，它可以列出所有正在运行中的进程。执行该命令后，你会看到如图 5-5 所示的列表。

PID	TTY	TIME	CMD
1951	pts/0	00:00:01	bash
2162	pts/0	00:00:00	ps

图 5-5 当前进程

从图中看到两个正在运行的进程，一个是 `bash` 终端，另一个则是 `ps` 本身，看起来似乎有点少，这是由于默认情况下，`ps` 只会显示当前用户的进程。要看到更多的进程，需要在命令后面加

上一些参数。例如要显示系统当前所有进程，可以使用 `-ef` 参数，再次执行 `ps` 会看到输出的列表明显变长了，如图 5-6 所示。

root	1756	1	0	17:05	?	00:00:00	Xtightvnc :1 -desktop X -auth /root/.Xauthority -geometry 1280x800 -depth 24 -rfbwait 120000
root	1760	1	0	17:05	?	00:00:00	/bin/sh /root/.vnc/xstartup
root	1771	1760	0	17:05	?	00:00:00	/usr/bin/ck-launch-session /usr/bin/dbus-launch --exit-with-session x-session-manager
root	1805	1	0	17:05	tty1	00:00:00	/sbin/getty --noclear 38400 tty1
root	1806	1	0	17:05	tty2	00:00:00	/sbin/getty 38400 tty2
root	1807	1	0	17:05	tty3	00:00:00	/sbin/getty 38400 tty3
root	1808	1	0	17:05	tty4	00:00:00	/sbin/getty 38400 tty4
root	1809	1	0	17:05	tty5	00:00:00	/sbin/getty 38400 tty5
root	1810	1	0	17:05	tty6	00:00:00	/sbin/getty 38400 tty6
root	1011	1	0	17:05	?	00:00:00	/sbin/getty -L ttyAMA0 115200 vt100
root	1815	1771	0	17:05	?	00:00:00	/usr/bin/ssh-agent /usr/bin/ck-launch-session /usr/bin/dbus-launch --exit-with-session x-session-manager
root	1817	1	0	17:05	?	00:00:00	/usr/sbin/console-kit-daemon --no-daemon
root	1887	1771	0	17:05	?	00:00:00	/usr/bin/lxsession -s LXDE -e LXDE
root	1890	1	0	17:05	?	00:00:00	/usr/bin/dbus-launch --exit-with-session x-session-manager
root	1891	1	0	17:05	?	00:00:00	/usr/bin/dbus-daemon --fork --print-pid 5 --print-address 7 --session
root	1897	1887	0	17:05	?	00:00:00	openbox --config-file /root/.config/openbox/lxde-rc.xml
root	1899	1887	0	17:05	?	00:00:12	lxpanel --profile LXDE
root	1900	1887	0	17:05	?	00:00:01	pcmanfm --desktop --profile LXDE
root	1906	1	0	17:05	?	00:00:04	/usr/lib/notification-daemon/notification-daemon
root	1908	1	0	17:05	?	00:00:01	nm-applet
root	1910	1	0	17:05	?	00:00:00	/usr/lib/arm-linux-gnueabi/lib/lsppolkit
root	1914	1	0	17:05	?	00:00:00	/usr/lib/arm-linux-gnueabi/lib/libmenu-cache/libexec/menu-cached
root	1916	1	0	17:05	?	00:00:00	/usr/lib/gvfs/gvfsd
root	1918	1	0	17:05	?	00:00:00	/usr/lib/gvfs/gvfs-fuse-daemon -f /root/.gvfs
root	1919	1724	0	17:05	?	00:00:01	sshhd: pi [priv]
root	1933	1	0	17:05	?	00:00:00	/usr/lib/arm-linux-gnueabi/lib/gconf/gconfd-2
root	1937	1	0	17:05	?	00:00:00	/usr/lib/gvfs/gvfs-gdu-volume-monitor
root	1939	1	0	17:05	?	00:00:00	/usr/lib/udisks/udisks-daemon
root	1940	1939	0	17:05	?	00:00:00	udisks-daemon: not polling any devices
root	1943	1	0	17:05	?	00:00:00	/usr/lib/gvfs/gvfs-gphoto2-volume-monitor
root	1945	1	0	17:05	?	00:00:00	/usr/lib/gvfs/gvfs-afc-volume-monitor
pi	1950	1919	0	17:05	?	00:00:03	sshhd: pi@pts/0
pi	1951	1950	0	17:05	pts/0	00:00:01	-bash
root	1980	1724	0	17:09	?	00:00:00	sshhd: pi [priv]
pi	1987	1980	0	17:09	?	00:00:00	sshhd: pi@pts/1
pi	1988	1987	0	17:09	pts/1	00:00:00	-bash
root	2094	1724	0	17:48	?	00:00:00	sshhd: pi [priv]
pi	2102	2094	0	17:48	?	00:00:00	sshhd: pi@pts/2
pi	2103	2102	0	17:48	pts/2	00:00:00	-bash
root	2137	2	0	18:02	?	00:00:00	[flush-179:0]
root	2138	2	0	18:08	?	00:00:00	[kworker/0:1]
root	2189	2	0	18:18	?	00:00:00	[kworker/0:0]
root	2208	2	0	18:25	?	00:00:00	[kworker/0:2]
pi	2209	1951	0	18:26	pts/0	00:00:00	ps -ef

图 5-6 ps -ef

这是一个很长的一个列表，因为 `ps -ef` 命令列出了所有用户进程和后台进程。如果你想给这个长列表加上分页控制，你可以在命令后面加上 `| less`，这样就可以在列表中前后翻页了（`less` 命令将在下一章讲解）。

```
$ ps -ef | less
```

`ps` 命令类似 `top` 命令，不过它显示的内容相对简单，有时反而更加方便。`ps` 命令显示的进程列表中，每一列的意义如下所示。

- UID：表示拥有该进程的用户。
- PID：表示该进程的 PID 号。
- PPID：表示该进程的父进程 PID 号。
- C：表示该进程的 CPU 使用率。
- STIME：表示该进程的启动时间。
- TTY：表示该进程使用的控制台。
- TIME：该进程累计使用的 CPU 时间。
- CMD：该进程的命令行参数。

我们看到了系统中的所有进程，之后我们能干什么呢？仔细观察一下 `ps` 命令显示的进程列表，其中最重要的内容就是 PID，PID 唯一的表示了某一个进程，所有对于进程的操作都需要其 PID。有了 PID，我们就可以介绍下一个命令了，即 `kill`。

杀死进程

`kill` 命令名副其实，它可以杀死进程，该命令还可以通过同样的机制对指定的进程进行其他操作。在 Linux 和 UNIX 操作系统中，系统通过信号机制（signal）向应用程序的进程发送消息，应用程序自己实现接收到不同信号时的行为，不过有些信号的确有固定的意义，所以 `kill` 的功能实际是将指定的信号发送给指定的进程。树莓派的 `kill` 命令可以发送 64 种不同的信号，使用如下命令发送：

```
$ kill -l
```

这里本书不会列出所有的 64 种信号，不过我们还是介绍最主要的几种。

- Signal 1 (SIGHUP)：hang-up（挂起）的缩写，该信号通知应用程序重新启动。
- Signal 3 (SIGQUIT)：该信号通知应用程序清理自身资源并退出。
- Signal 6 (SIGABRT)：该信号通知程序终止并立即退出。
- Signal 9 (SIGKILL)：该信号将立即终止应用运行。

这些是主要的几种终止应用运行的信号，其中使用 Signal 9 是最常见方法。结合 `kill` 命令的用法如下：

```
kill -<signal> <PID>
```

例如，你想强制退出一个进程，那么执行以下命令：

```
$ kill -9 <PID>
```

从命令中可以看到，首先通过 `ps` 命令获得 PID，之后就可以杀死指定 PID 对应的进程了。

通过/proc 读取信息

获得了 PID 之后，我们再来看看第 4 章介绍过的 `/proc` 文件系统，所有的进程信息都可以在这里找到。你可以执行一下 `ls /proc`，会看到一些文件和一些以数字命名的目录，每个目录名都对应了一个进程的 PID，目录中保存了所有和该进程相关的信息文件，有些文件对于日常使用也是很有用的。

使用最广泛的应该就是 `/proc/cpuinfo`，该文件中保存了整个系统中所有处理器的信息。你可以使用 `cat` 命令（concatenate 的缩写）将该文件的内容输出至终端的屏幕。

```
$ cat /proc/cpuinfo
```

你应该会看到类似以下的输出。

```
Processor       : ARMv6-compatible processor rev 7 (v6l)
BogoMIPS       : 697.95
```

```

Features      : swp half thumb fastmult vfp edsp java tls
CPU implementer : 0x41
CPU architecture: 7
CPU variant    : 0x0
CPU part       : 0xb76
CPU revision   : 7

Hardware      : BCM2708
Revision      : 0003
Serial        : 000000007a8a46ba

```

这就是 Linux 系统读取的树莓派处理器内部信息，如果你想探究一下其他处理器，也可以使用相同的方法。

除了 `cpuinfo`，`/proc` 中还有一些检查系统时很有用的文件。例如 `/proc/<pid>/cmdline` 文件，该文件中存储了 PID 对应进程的命令行参数。如果你想了解某个进程的启动参数，可以使用 `cat` 命令输出该文件内容。

5.4 文件命令

在上一节中，我们已经使用过 `cat` 命令来输出文件内容。除了该命令，还有很多命令可以帮助你处理文件中的内容。首先我们介绍 `less` 命令，之前我们也使用过该命令。通过该命令，你可以像在文件编辑器中浏览文件一样，在屏幕中上下滚动文件内容。该命令在查看日志文件时很有用，还可以结合 `ps` 等命令一起使用，方便浏览大量的输出内容。在 `less` 中，你可以使用上下箭头或者 **Page Up** 和 **Page Down** 键翻页。你还可以按 **/** 键进入搜索模式，之后输入需要搜索的内容，`less` 将会高亮所有匹配的文字，你可以按 **N** 键向下继续搜索，按 **Shift+N** 向上继续搜索。

除了这些基本功能，还可以让 `less` 自动监视文件的变化并一直显示文件的最后一行。该功能在监视运行日志时很有用，避免不断地重新打开日志文件。在 `less` 中按 **Shift+F** 键启用自动刷新，按 **Shift+G** 键让其显示文件的末尾，这样就会自动显示新加入文件末尾的内容。在 `less` 中使用 **G** 键可以跳转到指定行号，例如你想浏览文件的第一行，输入 `1g`。所有在 `less` 中使用的按键命令都可以在命令行参数中使用，例如在 `less` 命令后加上 `-N` 参数，可以直接启用显示行号。

介绍完 `less`，我们一起来学习另一个提供了很多功能的命令：`find`，该命令可以用来查找文件。该命令不仅查找当前目录中的文件，也会查找所有子目录中的文件。可以独立使用 `find` 命令，不过组合其他命令更是事半功倍，我们会在下一节介绍如何将多个命令组合起来使用。`find` 命令有不少有用的命令行选项，我在下面列出一些比较常用的。

- `-L`：该选项表示启用符号链接跟踪，所以可以知道 `find` 命令默认并不会跟踪符号链接。
- `-maxdepth <number>`：该选项指定 `find` 命令最大子目录查找深度，例如 `1` 表示当前目录文件和一级子目录文件。
- `-newer <file>`：该选项表示只查找比指定文件的修改时间更新的文件。
- `-empty`：该选项表示只查找空文件。
- `-atime <number>`：该选项表示只查找距离上次访问指定天数之后的文件。

- **-name <filename>**: 该选项表示搜索完全匹配指定文件名的文件。
- **exec <command>**: 该选项指示 **find** 命令为每个匹配的文件路径执行指定的命令。该选项真正体现了 **find** 命令的强大, 比如你可以指定删除命令来清理不需要的旧文件。不过执行命令前请先在几个文件上尝试一下, 否则一条错误的命令可能就造成几百个文件的错误。

例如结合这几个选项, 你可以执行如下的命令:

```
$ find /mnt/Volume1 -empty -name fooBar -exec rm
```

该命令将会搜索/mnt/Volume1 中的所有名称匹配 fooBar 的空文件, 并执行 **rm** 命令将其删除。所以清理文件系统时, **find** 命令会很有用。

下一个功能强大的命令是 **file**, **file** 命令的功能是通过读取文件内容判断文件的类型。当你想知道判断一个文件是文本文件还是应用程序时, **file** 可以告诉你。**file** 还可以用来检测压缩文件格式, 以确定应该使用哪个工具打开它。我们对/bin/bash 执行 **file** 命令, 会看到如下的显示:

```
$ file /bin/bash
/bin/bash: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses :
shared libs),for GNU/Linux 2.6.26, BuildID[sha1]=0xe370a0be978f49569f115c9f41bc54688be9
fe32, stripped
```

可以看出/bin/bash 是一个针对 ARM 平台编译的 ELF 格式 32 位 LSB 可执行文件, 所以它可以在树莓派上运行, 可以使用类似的命令确定互联网上下载的应用程序是否是针对树莓派的 ARM 平台。

本节最后一个介绍的命令就是著名的 **grep**, **grep** 是 Generalized Regular Expression Parser (通用正则表达式解析器) 的缩写, 它的主要作用是模式匹配。就像 **less** 和 **find** 命令, **grep** 无论单独使用还是结合其他命令一起使用, 都很强大。你可以使用 **grep** 搜索文件内容中的指定字符串, 例如:

```
grep <pattern> <file>
```

grep 所有的参数都支持通配符, 所以你可以搜索整个目录中的所有文件内容, 例如我们想在 /etc/init.d 目录下的所有文件中搜索 **find** 字符串, 可以输入如下命令:

```
$ grep find /etc./init.d/*
```

你将会得到如下的输出:

```
/etc./init.d/checkfs.sh:# Include /usr/bin in path to find on_ac_power if /usr/ is on the root
/etc./init.d/checkroot.sh:# Include /usr/bin in path to find on_ac_power if /usr/ is on the root
/etc./init.d/kbd:          LIST_CONSOLES='cd /etc./init; find -name 'tty*.conf' -printf '%f '
| sed -e 's/[^0-9]//g'
/etc./init.d/kbd:          # eventually find an associated SFM
/etc./init.d/nfs-common:# particularly heavy daemon, so we auto-enable it if we find an
/etc./exports
/etc./init.d/rc:# Now find out what the current and what the previous runlevel are.
/etc./init.d/sudo:          find /var/lib/sudo -exec touch -t 198501010000 '{} ' \;
```

`grep` 命令还提供了一些有用的命令行选项：`-i`，用于关闭大小写敏感；`-v`，使 `grep` 显示所有不匹配的内容，而不是显示匹配的内容。你可以使用 `grep` 获得各种信息，在管理系统时，它是一个非常有用的命令。

组合使用多个命令

本节我们将介绍命令行的真正强大之处，你可以将第一个命令的输出作为第二个命令的输入，组合多个命令的功能。我们还会介绍一些例子，展示如何充分利用各种命令。不过，第一个问题就是如何组合多个命令呢？其实很简单，使用管道符号（`|`）即可。

例如我们想查找一些内容，但是输出可能会很多，我们希望能够上下滚动或者翻页，那么可以使用 `grep` 命令，加上管道符号，再加上 `less` 命令。执行命令后，你会发现最终进入了 `less` 命令，显示的内容是 `grep` 的输出。再例如，想在某个目录中搜索文件名，我们可以用下面这种方法组合 `ls` 和 `grep` 命令。

```
$ ls <directory> | grep <search string>
```

组合 ps 和 grep

我们一起来看一个例子，如何更有效地管理进程。你可以组合使用 `ps -ef` 和 `grep` 来搜索指定的应用，例如搜索 SSH 应用：`sshd`，你可以这样输入命令：

```
$ ps -ef | grep sshd
```

你应该会看到类似的输出：

```
root      1722      1  0 19:24 ?        00:00:00 /usr/sbin/sshd
root      1956    1722  0 19:43 ?        00:00:00 sshd: pi [priv]
pi        1963    1956  0 19:43 ?        00:00:01 sshd: pi@pts/0
pi        2030    1964  0 20:33 pts/0    00:00:00 grep --color=auto sshd
```

过滤 grep 自身

我们看到输出中出现了 `grep` 自身，没关系，我们可以再次使用 `grep` 配合 `-v` 参数过滤掉 `grep`，命令变成了这样：

```
$ ps -ef | grep sshd | grep -v grep
```

输出变成了这样：

```
root      1722      1  0 19:24 ?        00:00:00 /usr/sbin/sshd
root      1956    1722  0 19:43 ?        00:00:00 sshd: pi [priv]
pi        1963    1956  0 19:43 ?        00:00:01 sshd: pi@pts/0
```

分隔结果

上一条命令的输出看起来准确多了，不过我们还可以做得更多。下面我们将使用 `awk` 命令，

`awk` 命令用于从一行文本中提取信息。这样我们就可以从命令输出中获得指定的信息，忽略不需要的内容。默认情况下 `awk` 命令自动按照空白字符对文本进行分隔，所谓空白字符是指空格和制表符（`tab`），你也可以使用 `-F <delimiter>` 命令行选项自定义分隔符。分隔完成后，你可以指定如何输出信息，`awk` 命令使用一套强大而复杂的语法来定义输出格式。

不过大部分情况下，你只需要指定输出位置即可。`awk` 使用 `{}` 来定义命令和输出格式，不过输入时请使用单引号包裹参数，避免命令行错误的理解语法，命令的格式如下：

```
$ awk '{ <command> }'
```

最常用的 `awk` 命令应该就是 `print`，很简单，它可以输出指定位置的内容。每个被分隔出来的字符串都通过固定格式的数字来表示位置，例如 `$1` 表示第一个，`$2` 表示第二个，以此类推。我们把 `awk` 命令和之前的 `ps -ef` 命令结合起来，并且只输出在第二个位置上的 `PID`。

```
$ ps -ef | grep ssh | grep -v grep | awk '{print $2}'
```

我们看看输出变成了什么样。

```
1722
1956
1963
```

现在只有 `PID` 了，我们很想把这些程序都关掉，该是使用 `kill` 的时候了。所以，我们结合了 `ps`、`grep` 和 `awk` 来获得指定应用的 `PID`，并最终结合 `kill` 将它们杀掉。

使用 `xargs` 对每行结果执行命令

由于 `kill` 命令并不能接受 `PID` 列表作为参数，我们需要想办法将 `PID` 一个个传递给 `kill`。这里我们可以利用 `xargs` 命令，该命令的作用类似于 `find` 中的 `-exec`，它可以将输入的每一行作为执行指定命令的参数。所以我们将 `xargs` 和之前的所有命令组合在一起，得到如下的命令：

```
$ ps -ef | grep ssh | grep -v grep | awk '{print $2}' | xargs kill -1
```

当然我不准备按回车键杀掉 `sshd` 进程，不过这已经展示了如何在一行输入中组合多种命令，发挥各种命令的强大能力。

还有很多命令可以成为你的 `Shell` “功夫”的一部分，例如 `wc` 命令（`word count` 的缩写）。默认情况下，`wc` 将会统计输入的空行、单词数量和字节数量。它也有很多有用的命令行选项，`-l` 表示只统计文本行数，`-w` 表示只统计字符数，`-m` 表示只统计单词数量。

还有 `sort` 命令，该命令对输入的每一行进行排序并输出，`sort` 命令经常和 `uniq` 命令（`unique` 的缩写）结合使用。你可以通过管道将 `sort` 的输入导入 `uniq`，得到不重复的排序输出结果。

本节最后我们介绍另一个很有用的符号：`>`，该符号和管道符号属于同一家族，它可以将原本输出至屏幕或者命令行终端的内容输出到指定的文件中，需要保存命令输出时会很有用。好了到这里，我们已经学习了如何在 `Shell` 中使用管道符号组合多个命令，这就是命令行真正的强大之处，你可充分利用这个功能来完成你的工作。

用户环境

这一节我们介绍一下一直在使用的用户环境，每次当你登录到树莓派时，用户环境都会被建立起来。系统通过执行一系列的文件和命令为你建立这个环境，其中最重要的文件之一就是 `/etc/profile`，该文件为所有的登录用户预设了 Shell 和各种环境变量。

`profile` 文件预设了很多环境变量，包括 `PATH`，该环境变量是 Shell 工作时最重要的环境变量之一。

`PATH` 作为最重要的环境变量，它告诉系统应该去哪些路径中寻找应用程序。例如前几节中，我们执行命令时都只输入了命令的名称，而不是该命令对应的应用程序文件路径。这是因为 `PATH` 环境变量包含了这些路径，Shell 会自动搜索其中的文件，并执行名称匹配的应用程序，因此你才可以不输入完整的文件路径。不过这也带来一个问题，就是执行命令时你不知道真正运行的是哪个应用程序，或者当你的系统上有多个版本的应用时，你无法确定运行的是哪个版本。为了解决这个问题，我们可以使用 `which` 命令。该命令会输出指定命令名称对应的应用程序路径。例如我们想知道 `ssh` 应用程序的路径，我们在命令行中输入：

```
$ which ssh
/usr/bin/ssh
```

该命令使用起来很简单吧。`PATH` 环境变量为各种程序的运行提供了重要的路径信息，它主要就是由之前介绍的 `/etc/profile` 文件设置的。我们试着修改 `PATH` 环境变量之前，最好还是先看看已经有什么了。有两种方法可以输出 `PATH` 环境变量的内容，第一种是使用 `echo` 命令直接打印指定的环境变量，例如我们以 `$PATH` 作为 `echo` 的参数执行命令，就能看到 `PATH` 环境变量中设置的路径。

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games
```

这就是我的树莓派中的 `PATH` 环境变量内容，其中每个目录路径之间使用冒号分隔。第二种方法是使用 `env` 命令，它会显示出所有的环境变量，当然 `PATH` 也包括在内。该命令在检查系统状态时也很有用，执行 `env` 后，你应该会看到类似输出。

```
$ env
TERM=xterm
SHELL=/bin/bash
XDG_SESSION_COOKIE=eb95d80869be1ad62af36ec5502c41a1-1349772229.897557-1575821964
SSH_CLIENT=10.0.0.104 3643 22
SSH_TTY=/dev/pts/0
USER=pi
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.lzm=01;31:*.tlz=01;31:*.txz=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lz=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.

```

```

mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.
flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.
yuv=01;35:*.cgm=01;35:*.emf=01;35:*.axv=01;35:*.anx=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.
au=00;36:*.flac=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.
ra=00;36:*.wav=00;36:*.axa=00;36:*.oga=00;36:*.spx=00;36:*.xspf=00;36:
ALL_PROXY=SOCKS://localhost:9050
MAIL=/var/mail/pi
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games
PWD=/home/pi
LANG=en_US.UTF-8
SHLVL=1
HOME=/home/pi
LOGNAME=pi
SSH_CONNECTION=10.0.0.104 3643 10.0.0.56 22
_=/usr/bin/env

```

这里输出了很多内容，包括表示我们当前所使用 Shell 的 SHELL 环境变量，控制 ls 输出颜色的 LS_COLOR 环境变量，表示当前目录的 PWD 环境变量，表示当前主目录的 HOME 环境变量，表示当前用户名的 USER 环境变量以及 PATH 环境变量，这些环境变量对于了解系统的运行状态都很有用。

我们已经知道怎么查看 PATH 环境变量了，下面我们来尝试修改它。我们将会使用 export 命令修改指定的环境变量，该命令的语法如下所示。

```
$ export VARIABLE="SOMETHING"
```

如果你直接按照上面的写法输入了命令行，那么你再运行一下 env 命令，在所有的环境变量中都会看到下面这一行输出。

```
VARIABLE=SOMETHING
```

如果我们直接这样修改 PATH 环境变量，则结果是重置了它，而我们真正想做的其实是在已有的 PATH 环境变量后面增加一些内容，而不是整个重写。最简单的增加路径的方式是在设置环境变量时引用其自身内容。例如我们想将 /opt 目录加入 PATH，首先还是按照格式写上 export PATH=，之后引用当前的 PATH 环境变量内容，我们写上 \$PATH，使用冒号作为分隔，最后写上 /opt，最终我们得到如下命令：

```
$ export PATH="$PATH:/opt"
```

执行该命令后，PATH 环境变量就变成了：

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games:/usr/games:/opt
```

这样 /opt 路径就被添加到 PATH 环境变量最后了。你会发现，其实 PATH 环境变量中的目录路径是有顺序的，其中靠前的路径（左边的）将更早地被搜索，如果存在匹配的命令就不再搜索后面的目录了，因此我们设置的 /opt 目录是最后被搜索的。

目前我们介绍了什么是环境变量，如何调整它们，PATH 环境变量的用法等，我们下面介绍用户主目录中的几个特殊文件。这些文件是用户的 profile 设置文件，它们实际上是存在于主目录中的几个隐藏文件，文件名均以 rc 结尾。当你登录到 Shell 或者有应用程序首次开始运行时，这

些文件有可能会被执行。如果你使用的是 Linux BASH Shell, 那么对应的就是 `.bashrc` 文件。该文件会设置一些环境变量, 也会为你的命令行设置显示颜色。例如对于 `ls` 命令, 该命令对应的文件一般位于 `/bin/ls`, 如果你运行完整路径的 `/bin/ls` 而不是 `ls`, 你不会看到不同的颜色, 这是因为 `.bashrc` 通过 `alias` 命令为 `ls` 加上了颜色。

那么什么是 `alias` 命令? `alias` 可以为一个或者一组命令定义一个别名。例如在 `.bashrc` 文件中, `ls` 其实是以下命令的别名。

```
$ ls --color=auto
```

所以每次当你执行 `ls` 命令时, 命令行自动为标准的 `ls` 命令设置了颜色。在 `.bashrc` 中是通过类似以下的方法来定义的。

```
alias ls='ls --color=auto'
```

你可以自定义 `.bashrc` 的内容, 因为这是你的本地系统环境变量, 通过修改 `.bashrc`, 你可以覆盖 `/etc/profile` 中的设置, 还可以增加其他环境变量。

cron 命令

最后一个我们将要介绍的强大系统管理命令是 `cron`。`cron` 可以根据一个固定格式的控制文件定时自动执行命令。该控制文件被称为 `crontab`, 每隔一分钟 `cron` 就会去读取 `crontab` 文件, 检查是否有命令需要执行。如果想看一下当前的 `crontab` 内容, 可以执行以下命令。

```
$ crontab -l
```

不过当前 `crontab` 应该没有什么内容, 所以你会看到下面的输出。

```
no crontab for pi
```

让我们在 `crontab` 中增加一个基本的定时执行命令, 使用以下的命令来编辑 `crontab` 文件。

```
$ crontab -e
```

该命令执行后, 系统会自动创建一个 `crontab` 文件, 并使用文本编辑器打开它, 文件中已经填写了一些说明和一个例子, 帮助你编辑命令。

```
0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
```

该例子将会在每周一凌晨 5 点执行 `tar` 命令压缩备份所有用户目录中的内容, 我们之前介绍过 `tar` 会将所有文件打包成一个存档文件, `zip` 命令将会压缩该存档文件。这行命令开始的 5 个数字按照如下的定义:

- 分钟
- 小时 (24 小时制)
- 日期
- 月份
- 星期 (0~6, 0 表示星期日, 6 表示星期六)

按照此定义,该行内容表示的是 `tar` 命令将会在每年每个月的每个周一凌晨 5 点 0 分被执行。你可以自己写一些自动执行的系统管理命令,而且你可以使用我们之前介绍过的所有命令。比如可以在特定的时间杀死某个进程,那么你可以组合 `ps`、`grep`、`awk`、`xargs` 和 `kill` 命令。到这里,加上刚刚介绍的 `cron` 命令,你的系统已经完全在你的掌握之中了。

本章小结

祝贺你,读到这里你已经能够管理你的系统了。你可以任意组合这 20 多个命令,按照你的需求执行工作。而且通过使用 `cron` 命令和编辑 `crontab` 文件,你也能够让这些命令定时自动运行。这些工具都有强大的命令行选项,并且能够自动执行,可以帮助你更好地管理 Linux 系统。

在命令行中编辑文件

不管你信不信，使用 Linux 时大部分时间你可能都是在编辑文本文件。文本文件无处不在，包括简单的文本存储、软件的源代码、配置文件和启动脚本都是文本文件。文本文件在 Linux 中得到大量使用是因为文本文件操作方便，读取内容容易。不像 Windows 操作系统，Linux 系统中并不存在注册表，所以应用程序都使用文本文件来存储自己的配置，连复杂的 Sendmail¹ 配置文件（这个配置文件必须由另一个应用程序的配置文件生成）也是纯文本文件。

本章我们将简要介绍文本文件的使用，并讲解如何使用两款文本编辑器（nano 和 Vim）来编辑文本文件。

6.1 什么是文本文件

计算机在文件中存储数据时，有两种方式可以选择，文本文件或者二进制文件。等等，计算机不是都以二进制存储数据的么（0 或 1）？那么文本文件和二进制文件有什么不同呢？的确，文本文件最终还是以二进制的形式存储的，不过文本文件和二进制文件在读取时还是有明显的区别。

就像你能阅读本书中的内容，那是因为所有的内容都是以文字的形式印刷在纸上的。页面中的文字进行了合理的排版，让阅读变得更容易。我们在每段首行进行缩进，使用句号分隔句子。这就是应用于人类阅读的一种协议，我们将文字写下来时遵守了这种协议，你在阅读时也按照这种协议，所以能够理解书中的内容。阅读时，我们需要同时理解文字的结构（数据是如何被存储和读取的）和文字的语义（每个文字的意思），如果不能同时满足这两个条件，我们就不能理解句子的意思，例如：

Goedemorgen. Hoe gaat het?

这段话是什么意思？我们能确定这不是中文（也不是英文），不过如果你懂荷兰语，那你应该能读懂这段话的意思。不过，你会发现即使你不懂荷兰语，你也能够大概看出这段话的结构，通过句号和问号可以知道其中有两个句子，一个是陈述句，另一个是疑问句。更进一步，我们还能猜测

¹ 译者注：一种运行于 Linux 上的通用电子邮件传输服务程序。

一下这段话的意思。按照英语的读法，第一个句子似乎有点像“Good morning”（Goedemorgen 的确就是早上好的意思），第二句看起来很像“How are you?”。

虽然不懂荷兰语，但是我们能够猜测出句子意思，这是因为荷兰语和英语都有类似的句式（结构），这两种语言也拥有相同的字母（虽然有一些差别）和标点符号。虽然可能读音不对，但是相同的语法结构让我们能够多少理解一些。那么图 6-1 中¹的句子你能理解么？

Доброе утро. Как дела?

图 6-1 俄语

大部分读者估计都不认识图 6-1 中的文字，其实这仍然是之前用荷兰语写的那两句话，只不过这次使用了俄语。但这次仅仅从文字或者字形，我们无法识别这两个句子的意思，也完全没有可能按照英语的读法将其读出来。

这里用外语来举例子是为了说明文本文件就像英语和荷兰语一样，虽然要完全理解内容必须懂得相应的语言（理解语义），但是并不影响我们大概浏览一下（理解结构）。对于不同应用程序的配置文件，例如 Apache Web 服务器和 Sendmail 的，虽然 Sendmail 不能使用 Apache 的配置文件，但是它仍然可以打开并读取配置文件内容。

而俄语就像是二进制文件，对于不是同一个语系发展而来母语国家的人来说，是完全无法理解的，也很难看出大概的结构。在计算机的世界里，二进制文件就是这样，除了创建它的应用程序以外，其他应用程序完全无法读取其中的二进制数据的意义。

也许你觉得使用什么形式来存储数据似乎不是什么大问题，毕竟文件一般都是由创建它的应用程序来读取的。但是万一程序崩溃了，或者你需要在其他计算机上打开某种二进制文件，你可能就需要厂商提供的专用的程序才能做到，没有通用的方案。

但文本文件就没有这些问题，你可以使用任何文本编辑器打开它，不需要使用厂商专门提供的工具。在 Linux 系统中，几乎所有的应用程序都使用文本文件作为配置文件，你可以用任何你喜欢的文本编辑器打开并编辑它们。所以在 Linux 的世界里，只需要熟练地使用一种文本编辑器，你就可以修改任何应用程序的配置文件，修改任何脚本的源代码，甚至是写一本书（我们就是这么做的）。

6.2 文本编辑器的竞争

文本编辑器就像 Linux 世界中的瑞士军刀，在 Linux 的日常使用中至关重要。不过就像瑞士军刀有很多品牌，Linux 中也有很多的文本编辑器可供选择，功能各有不同。有时你可能只需要一个基本的文本编辑器，可以打开、编辑并保存文件；有时你可能需要一个全功能的文本编辑器，执行复杂的搜索。你当然可以使用一个全功能编辑器执行简单的操作，不过你完全可以使用一个

¹ 译者注：原书中使用了中文“早晨好。你怎么样？”，本书中文版这里替换为俄语，以符合原文讲解的意思。

轻量级的编辑器更迅速地完成工作。所以很多人都会使用两种不同的编辑器，一个是轻量级的，另一个是全功能的，以满足不同的需要。本章我们也会相应地介绍两种编辑器，轻量级的 nano 和复杂的 Vim。

我们第一个介绍的文本编辑器是 nano，它是一种轻量级、易使用的文本编辑器，并且预装在几乎所有的 Linux 系统中。最新版的 nano 功能更加强大（例如源代码语法加亮），不过使用起来仍然轻巧简单，你可以在所有 Linux 系统上使用 nano，当然也包括树莓派。

nano 文本编辑器也有一段历史，它是作为 Pine（一种文本 E-mail 客户端）中集成的 pico 编辑器替代者出现的。在一些旧版本的系统中你也许还能发现 pico 的身影，如果计算机上没有 nano，那应该会有 pico，nano 和 pico 有着完全相同的命令和操作方法。总的来说，这两个几乎完全相同的编辑器可用于几乎所有的 UNIX 和 Linux 系统。

nano 编辑器的最大特点就是使用简单，不过这并不代表它功能不强大，你可以通过菜单调用各种功能。nano 足以胜任编辑配置文件这种简单的任务，不过有时你需要更多的功能，这时就该 Vim 登场了。Vim 除了可以做到 nano 的所有功能，它还有更多的高级功能，例如语法高亮、自定义配色方案、复制粘贴、删除文本段、整块文本缩进以及同时打开多个文件等。

读到这里你也许会想 Vim 的确更好，所以学会使用 Vim 就能搞定一切，何必再学另一个完全不同的文本编辑器呢。这么想也没错，而且 Vim（或者其他操作完全相同的变种程序，例如 vi）也预装在几乎所有的 Linux 系统上。不过我们之所以推荐你学习 nano 编辑器，是因为你可以在几分钟内学会使用 nano 编辑器。也许 Vim 的用户不赞同我们的观点，但必须承认学习 Vim 的确比较困难。所以只学习 Vim 没有什么问题，不过想真正地使用它需要花不少时间，nano 则可以在这段时间内帮助你完成文本编辑等简单操作。

6.3 使用 nano

我们从 nano 这款简单的编辑器开始，它应该能够满足你对文本编辑的基本要求。我们可以在命令行输入 nano 命令打开 nano 编辑器，可以仅仅输入 nano，nano 编辑器将会新建一个未命名的空白文件。不过使用 nano 的目的不就是编辑文件么，所以在命令行中指定文件名，直接让 nano 打开它更加方便。比如我们想打开并编辑 test.txt 文件，我们可以输入以下命令：

```
$ nano test.txt
```

按回车键后，你将会看到如图 6-2 所示的界面。

nano 编辑器是一个命令行应用程序，所以无论你通过何种方式连接到树莓派，你都会看到类似的界面。命令行应用程序都有一个特点，它们的界面会受到终端显示区域大小的影响。例如，当你的终端是通过 SSH 连接的或者是虚拟终端时，你可以调整终端界面的大小，这种情况下 nano 编辑器可以自适应你的调整。

nano 的界面

nano 编辑器的界面看起来似乎有点奇怪（尤其是底部的区域），这样设计的好处在于当你编

辑文字时，不会被界面元素打扰。我们使用 nano 编辑本书时，几乎就没有注意到左上角的版本号，直到写到这里，我们才发现所使用的是“GNU nano 2.0.9”。

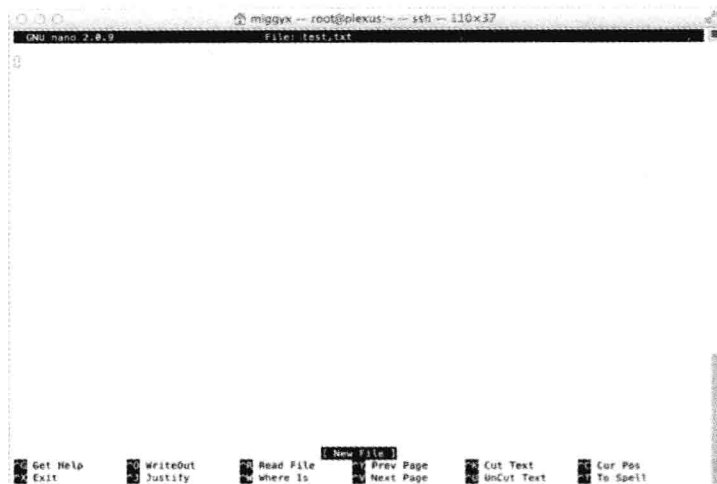


图 6-2 使用 nano 打开一个空白文件

编辑器顶部的标题栏还显示了文件名，例如我们的例子中显示的是 test.txt，当你同时打开多个终端窗口编辑文件时，文件名能保证你不会编辑错。标题栏最右边会显示文件是否已经被修改，或者说当前编辑的内容还未保存。这个可以提醒你不要忘记保存文件，不要让你的工作白费。例如我们按一下空格键改变当前的文件，你将会在屏幕右上角看到“Modified”，如图 6-3 所示。



图 6-3 nano 编辑器显示文件被修改

nano 的标题栏很简单，主要的功能区都集中于底部。第一部分是状态栏，例如图 6-2 中显示 “[New File]” 表明当前是一个新建的文件。第二部分是快捷键提示栏，这里显示各种功能和对应的按键组合，常用的操作都可以在这里找到。让我们来仔细看一下快捷键提示栏，如图 6-4 所示。



图 6-4 快捷键提示栏

如果你使用的是 Windows 或者 Mac 系统，应该已经习惯于使用鼠标进行拷贝粘贴操作，而不是使用键盘组合键。对于复制操作，在 Windows 系统中，可以使用 **Ctrl** + **C** 组合键，在 Mac

系统中，可以使用 **Command** + **C**，这些按键组合可以替代鼠标点击。Nano 的操作和这些组合键很类似，不过不仅仅是复制粘贴，所有的操作都通过键盘组合实现。因为要记住这么多操作的组合键很困难，所以 nano 将它们显示在快捷键提示栏，虽然只显示常用的操作，但是这些已经足以满足你的日常需求。

快捷键提示栏中，使用 ^ 表示控制键（Windows 和 Linux 系统的 **Ctrl**、Mac 系统的 **Command**）。例如我们想打开 nano 的帮助，提示栏中显示按 ^G，我们按住 **Ctrl**，并按下 **G** 即可。

保存文件

这一节我们介绍如何在 nano 中保存文件，不过只保存一个空白文件没什么意思，也不能确定保存是否成功了，所以我们在文件中输入一些“Mary Had a Little Lamb”¹的内容。

不像其他的一些编辑器，nano 并不需要切换模式（Vim 有模式的概念，我们后面会介绍），当你在 nano 中进行任何输入，都会被理解为是正在输入文件的内容，当然也包括直接用回车键换行，输入一些内容后，我们得到如图 6-5 所示的文本内容。

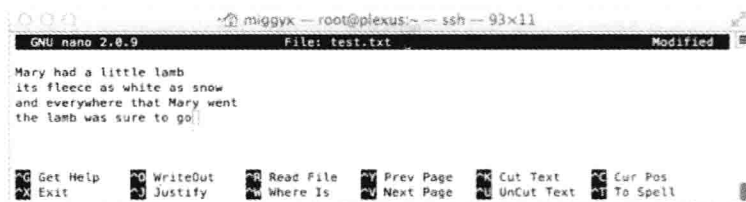


图 6-5 在 nano 中输入 Mary Had a Little Lamb

Nano 编辑器的窗口被我们缩小了一些，你会发现 nano 很好地适应了小窗口，并且会在你调整时自动变化。

此时原来提示“[New File]”的状态栏消失了，而右上角的标题栏 **Modified** 标记仍然在提醒我们的输入还未被写入磁盘。为了保证刚才的输入不会白费，让我们来一起看看快捷键提示栏。最常使用的就是 **WriteOut**（也就是保存文件）和 **Exit** 命令，其中 **Exit** 命令会在退出时询问你是否需要将修改保存到文件。我们先使用 **WriteOut** 命令，具体的操作方法是按住控制键（**Ctrl** 或 **Command**），同时按 **O**，nano 此时会显示如图 6-6 所示的提示。

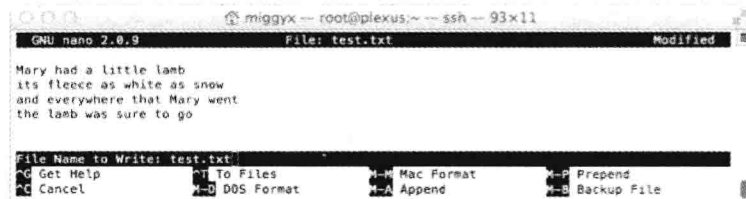


图 6-6 使用 WriteOut 命令

¹ 19 世纪美国的英语童谣“玛丽有只小绵羊”。

从图中可以看到两点改变。

- 状态栏再次出现，此时它向我们确认是否将文件写入名为 `test.txt` 的文件，这正是我们启动 `nano` 时提供的文件名，所以我们将按回车键确认并保存文件，不过按回车键之前我们先看一下快捷键提示栏的变化。
- 快捷键提示栏显示了一组全新的命令，这是因为它会随着命令上下文变化，例如当前处于保存文件状态，它显示的就是所有和保存文件相关的命令，其中比较有用的就是 `Cancel` 命令，可以取消当前的保存操作。

你也可以在保存时修改文件名，并且保存后 `nano` 将会自动打开新文件名的文件。如图 6-7 所示，我们看到 `nano` 的状态栏在保存成功后提示我们实际保存了多少内容。

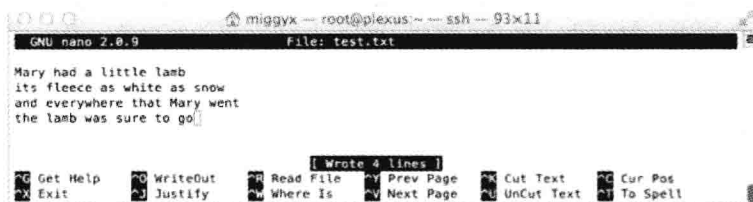


图 6-7 修改已经写入文件

还可以看到快捷键提示栏恢复了初始的状态，状态栏提示写入了 4 行内容，所以可以确认文件已经被保存。

如果修改了文件之后，你试图直接退出 `nano` 会发生什么呢？如图 6-8 所示。

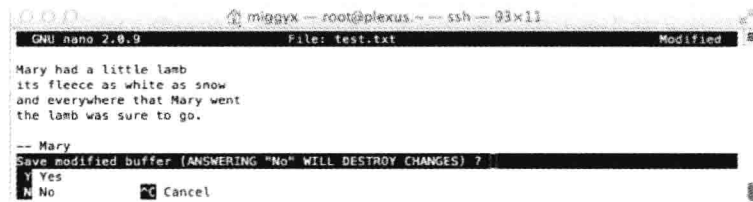


图 6-8 退出时提示保存

我们稍微修改了内容，并按控制键和 `X` 准备直接退出 `nano`。此时状态栏提示是否需要保存文件的修改，并在快捷键提示栏提供了几个选项。如果我们选择 `Yes`，`nano` 将会保存修改并退出；如果选择 `No`，`nano` 将会丢弃所做的修改并直接退出；或者我们还可以按下控制键和 `C` 选择 `Cancel`（取消），这样 `nano` 将会回到编辑状态。

更多功能

在 `nano` 中移动光标很简单，使用键盘的方向键即可。有些编辑器的作者觉得方向键离字母键太远，使用其他按键移动光标，`nano` 并没有如此设计。移动光标太简单了，所以我们本节主

要介绍的是两个关于搜索的功能。

搜索功能在 nano 编辑器中被称为 Where Is，快捷键提示栏提示我们按控制键和 **W** 启动该功能，如图 6-9 所示。

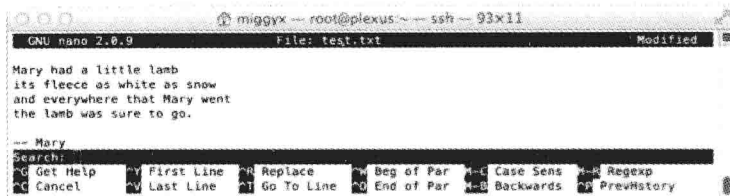


图 6-9 搜索

我们看到快捷键提示栏发生了变化，提供了很多与搜索相关的功能，我们先从最简单的搜索文字开始，输入“Mary”并按回车键，如图 6-10 所示。

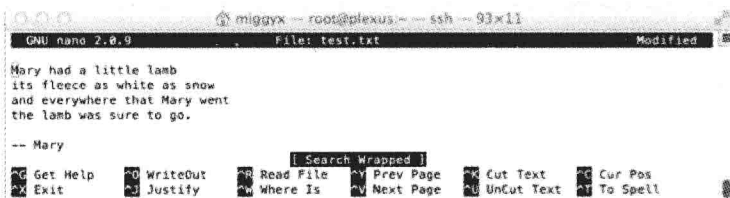


图 6-10 Nano 搜索“Mary”

Nano 在状态栏显示“Search Wrapped”（循环搜索），是因为搜索时我们的光标处于文件末尾，nano 越过末尾，并从文件头部开始搜索。如果你想搜索下一个匹配的内容，可以再次按控制键和 **W** 键并直接按回车键，nano 将继续搜索上次指定的内容。

除了搜索文字，我们有的时候想直接跳至指定行，例如你的某个脚本运行出错了，一般都会提示错误行号，此时就需要直接显示该行内容，以尽快解决错误。我们仍然使用控制键和 **W** 进入 Where Is 命令，不过我们不输入文字，而是根据提示按控制键和 **T** 启用 Go To Line 功能，此时 nano 的状态如图 6-11 所示。

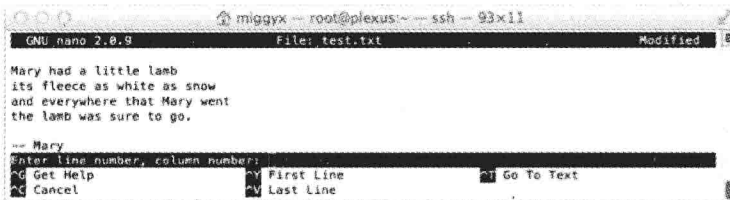


图 6-11 显示指定行

此时你可以输入行号并按回车键，nano 就会跳至指定行。快捷键提示栏还提示可以通过快捷键直接跳至文件头（First Line）或者文件尾（Last Line），或者再次按控制键和 **T** 回到文字搜

索 (Go To Text)，也可以按控制键和 **C** 取消当前搜索 (Cancel)。

Nano 使用小结

我们介绍了 nano 的基本使用和一些简单的功能，你会发现其实这些功能已经可以满足日常的使用了。nano 功能简单，操作也很简单，能够快速完成你的工作。不过，我们下一节将要向你介绍一位重量级选手 Vim，它能提供更多更强大的功能。

6.4 使用 Vim

Vim 比 nano 复杂不少，nano 是一款简单的文本编辑器，而 Vim 则是一款功能丰富的编辑器，甚至远远地超越了编辑器的范畴，你甚至可以让 Vim 和其他应用程序集成起来一起工作（例如 shell 或者源代码管理），所以 Vim 也被称之为编辑器之神¹。

当然 Vim 也有一个明显的缺点，就是学习困难。不像 nano 可以即开即用，对于 Vim 你需要首先学习一下模式的概念。使用 Vim 时，主要会使用的是普通模式 (normal mode) 和插入模式 (insert mode)，这两个模式也是我们将会详细介绍的。

虽然我们日常使用 Vim 编辑器，但是也远远不敢自称专家。我们熟悉一些工作时需要使用的功能，但是也有大量的功能不会使用，甚至都没有听说过。所以本节将只是 Vim 的入门介绍，如果你觉得需要深入学习一下，有很多书籍和视频专门介绍如何使用 Vim，你可以进一步地阅读和了解。

首先在我们能输入 “Mary Had a Little Lamb” 内容之前，我们需要安装 Vim，使用以下命令执行安装。

```
$ sudo apt-get install vim
```

命令自动完成后，我们就装好了 Vim。让我们来运行它。

```
$ vim test.txt
```

图 6-12 所示的就是 Vim 启动后的界面，和 nano 一样简洁。



图 6-12 启动 Vim

没有标题栏、状态栏和快捷键提示栏，Vim 界面非常简单。让我们来看看从界面上能看出什么，在屏幕左下角能看到当前编辑的文件名，旁边显示 6L 和 115C，表示文件有 6 行内容，115

¹ 译者注：还有一款编辑器名为 Emacs，被称为神之编辑器。

个字符。右下角则显示 1,1，表示当前光标处于第一行第一个字符处。由于我们刚刚打开文件，所以光标当然处于文件的开头。

注意 不同的 Vim 版本在刚刚打开文件时的行为有所不同，有些版本会直接将光标定位于上次关闭时的位置，这也是一个很方便的功能。

在文件内容后面会看到一些~符号，这是用来表示这些空白并非文件的内容，仅仅是屏幕的显示空白，这个提示在编程时会非常有用，可以知道文件尾部是否存在多余的内容。

Vim 中的模式

初学者最常见的问题是，因为发现在 Vim 中无法像在其他文字处理软件和文本编辑器中那样直接进行文字输入而倍感郁闷。出现这样问题的原因在于 Vim 存在着不同的模式和设定，当你打开 Vim 时，它处于命令模式，只进行命令的处理而不进行文本编辑。与 nano 通过组合键执行命令的方式不同，Vim 使用字母组合来执行命令。不出意外的是，这种方式导致 Vim 的学习曲线比较陡峭，需要不断的练习。可是一旦你学会了 Vim 神奇的键绑定方式，你就再也离不开它了。

首先，让我们先离开命令模式尝试输入点什么吧。按下 **␣**，Vim 就会切换到插入模式，在这个模式下我们就可以像在 nano 中一样进行文本编辑了。如图 6-13 所示，处于插入模式下时，屏幕的左下角会显示 “—INSERT—”。

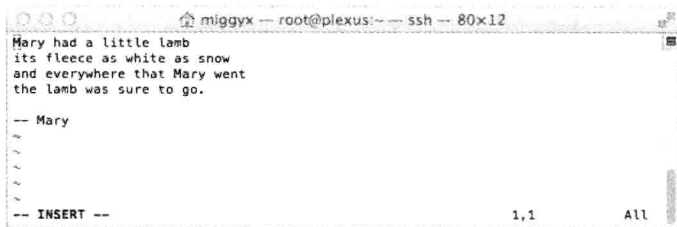


图 6-13 Vim 插入模式

这时你可以像在 nano 中一样使用方向键移动光标、添加或删除文本。但我们该如何保存我们进行的修改呢？

保存修改

首先我们需要按 **ESC** 回到命令模式，这样才能进行命令处理和文档保存。保存文件之前，我们不妨将例子中的落款 Mary 改成 John，造成一点实质性的修改。我们可以使用下面的命令来保存文件。

```
:w
```

冒号后跟一个小写的 “w”。Vim 是大小写敏感的，如果你输入的是 “:W”，将会看到图 6-14

所示的错误提示。

命令无误时，提示如图 6-15 所示。

如果你想将文件另存为一个新的文件，在“:w”命令后面加上文件名就可以了。

```
:w test2.txt
```

这样就可以将当前文件另存为一个新文件 test2.txt，并打开新文件以供继续编辑。

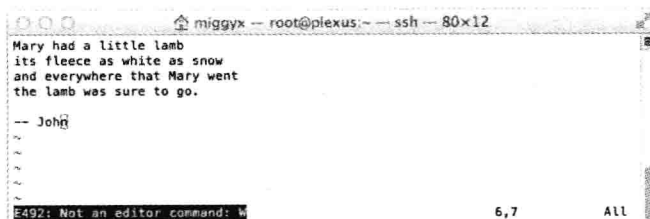


图 6-14 误用了大写“W”



图 6-15 “:w” 正常运行

退出 Vim

我们已经了解了一些 Vim 最为基础的操作，接着我们来看看如何退出。

```
:q
```

如果你没有修改文件内容，Vim 会立刻退出。但是，如果是在修改文件后再执行上面的命令，Vim 可不会乖乖地退出，如图 6-16 所示。



图 6-16 退出未保存的文件需要强制执行

幸运的是，错误提示中已经给出了解决方法¹。

:q!

在 Vim 中进行搜索

这里我们来看下如何执行与 nano 类似的搜索功能。事实上 Vim 的搜索功能十分强大（可以从正则表达式开始了解，参见“正则表达式”）。

正则表达式

正则表达式充满了魔力，前提是你能够熟练地使用它。这是一把双刃剑，它能够帮助你如同庖丁解牛般解决各种复杂的文本处理任务，也可以成为系统管理员最深沉的噩梦。尽管如此，学习些正则表达式的基本知识总是有用的，在 Vim 中进行搜索、替换、批量修改时离不开它。

虽然在本书中不会展开讨论这部分内容，但是关于正则表达式，线上线下都有很多很棒的入门资料。Jan Goyvaerts 的网站（www.regular-expressions.info）上有各种有用的相关信息，Jan 还写了一个称为 RegxBuddy 的程序来帮助调试复杂的正则表达式。rubular.com/ 这个站点可以帮助我们方便地测试简单的正则表达式。

在 Vim 中进行搜索，只需要在命令模式（不确定现在处于什么模式的话，按两次 `Esc` 可以确保退到命令模式）下输入 `/` 符号，然后跟上需要搜索的字符串。

`/Mary`

按下回车键后就会发现文档中所有的“Mary”都被高亮了，如图 6-17 所示。

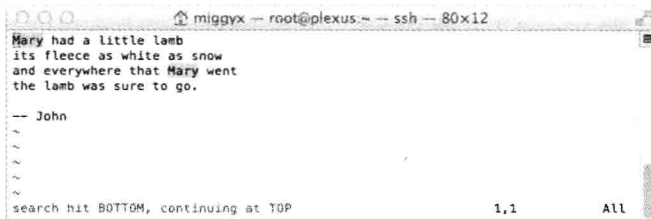


图 6-17 搜索“Mary”

可以看到提示信息“search hit BOTTOM, continuing at TOP（已搜索到文档底部，将从头开始搜索）”相当于 nano 中的“Search Wrapped（循环搜索）”。因为 Vim 的搜索功能支持正则表达式，所以它默认是大小写敏感的。这意味着如果我们搜索的是“mary”而非“Mary”，会得到图 6-18 所示的结果。

¹ 译者注：“:q!”表示不保存修改并退出，如果想要保存修改并退出可以使用“:wq”。

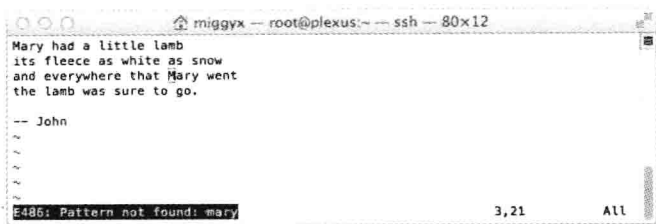


图 6-18 搜索 “mary”

在搜索重复内容方面 Vim 比 nano 更加方便，我们可以直接按 **N** 将光标移动到下一个搜索结果。在前面的例子中，使用 **/Mary** 查找 “Mary” 后，按 **N** 可以在各结果间切换，这是个很有趣的功能，不妨试试看。如果你需要从后向前切换结果，可以按 **Shift** + **N**。

跳转到特定行

我们已经学过如何在 nano 中跳转到特定行了，类似的操作在 Vim 中更为简单，这或许是因为对程序员来说这样的操作十分常用。我们可以在命令模式下输入 “:” 符号加上行号来进行跳转。比如，跳转到第 5 行，输入：

:5

按下回车键后，光标就会直接跳转到第 5 行。在编辑普通文本时或许用不上这样的功能，进行程序调试时显然就不同了。

Vim 中的光标移动

前面我们已经涉及了 Vim 中基本的光标移动方法，其实在 Vim 中还有一些进阶的方法可以使光标移动更为便捷。

在命令模式中，我们仍可以通过键盘方向键移动光标。而通过 **H** 和 **L** 这两个键可以进行更便捷的移动，按 **H** 后光标将回退一个字符，按 **L** 后光标将前进一个字符。进一步，我们可以使用 **B** 或 **W** 来以单词为单位移动光标。这样在内容非常多的行上移动比使用方向键更为高效。

再进一步，我们可以按 **0**（数字零）来跳转到当前行的开头，输入 “\$”（**Shift** + **A**）跳转到当前行的末尾。在做文本编辑时我们不时需要跳转到行首或行尾，现在我们可以一键跳转了。在表 6-1 中我们进行了小结。

表 6-1 Vim 常用导航快捷键

跳转位置	快捷键	跳转位置	快捷键
行首	0	前进一个字符	L
回退一个单词	B	前进一个单词	W
回退一个字符	H	行尾	Shift + A

现在我们已经可以在字里行间进退自如、随心所欲地移动光标了。掌握这些快捷键后，等待光标如同蜗牛般爬行的日子一去不返。然而，这还只是开始。Vim 中快捷键能做的远不止移动光

标这么简单。

Vim 中的删除

在命令模式中我们可以直接按下 **X** 删除当前光标所在的字符。这在修正一些排版上的小错误时十分有用，不过显然它不能代替插入模式下的一些操作。不过使用 **D** 可以与其他快捷键组合实现各种复杂的删除操作。

我们首先尝试下“dd”这组快捷键。让我们来试着删除文件中的第1行。通过下面的命令可以快速返回第一行。

```
:1
```

现在我们来删除当前行。

```
dd
```

执行后文档看起来应该和图 6-19 所示一致。



图 6-19 删除首行结果

那么万一误删了怎么办呢？别着急，在命令模式下按 **U** 就可以撤销刚才的操作。这样我们就可以放心尝试各种删除操作了。如果我们需要删除不止一行内容，可以在“dd”前加上相应的数字。比如我们需要删除3行。

```
3dd
```

这相当于连续按6次 **D**，结果如图 6-20 所示。



图 6-20 删除3行结果

可以看到图中左下角的提示“3 fewer lines”（减少了3行），操作成功了。好了，按 **U** 来撤销吧，我们再试试别的。现在光标在文档的首行，试试下面这组快捷键。

```
dw
```

这组快捷键融合了移动和删除操作，表示向右删除一个单词。结果如图 6-21 所示。



图 6-21 删除一个单词

通过这样的方法，我们可以在不移动光标的情况下有效选择要删除的文本。上面的例子里会从光标的位置开始向右删除一个单词。当然我们也可以通过多次使用的快捷键 **X** 实现相同的效果，但显然使用 “dw” 要简捷精准得多。好了，让我们再一次使用快捷键 **U** 撤销刚才的操作。接下来，我们再进行一步，我们已经了解在命令前面加上数字可以指定命令的重复次数，那么我们可以进一步改进我们的命令，一次删除多个单词。

2dw



图 6-22 一次删除两个单词

有点意思不是吗？通过这样的方式我们就可以通过组合命令来实现复杂的功能。例如，如果你想要删除到行末的所有内容，则使用 **d\$**，正是这种命令组合方式赋予了 Vim 无与伦比的文本编辑魔力。虽然很多命令是为英文文本设计的，但当你进行程序编写时自然会体会到其便捷。

一些短小精悍的命令

在进入下一部分前，我们再来看看还有哪些命令值得注意，以防有漏网之鱼。快捷键 **I** 可以帮助我们进入插入模式，但有时候我们需要在当前行的前一行或是后一行插入内容，这时我们就要用到快捷键 **Shift + O** 和 **O** 了。通过这两个快捷键，我们可以在当前行前或行后插入一行并进入插入模式。有时我们不需要做大量的文本修改，只是需要修改当前光标下的一个字符。这时我们可以使用快捷键 **R**，按 **R** 后直接输入想要替换的字符，无需进行模式切换，简单高效。

可视模式

可视模式并不常用，但是它却可以为我们节省大量时间。类似于文字处理软件中用鼠标选择文本，在这一模式下我们可以选中一块文本，以便进一步编辑。我们可以使用快捷键 **Shift + V** 和 **V** 从命令模式切换到可视模式。两者的区别在于：使用 **Shift + V** 时，以行为单位选择文本，使用 **V** 时，以字符为单位选择文本。

可视模式存在的意义在于方便地为 Vim 的各种命令指定一个作用范围。我们选择一段文本，如图 6-23 所示。

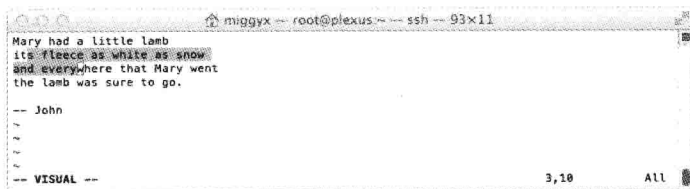


图 6-23 通过可视模式选择文本

你当然可以按需选择按行或是按字符选择文本，这里我们使用快捷键 **V** 来按字符选择。当文本选定后，按快捷键 **D** 可以发现，所有被选定的文本都被删除了，这正是可视模式的作用。

文本缩进

如果想要缩进文件的前两行，我们可以通过可视模式选定前两行文本，然后通过快捷键 **Shift** + **[** 和 **Shift** + **]** 来调整缩进量，结果如图 6-24 所示。

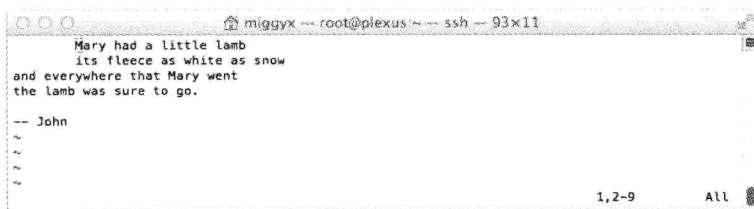


图 6-24 缩进选定的文本

除了删除和缩进外，可视模式还可以帮助我们更方便地复制粘贴文本。

6.5 Vim Style 的复制、粘贴

在终端中进行复制、粘贴多少有些麻烦。这是因为在终端里运行的程序都是“两耳不闻窗外事”的。比如，从终端复制文本到本地的剪贴板时，终端上运行的程序（比如 Vim）并不知道你做了些什么。相应的，从本地剪贴板中复制内容到终端上的 Vim 时，Vim 只会简单地认为这是一连串的键盘输入，而不会理解其来源。大多数情况下，这不会带来什么问题。

可惜，凡事总有例外。比如你有一个一分为二的终端窗口，需要将左侧窗口的第一段文本复制到右侧，无论操作多少次却总是不成功时，自然令人抓狂。

如果你只是想进行一次粘贴，当然没什么，但如果你想要粘贴 10004 次呢？Vim 帮不了你，因为它不知道你的本地剪贴板上有些什么。你会发现类似的问题会不时地出现，你难免会抓狂地想：“如果我能复制这段文本的话……”

我们需要的是一个远程剪贴板，供远程应用使用。虽然没有统一的标准，但很多程序提供了

自己的解决方法，比如 Vim。我们可以通过快捷键 **Y** 进行文本复制，与快捷键 **D** 的操作有些类似，我们可以通过 “yy” 来进行整行的复制，用 “yw” 来复制一个单词，或是使用可视模式进行操作。在可视模式中，首先选定需要复制的文本，然后通过快捷键 **Y** 将之复制到剪贴板。使用快捷键 **Shift** + **P** 和 **p** 可以进行粘贴操作，两者的区别在于：使用 **Shift** + **P** 时，将粘贴到当前行之前，使用 **p** 时，将粘贴到当前行之后。我们也可方便地指定粘贴操作的执行次数，比如 “40p”，结果如图 6-25 所示。

或许你还没注意到，其实我们已经接触过如何进行剪切操作了。当使用快捷键 **D** 时，在删除文本的同时，也将相应的文本发送到了剪贴板。所以，当通过 “dd” 删除一行后，可以使用快捷键 **Shift** + **P** 进行恢复。注意，如果使用快捷键 **p** 会将这一行插入到原位置的下一行。



图 6-25 重复粘贴

本章小结

本章介绍了文本文件和二进制文件的区别，以及文本文件成为软件配置文件通用标准的原因。我们涉及了两种常用的文本编辑器的基础知识，以便你能够进一步地探索。在下一章中，我们将接触 BASH Shell 以及一些系统管理员的必备知识。

管理你的树莓派

至此，我们已经学会了树莓派的各种安装配置，十八般武艺“略”通。你或许会发现，在使用过程中最让人郁闷的是，每次树莓派断电后我们都需要重新手动登录、配置，让人不胜其烦。好消息是，看完本章后，这些问题将不复存在！

本章将讲述系统管理相关内容，包括一些开机启动基础知识。你将学会如何编写 **BASH** 脚本（放心，我们会先告诉你脚本是什么）。余下的部分将涉及诸如用户管理的一些系统安全常识，教你如何修改账号密码以及添加新用户。

7.1 远程访问树莓派

命令行的使用可能是从 Windows 迁移到 Linux 的最大障碍。对初学者来说，它的界面不够友好，更不用说它的操作第一眼看上去远不如图形界面直观。然而，深入了解后我们会发现，命令行的工作效率要远高于图形界面，运键如飞的感觉比用各种拖沓的鼠标操作好太多。当然想要达到运键如飞的境界，大量的练习是必不可少的。

这里再次强调命令行的重要性是因为大多数树莓派的管理操作需要通过命令行进行。这样的管理方式比图形界面要高效环保得多。进行远程访问时，除了保证有可用的网络连接外，还需要用到我们在第3章提到的 SSH 工具。

网络

大多数人使用路由器来实现多种设备同时访问网络。路由器使用一种称为 DHCP（Dynamic Host Control Protocol，动态主机控制协议）的协议来进行 IP 地址的自动分配（在第9章中我们会介绍更详细的网络配置方法）。

在某些情况下，我们可能需要手动申请 DHCP 为我们分配 IP 地址，这时我们需要用到 **dhclient** 命令。与大多数系统管理命令相同，这一命令需要使用 **root** 权限执行。此外，如果不指定参数，**dhclient** 命令会作用于当前系统中的所有网络连接。如果需要指定特定的网络连接，需要以该连接名作为参数，比如：**dhclient eth0**。

DNS

DNS (Domain Name System, 域名系统) 是网络设置中不可或缺的一环。它实现了网络地址 (比如 `www.apress.com`) 到 IP 地址 (比如 `173.203.147.183`) 的转换。在 Linux 系统中, DNS 服务器的配置文件为 `/etc/resolv.conf`。其内容如下 (其中的 IP 地址可能是你的路由器, 也可能是某台 DNS 服务器, 比如 Google 的 `8.8.8.8`):

```
nameserver 10.0.0.1
```

SSH

SSH 为我们提供了一种安全地远程访问树莓派的方式。在第 3 章中我们已经学习了相应内容, 这里不再赘述。sshd 守护进程为我们提供了开箱即用的 SSH 功能。我们也可以通过 `sshd init` 脚本手动启动或关闭 sshd。类似的初始化脚本都存放在 `/etc/init.d/` 目录下。开机时, 这些脚本会自动执行。我们可以使用 `start`、`stop` 以及 `restart` 这 3 个参数来控制这些脚本。比如, 执行 `/etc/init.d/ssh restart` 就可以重启 SSH。通过模仿这些脚本, 我们也可以实现自己的开机自动执行脚本。

7.2 通过 BASH 学编程

如果除去“你能帮我修下电脑吗?”, “你能教我怎么用 C 语言写程序吗?” 这个问题应该算是程序猿和程序媛们最常遇到的问题了。通常, 如果你追问一句, 往往会发现人们想实现的其实是十分简单的功能。虽然 C 语言有趣有用, 但是杀鸡焉用牛刀。除去 C 语言外, 另有蹊径可以让我们更轻松地领略到软件设计的大道至简。

本章正是为此而设。我们会通过 BASH 来介绍程序设计的一些基础知识、基本逻辑结构和一些必要的基本设计原则。或许有人不信, 但 BASH 甚至可以看作学习程序设计基础的最佳方式。由于在 Linux 中的广泛应用, BASH 也成为世界上最为常用的编程语言中的一种。

BASH 是什么

在第 3 章中我们已经介绍过 BASH (Bourne Again Shell)。BASH 是迄今为止使用最广泛的 shell, 大多数 Linux 及 UNIX 都将其作为默认 shell。当你通过命令行登录系统看到自己的命令行提示符时可能并没有注意, 这个命令行提示符正是由 BASH 所提供的。shell 会接收各种命令和指令并执行相应的系统功能, 并通过与其他程序的交互来产生输出结果。

虽然上面这句话听上去有些过于简单, 但这正是对软件运行核心概念的精炼描述, 即输入、处理、输出的三段论。这一模型在日常工作中应用广泛, 人们往往熟视无睹。但它正是构成了编程的基础模型。

让我们再回到什么是 BASH 这个问题。BASH 是一种 shell, 而 shell 则是提供了一种接收输入、进行处理、生成输出的方法。与其他编程语言一样, BASH 提供了一系列工具让我们能够实现基本的逻辑结构。这也是 BASH 既可以作为编程语言又可以作为 shell 的原因。至此我们已经大致了解了 BASH、shell 和程序设计的基本概念, 下面将以此为基础介绍如何使用 BASH 编写

程序。

BASH 初步

许多面试者无法在纸上完成一道十分简单的程序测试题。这道题被称为 FizzBuzz 测试¹，即写一个程序，依次输出 1~100 的整数，遇到可以被 3 整除的数则输出“Fizz”，遇到可以被 5 整除的数则输出“Buzz”，遇到可以同时被 3 和 5 整除的数则输出“FizzBuzz”，就这么简单。归根结底，大多数时候我们在编程时，不过是在构建逻辑模块，黏合、处理输入信息，积少成多，周而复始。

稍后，我们将教会你如何使用 BASH 编写答案，通过 FizzBuzz 测试。从第 6 章介绍过的文本编辑器里选一个，让我们开始编程吧！首先，输入这样一行：

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

这被称为脚本标准起始行（shebang），它包含脚本解释器的绝对路径。在上面的例子中我们使用 BASH 的脚本解释器（/bin/bash），在树莓派和其他 Linux 或 UNIX 系统中都可以找到它。需要记住，脚本标准起始行是脚本程序不可或缺的组成部分。

解释 VS 编译

这两个概念听起来或许有些陌生。虽然不管是解释型语言还是编译型语言所写的程序最终都交给计算机执行，但了解下两者的区别总是没有坏处的。解释型语言（有时被称为脚本语言）将一系列指令交由解释器解析并最终执行。这就像是在纸上写下蛋糕的详细配方，解释器为你烹调出美味的蛋糕。

注意 这里所说的配方其实和戏剧的脚本十分相似，你应该可以照本宣科地执行上面的内容。这也正是解释型语言又被称为脚本语言的原因，和戏剧脚本相似，同一个脚本可以重现相同的故事（输出结果）。

对编译型语言来说，源代码才是它的配方。但与解释型语言不同，编译型语言事先通过编译器生成程序的可执行文件。这一可执行文件由机器语言构成，可以独立运行。这就像先设计一台制作蛋糕的机器，组装机后，让它去做蛋糕。虽然最终都能吃到蛋糕，但相对的第一种方式因为要花在沟通上，执行效率相对较低。而第二种方式则需要花费更多精力在蛋糕机器的设计制造上。之前我们说用 C 语言解决小问题是杀鸡用牛刀，正是因为它是一种编译型语言。

BASH 中的输出

再说回我们刚写了一行的这个例子，如果我们现在执行它，显然是得不到任何输出结果的。让我们试着用 echo 命令输出点什么吧。

¹ 译者注：名字取自同名游戏 https://en.wikipedia.org/wiki/Fizz_buzz。

```
#!/bin/bash
echo "Hello World!"
```

好了，现在我们来执行下看一看。将我们的脚本命名为 `fizzbuzz.sh`，给我们的脚本加上执行权限，并执行它。

```
$ chmod +x fizzbuzz.sh
$ ./fizzbuzz.sh
```

我们会得到如下输出结果。

```
Hello World!
```

这应该算是最为常见也最为简单的编程入门程序了。在这个程序中我们的输入是“请输出 Hello World!”。在源程序中，我们还可以看到一对双引号，它们的作用是告知程序，Hello World 是字符串，而不是程序指令。在所有的编程语言中都需要通过一定的方式来区分数据和逻辑语句。

那么如果我们想要输出双引号该怎么办呢？别担心，通过转义字符我们就可以实现这样的输出。转义字符是一个特殊的符号，告诉系统紧随其后的那个字符直接输出。在大多数情况下转义字符都是反斜杠（\），我们的脚本可以改为：

```
#!/bin/bash
echo "\"Hello World!\""
```

输出结果为：

```
$ ./fizzbuzz.sh
"Hello World!"
```

别忘记，我们也可以通过转义字符来转义其自身¹。忘记使用转义字符是编程新手最常见的错误，别忘了你写的程序是要交给计算机执行的。

 **注意** 最为常用的转义字符是 `\n`，表示换行符。换行符后的内容会在新的一行显示。

此外，有语法高亮功能的文本编辑器可以帮助我们判断字符串的界限。Vim 和 nano 都有语法高亮功能，遗憾的是图形界面下默认的 LeafPad 不支持这一功能，不过找一个合适的替代品并不困难。

变量

变量是一种对数据的抽象表示。也可以说变量是计算机检索存储信息的一种方式。大多程序并不像上面的 Hello World 那样简单，它们需要处理各种信息。为了更好地进行说明，我们来看看前面提到的 FizzBuzz 测试中从 0 到 100 的计数部分。依次写入并输出当然算是一种实现方法，但实在不能算是聪明的方法。我们可以定义一个变量“number”，并对它不断加 1 直到 100。

有了变量这一基本概念后，还有一些注意事项我们需要了解。首先，变量的类型多种多样，包括布尔型、整型、双精度整型、长整型、字符型、字符串、浮点型、向量、数组等等。好在

¹ 译者注：将相应语句改为 `echo "\""`，则得到一个反斜杠的输出。

BASH 中涉及的不多，以下几种十分重要。

- 整型：顾名思义，这种变量用来表示一个整数，并可以方便地进行加减乘除等运算。
- 字符串：这种变量用来表示一串字符。往往用于输入输出，而非内部逻辑。
- 数组：数组的存在有些特殊。因为这是一种储存用于存储一定数量变量的变量。数组中存储的变量称为数组元素。数组元素的标号从“0”开始，我们可以通过在数组名后面添加[3]来找到数组中的第4个元素。

if 语句

在学会了输出和存储后，我们再来看看如何进行数据的处理。程序设计中有两种最为常见的逻辑结构，即选择结构和循环结构。

if 语句就属于选择结构。这种语句主要实现了当特定条件满足时，执行相应语句的功能。FizzBuzz 测试的例子中，我们需要判断变量 number 能否被 3 或 5 整除，并给出不同的输出。

BASH 中 if 语句的基本结构如下：

```
if [ <判断条件> ]; then
    执行语句
fi
```

结尾处的 fi 表示语句结束，接下来让我们看看如何通过 if 语句执行一些简单的计算操作。

注意 前一半方括号两侧都需加上空格；后一半方括号前面加空格，后面加分号。不这样做的话会导致程序出错。

简单运算测试

在 FizzBuzz 测试中我们需要验证一个数是否能被另一个数整除。想要通过四则运算来实现这样的功能对新手来说多少会有些困难。幸运的是我们可以通过取余运算(%)来方便地得到除法运算中的余数。

```
12 % 3 = 0
12 % 5 = 2
```

所以我们可以这样解答 FizzBuzz 测试：变量 number 对 3 取余为 0 时输出 Fizz，对 5 取余为 0 时输出 Buzz，对 3 和 5 取余都为 0 时输出 FizzBuzz。

```
if [ number % 3 = 0 ]; then
    echo "Fizz"
fi
if [ number % 5 = 0 ]; then
    echo "Buzz"
fi
if [ number % 3 = 0 ]; then
    if [ number % 5 = 0 ]; then
        echo "FizzBuzz"
```

```
fi
fi
```

看起来是个不错的尝试，现在我们需要再做些工作让程序能实际在 BASH 中运行起来。首先来看变量，在 BASH 中对变量有两种操作模式。首先是赋值模式，它会创建一个变量并为其赋值。我们可以创建变量 `number` 并赋值为 1。

```
number=1
```

当我们需要再次使用变量 `number` 时，我们就需要使用引用模式，即在变量前加上美元符号 (\$) 来获取变量的值。了解了变量的两种操作模式后我们来改写下我们的代码。

```
if [ $number % 3 = 0 ]; then
    echo "Fizz"
fi
if [ $number % 5 = 0 ]; then
    echo "Buzz"
fi
if [ $number % 3 = 0 ]; then
    if [ $number % 5 = 0 ]; then
        echo "FizzBuzz"
    fi
fi
```

这样我们的代码看起来就更像回事了，不是吗？不过你或许也发现了，等号 (=) 可以用来赋值但却不能辨别两个数是否相等。大多数编程语言中使用连续两个等号 (==) 来辨别两个数是否相等，而 BASH 自己定义了一系列运算测试符来解决类似的问题。

- `-eq` 验证两数是否相等。
- `-ne` 验证两数是否不等。
- `-gt` 验证前数是否大于后数。
- `-lt` 验证前数是否小于后数。
- `-ge` 验证前数是否大于等于后数。
- `-le` 验证前数是否小于等于后数。

让我们把运算测试符加入我们的代码来实现取余运算结果是否为 0 的检测。

```
if [ $number % 3 -eq 0 ]; then
    echo "Fizz"
fi
if [ $number % 5 -eq 0 ]; then
    echo "Buzz"
fi
if [ $number % 3 -eq 0 ]; then
    if [ $number % 5 -eq 0 ]; then
        echo "FizzBuzz"
    fi
fi
```

最后，需要告知 **BASH** 我们正在进行的是一种运算测试。我们需要使用美元符号 (\$) 和两对圆括号， $5 \% 3$ 应该写作 `$(5 \% 3)`。

```
if [ $((($number \% 3)) -eq 0 ]; then
    echo "Fizz"
fi
if [ $((($number \% 5)) -eq 0 ]; then
    echo "Buzz"
fi
if [ $((($number \% 3)) -eq 0 ]; then
    if [ $((($number \% 5)) -eq 0 ]; then
        echo "FizzBuzz"
    fi
fi
```

这样我们就简单实现了单次 **FizzBuzz** 测试，用到了 **if** 语句、变量、**print** 命令以及运算测试符。涉及的东西真不少不是吗？让我们来试试看代码能否正常工作。

我们可以将变量 **number** 赋值（这里赋值为 15）来进行测试。

```
#!/bin/bash
number=15
if [ $((($number \% 3)) -eq 0 ]; then
    echo "Fizz"
fi
if [ $((($number \% 5)) -eq 0 ]; then
    echo "Buzz"
fi
if [ $((($number \% 3)) -eq 0 ]; then
    if [ $((($number \% 5)) -eq 0 ]; then
        echo "FizzBuzz"
    fi
fi
```

保存并运行我们的脚本，可以得到如下输出。

```
$ ./fizzbuzz.sh
Fizz
Buzz
FizzBuzz
```

排错

看出结果不正确了？别担心，做几次深呼吸，这并不是世界末日，事实上这正是学习 **if** 语句新功能的好机会。我们已经知道 **if** 语句在判断条件成立的时候会执行相应语句，而其实我们还可以加上 **else** 分支。**else** 分支和原有的 **if** 分支同级，整个 **if** 语句的逻辑就变成了这样：满足判断条件执行 **if** 分支的语句，不满足判断条件则执行 **else** 分支的语句。

```
if [ <判断条件> ]; then
    执行语句
```

```
else
    执行另一些语句
fi
```

想要把 `else` 分支应用于 `FizzBuzz` 测试，我们需要思考如何厘清 3 种输出间的逻辑关系。正是这种思考让许多人觉得程序设计很难，别担心，我们一起来解决这个问题。

在前面的程序中，我们串行地进行 3 种输出的判断，这样就出现了一个数字同时导致 3 种输出的问题。这次我们先从同时检查对 3 和对 5 的整除情况开始。

```
if [ $((($number % 3)) -eq 0 ]; then
    if [ $((($number % 5)) -eq 0 ]; then
        echo "FizzBuzz"
    fi
fi
```

如果一个数可以被 3 整除但不能被 5 整除呢？我们应该输出 `Fizz`。这样我们就应该在检测对 5 整除的 `if` 语句中添加 `else` 分支。

```
if [ $((($number % 3)) -eq 0 ]; then
    if [ $((($number % 5)) -eq 0 ]; then
        echo "FizzBuzz"
    else
        echo "Fizz"
    fi
fi
```

我们只剩下一种情况了，如果一个数可以被 5 整除但不能被 3 整除呢？我们就应该在检测对 3 整除的 `if` 语句中添加 `else` 分支。

```
if [ $((($number % 3)) -eq 0 ]; then
    if [ $((($number % 5)) -eq 0 ]; then
        echo "FizzBuzz"
    else
        echo "Fizz"
    fi
else
    if [ $((($number % 5)) -eq 0 ]; then
        echo "Buzz"
    fi
fi
```

让我们再来测试下完工后的语句。

```
$ ./fizzbuzz.sh
FizzBuzz
```

终于，我们得到了正确的结果。你可以改变变量 `number` 的值再多尝试几次，相信都能够得到正确的结果。下一步，让我们来看看如何通过另一种逻辑操作来实现从 1 到 100 的计数。

逻辑控制：循环

循环逻辑有点像之前介绍的 if 语句，不过区别在于当条件为真时，循环逻辑将会不断执行其中的语句块，直到条件变为假，最简单的循环逻辑例子就是将数字从 1 开始，不断加 1，直到变为 100 停止。BASH 脚本中的循环逻辑语法如下：

```
while [ TEST ]; do
    DO SOMETHING
done
```

循环逻辑和 if 语句的相同之处都是当条件为真时执行代码，不过 if 语句只执行一次，而 while 循环只有当条件为假时才会停止，否则将会不断地执行其中的代码块，所以使用循环逻辑时最重要的就是控制条件的真假，否则循环有可能无法停止，这种情况被称为**死循环**，程序再无法继续执行下去了。一旦遇到这种情况，可以使用 Ctrl+C 组合键中止程序的运行。

很多程序停止响应就是因为出现了死循环，大多是由于没有考虑到一些特殊的条件。所以当你编程时，一定要注意各种可能发生的状况。

我们的 FizzBuzz 小程序也要使用一下循环逻辑，实现从 1 数到 100，循环的条件就是数字是否小于等于 100，当数字不大于 100 时执行循环中的代码。我们使用循环结构将之前已有的代码包裹起来，得到类似下面的代码。当然先不要急着执行这段代码，因为该代码还不会自动增加数字的值，所以循环永远也无法停止。

```
#!/bin/bash
number=1
while [ $number -le 100 ]; do
    if [ $((($number % 3)) -eq 0) ]; then
        if [ $((($number % 5)) -eq 0) ]; then
            echo "FizzBuzz"
        else
            echo "Fizz"
        fi
    else
        if [ $((($number % 5)) -eq 0) ]; then
            echo "Buzz"
        fi
    fi
done
```

循环逻辑的结构已经写好，我们需要让循环可以停止，所以在每次循环时对 number 加 1。不过我们希望先执行计算再调整 number 的值，所以在循环内容最后的 done 语句之前增加一行代码。

```
number=$((($number + 1))
```

完成后，我们可以执行这段代码，应该会得到如下的输出。

```
Fizz
Buzz
Fizz
```

```

Fizz
Buzz
Fizz
FizzBuzz
...

```

看起来效果不错吧,不过让我们更进一步,每次循环都输出当前的 `number` 值。我们将 `$number` 写在 `echo` 语句的 `Fizz`、`Buzz` 和 `FizzBuzz` 之前。你会发现,虽然双引号将其中的内容都理解为字符串,但是如果表达式以 `$` 开头,就会被理解为一个变量,这样输出就很方便。

我们还要增加一个 `else` 语句,因为之前的条件并没有覆盖 1~100 所有的情况,你可以自己试着增加这个 `else` 语句,最终你得到的应该是类似以下的代码。

```

#!/bin/bash
number=1
while [ $number -le 100 ]; do
if [ $((($number % 3)) -eq 0 ); then
    if [ $((($number % 5)) -eq 0 ); then
        echo "$number - FizzBuzz"
    else
        echo "$number - Fizz"
    fi
else
    if [ $((($number % 5)) -eq 0 ); then
        echo "$number - Buzz"
    else
        echo $number
    fi
fi
number=$((($number +1))
done

```

到这里我们的 `FizzBuzz` 就完成了。

排错

调试代码中的错误可能会是一场噩梦,不过 `echo` 命令可以很好地帮助你解决问题,你可以使用 `echo` 输出任何内容。如果你不确定程序到底是如何执行的,那么可以在代码中使用 `echo`,运行后检查输出的内容,了解程序的执行情况。例如你不确定 `if` 语句中的代码是否被执行了,可以在其中加上 `echo`,还可以再写一个 `else` 和 `echo`,以确定条件到底为真或者假。调试时另一个重点是注意观察命令行输出的任何错误,很多情况下错误消息会带有行号,这可以明确地告诉你错误发生在哪里。最后你还可以请教 Google,互联网上有各种问题的解答,Google 可以帮助你很快地找到答案。

7.3 第一个 BASH 脚本

很多应用程序都使用 `init` 脚本来启动¹，该脚本在 `BASH` 中进行一些特别的操作和配置，为应用程序的执行准备环境，启动并控制程序，不过 `init` 脚本基本不涉及到程序的逻辑。

让我们先来看一个 `init` 脚本的例子，看看它是如何工作的。下面的 `init` 脚本来自 `XBMC`²，我们后面会介绍如何使用 `XBMC`，现在仅仅看看 `init` 脚本。

```
### BEGIN INIT INFO
# Provides:          xbmc
# Required-Start:    $all
# Required-Stop:     $all
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start XBMC
# Description:       Start XBMC
### END INIT INFO
DAEMON=/usr/bin/xinit
DAEMON_OPTS="/usr/lib/xbmc/xbmc.bin"
NAME=xbmc
DESC=XBMC
RUN_AS=root
PID_FILE=/var/run/xbmc.pid
test -x $DAEMON || exit 0
set -e
case "$1" in
  start)
    echo "Starting $DESC"
    start-stop-daemon --start -c $RUN_AS --background --pidfile $PID_FILE --make-
-pidfile --exec $DAEMON -- $DAEMON_OPTS
    ;;
  stop)
    echo "Stopping $DESC"
    start-stop-daemon --stop --pidfile $PID_FILE
    ;;
  restart|force-reload)
    echo "Restarting $DESC"
    start-stop-daemon --stop --pidfile $PID_FILE
    sleep 5
    start-stop-daemon --start -c $RUN_AS --background --pidfile $PID_FILE --make-
-pidfile --exec $DAEMON -- $DAEMON_OPTS
    ;;
  *)
    echo "Usage: $0 {start|stop|restart|force-reload}"
    exit 1
esac
```

¹ 译者注：这里指的是应用程序的守护进程。

² `XBMC` 是一种跨平台的媒体中心程序。

```

*)
    echo "Usage: /etc./init.d/$NAME{start|stop|restart|force-reload}" >&2
    exit 1
;;
esac
exit 0

```

脚本的一开始有很多行以#开头的文本，这些都是注释。在 BASH 脚本中，#是注释符号，以此开始的文本行不会被视为代码。这段开头的注释被称为 INIT INFO，这对于 init 脚本是很重要的，INIT INFO 使用固定的格式以便其他应用或者开发人员读取。

```

### BEGIN INIT INFO
# Provides:          xbmc
# Required-Start:    $all
# Required-Stop:     $all
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Start XBMC
# Description:       Start XBMC
### END INIT INFO

```

INIT INFO 描述了该脚本为哪个应用程序提供服务，例如 xbmc。这里还说明了运行该应用程序之前需要先运行其他什么应用程序，例如这里写的\$all，表示 XBMC 将最后启动，以确保其他的依赖应用程序都已经启动。init 脚本中最重要的描述可能就是 Default-Start 和 Default-Stop，这里描述了该脚本在 Linux 系统中的运行级别，运行级别的定义如下：

- Level 0: 停机 (Halt; shut down and power off)。
- Level 1: 单用户 (Single User Mode; only basic system functionality)。
- Level 2 & 3: 多用户，Level 3 包括网络连接 (Multi User; adds networking and multiple user functionality)。
- Level 4: 用户自定义 (User Defined)。
- Level 5: 系统一般运行状态 (Normal System running state)。
- Level 6: 重启 (Reboot)。

根据该定义 XBMC 将在 level 2~5 时启动，在 level 0、1 和 6 时停止。

使用 case 语句选择并执行代码

脚本的初始化结束后，将会执行一些测试语句以确定执行哪一个 case 语句。BASH 脚本中的 case 语句有点像是一系列的 if 语句，不过多个 case 语句测试同一个变量，根据这个变量的不同值，执行不同的语句。在大部分 init 脚本中，case 语句用于确定执行什么操作。

在 XBMC 的 init 脚本中，通过判断\$1 变量决定执行哪一段代码，\$1 变量表示了执行脚本命令的第一个参数，例如启动 XBMC，你可以执行以下命令。

```
/etc./init.d/xbmc start
```

该命令为\$1 变量赋值 start，之后根据脚本中定义的 case 语句执行不同的代码。

- start: 启动 XBMC。
- stop: 停止 XBMC。
- restart|force-reload: 使用 restart 或者 force-reload 命令行参数将会停止并重新启动 XBMC，这里的|符号并不是之前介绍的管道符号，而是表示“或”，当匹配任何一个条件时，均执行定义的代码。
- *: 其他的代码。

每个 case 语句后面都有一块代码，当匹配该 case 条件时将执行这些代码，代码的最后使用两个分号 (;) 表示该 case 语句结束，就像 fi 表示 if 语句的结束一样。

应用程序启动应用程序：forking

在 init 脚本的 case 语句中我们能看到很多 start-stop-daemon 命令，这不是 BASH 脚本命令而是一个应用程序。这就是 BASH 脚本的另一个强大之处，你可以在脚本中执行应用程序。

start-stop-daemon 程序为你创建指定的进程，该操作被称为 forking，因为它会将当前的进程复制，并通过复制的进程执行指定应用程序。通过 start 参数，脚本将调用 start-stop-daemon 程序为你复制进程，启动程序，并管理该程序。

学习了这些之后，你可以自己写一个 init 脚本来启动和停止应用程序了，写一些注释来说明如何启动应用程序，并且在 case 语句中调用 start-stop-daemon 启动应用程序并管理其运行状态。

设置启动脚本

有了启动脚本，还需要进行一些设置才能使用它。首先需要使用 chmod 为你的启动脚本增加可执行标志，之后你可以尝试使用 start 和 stop 参数，测试脚本执行得是否正常，如果能够正常运行，那很好。让我们把它添加到系统的启动过程中。在以前的 Linux 系统中，一般通过创建链接文件来实现不同的启动级别，不过在新版的 Linux 系统中，update-rc.d 可以帮助你更快地完成 init 脚本添加。

我们调用 update-rc.d 脚本，并向其提供需要启动的 init 脚本名称（例如 xbmc），之后再加上运行级别参数（例如 defaults），它将会从 init 脚本头部的注释中读取对应的配置。最终组合成 update-rc.d xbmc defaults，我们以 root 用户执行该命令，应该会看到类似下面的输出。

```
$ sudo update-rc.d xbmc defaults
update-rc.d: using dependency based boot sequencing
```

完成后重启系统，指定的应用程序就会在系统启动时运行了。

自己写一个 init 脚本

上一节我们介绍了 init 脚本的格式和内容，下面让我们自己写一个试试看。其实写一个 init 脚本也很简单，只要按照之前介绍的文件结构，在指定的位置填上我们自己的代码就可以了。在

init 脚本中可以执行各种操作，不过本节我们准备写一个很简单的例子，该 init 脚本将使用 touch 命令创建文件，并在退出时使用 rm 命令删除文件。touch 命令可以在磁盘上创建一个文件，如果指定文件名的文件已经存在，那么该命令将会修改文件的修改时间。你也可以自己尝试调用 start-stop-daemon 程序来丰富 init 脚本的功能，不过先让我们来看看这个简单的怎么写。

我们之前介绍过，init 脚本是 BASH 脚本的一种，所以也需要符合所有的 BASH 脚本要求，因此和其他的 BASH 脚本一样，我们以下面的代码开头。

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

接下来我们写的内容需要符合 init 脚本的格式，所以就像 XBMC 的启动脚本一样，我们需要写 INIT INFO，以说明我们将要使用的运行级别和其他必要的信息。我们的脚本命名为 touchfile.sh，因此服务名称为 touchfile，你也可以根据需要自己调整内容。最后应该会写出类似下面的 INIT INFO 内容。

```
### BEGIN INIT INFO
# Provides:          touchfile
# Required-Start:    $all
# Required-Stop:     $all
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Run TouchFile
# Description:       Run TouchFile
### END INIT INFO
```

完成 INIT INFO 后，我们开始定义变量，应用程序启动时需要的各种信息都可以定义为变量。例如我们将需要使用 touch 创建的文件路径定义为变量。

```
TOUCHFILE="/var/tmp/touch.file"
```

下面我们来写 case 语句，case 条件将会覆盖所有可能的操作，我们的 touchfile 将会支持以下的操作。

- Start: 创建文件。
- Stop: 删除文件。
- Restart: 删除文件并重新创建。
- Reload: 创建文件。
- Default Case: 显示如何使用。

为了覆盖这些操作，我们要写 4 个 case 语句，我们的代码如下：

```
case "$1" in
    start)
;;
    stop)
;;
    restart)
;;
    reload)
;;
```

```
;;
*)
;;
esac
exit 0
```

这样 init 脚本的大体结构就出来了，我们再填写一些代码让它能够运行起来。我们将所有的内容写在一起，并按照之前列表中的定义，使用一系列的代码实现每一个 case 语句，于是得到了如下的脚本。

```
#!/bin/bash
### BEGIN INIT INFO
# Provides:          touchfile
# Required-Start:    $all
# Required-Stop:     $all
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: Run TouchFile
# Description:       Run TouchFile
### END INIT INFO
TOUCHFILE="/var/tmp/touch.file"
case "$1" in
    start)
        echo "Creating $TOUCHFILE"
        touch $TOUCHFILE
    ;;
    stop)
        echo "Removing $TOUCHFILE"
        rm $TOUCHFILE
    ;;
    restart)
        echo "Recreating $TOUCHFILE"
        rm $TOUCHFILE
        touch $TOUCHFILE
    ;;
    reload)
        echo "Re-Touching $TOUCHFILE"
        touch $TOUCHFILE
    ;;
    *)
        echo " Usage: touchfile.sh <start|stop|restart|reload>"
    ;;
esac
exit 0
```

到这里，我们写出了一个简单的 init 脚本，可以使用 start、stop、restart 和 reload 作为参数，执行相应的操作。我们将其放入 /etc/init.d 目录，并使用 chmod +x 将其标记为可执行文件。完成后你可以测试一下脚本是否能够正常工作，最后使用 update-rc.d 将其加入系统启动过程。这就是整个 init 脚本的编写和使用，你可以将其应用于你自己的应用程序上。

7.4 安全和用户管理

管理系统时，安全往往会被忽略，因为安全性似乎有点像黑魔法，很难学习和掌握。

树莓派安全守则

我们总结了一些简单的树莓派安全守则，遵守这些规则可以带来更多的安全性。

- 不要以 root 用户登录系统，如果必须使用 root 登录，完成后立即注销。
- 在需要 root 权限时尽量使用 sudo。
- 使用复杂的密码——足够的长度和足够的复杂度，不会很容易地被猜出或者试出。
- 经常修改密码。
- 经常检查系统日志，特别是/var/log/auth.log，这里记录了系统所有的用户验证信息。
- 非必须不要启动应用程序。
- 尽量不要将系统暴露在互联网上。
- 尽量限制文件权限。

这些守则看起来都很简单，但是很多人都会忽略它们，人们都觉得自己电脑不会被人攻击，不过最好还是能尽量遵守这些规则，安全一点总是没错的。看了这些规则，你可能会说：“我立即就改密码，不过我先要知道怎么改”。好吧，在 Linux 系统中修改密码也很简单，输入 passwd 命令，系统就会提示输入新密码（两次），两次确认后就完成了密码修改。

如果当前是 root 用户，你还可以修改其他用户的密码，只需将用户名作为 passwd 的唯一参数，其余的过程都是一样的。

新增用户

我们不推荐一直使用 root 用户登录系统，所以需要学习一下如何新建一个普通用户。在 Linux 系统中，用户管理文件位于/etc/passwd，你可以打开该文件看看其中的内容，应该和下面的类似。

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
```

每一行代表一个用户，显示了用户名、密码 (x)、用户 ID、用户组 ID、用户组名称、用户主目录路径和用户 shell 程序，这些就是 Linux 系统所需的用户信息。

虽然该文件中有密码字段，但是现在 Linux 系统已经不在该文件中存储密码了，而是使用一个占位符 (x) 代替。密码则是单独存储在/etc/shadow 文件中。之所以独立出 shadow 文件是为了能够让系统管理程序读取用户信息，但是保证只有 root 用户可以读取密码数据。

我们接下来使用 useradd 命令和一些参数为树莓派添加一个名为 raspberry 的用户。其实还有另一个命令称为 adduser，不过该命令不提供丰富的参数来配置用户的细节信息，而是仅仅添加一个用户。所以如果突然忘了如何使用 useradd 的参数，可以使用 adduser。

使用 useradd 命令时，我们除了提供用户名，另外可以提供的是主目录路径，通过 -d 开关加上路径定义主目录，常见的主目录路径都是/home/<username>，所以我们使用/home/raspberry 作

为 `raspberrry` 用户的主目录。目前 `/home/raspberrry` 并不存在，不过你不需要急着手工新建该目录，我们通过 `-m` 开关让 `useradd` 为我们建立指定的目录，最终我们得到了以下的命令。

```
$ sudo useradd -d /home/raspberrry -m raspberrry
```

更进一步，我们可以使用 `-g <groupname>` 为新建的用户定义用户组，还可以使用 `-s <shell path>` 定义用户的 `shell` 程序，使用 `-p <password>` 设置密码。执行了 `useradd` 命令后，你可以再次打开 `/etc/passwd` 查看，你会发现 `raspberrry` 用户位于文件的最后。

```
raspberrry:x:1001:1001:~/home/raspberrry:/bin/sh
```

这说明你已经成功新建了用户，如果没有在 `useradd` 命令中设置密码，可以立即使用 `passwd` 命令为 `raspberrry` 用户设置密码。除了设置密码，如果你还想修改该用户的 `shell` 程序，应该如何操作呢？你可以手动编辑 `/etc/passwd` 文件，或者你可以使用 `usermod` 命令。`usermod` 命令的参数和 `useradd` 命令是一样的，所以我们可以这样做：

```
$ sudo usermod -s /bin/bash raspberrry
```

执行后再查看 `passwd` 文件。

```
raspberrry:x:1001:1001:~/home/raspberrry:/bin/bash
```

好了，到这里你学会了如何在树莓派上创建和配置用户账户，你可以自己尝试一下。

本章小结

本章我们介绍了很多内容，包括 `SSH`、`DHCP` 和 `DNS`。我们还介绍了一些系统函数，介绍了如何使用 `init` 脚本启动守护进程。之后我们学习了 `BASH` 脚本的编写，并尝试编写了我们自己的 `init` 脚本。最后我们介绍了一下基本的 `Linux` 安全规则以及如何创建和配置用户。

读完本章后你可以配置树莓派的网络连接和启动程序，并且还初步学习了一些编程知识。

第8章



搭建 LAMP

使用 Linux 操作系统很重要的目的就是用来搭建 Web 服务器，但是如何做到呢？需要什么硬件？安装什么应用程序？具体步骤是什么？在拥有很多选择的 Linux 系统中，解答起来比较复杂。

本章我们将深入浅出地介绍 Linux 系统的整个环境，重点介绍如何在 Linux 系统上安装、配置和维护 Web 服务器。我们将在树莓派上搭建一个运行于局域网的服务器，介绍如何编写一个简单的网站。阅读完本章后你也可以进一步配置，使其可以连接在互联网中。本章的一些内容甚至可能需要一本书才能详细介绍，限于篇幅我们仅仅介绍如何简单地在树莓派上实现这些功能。

阅读完本章，你会了解什么是 LAMP，这里并不是指床头的台灯¹，而是指 Linux、Apache、MySQL 和 PHP 组成的网站系统。这是搭建网站最简单快速的组合，LAMP 还有一些变种，例如：

- WAMP（Windows、Apache、MySQL 和 PHP）。
- MAMP（Mac OS、Apache、MySQL 和 PHP）。
- 使用 Python 或者 Perl 替代 PHP 表示 P。
- 使用 MariaDB 替代 MySQL 表示 M。

你可以在这么多选择中挑选一个合适的组合，这就是 LAMP 最大的优点，总有一个组合可以运行在你的硬件设备上。在树莓派上使用 LAMP，可以以最小的成本搭建、运行和维护一个网站，下面我们来一步步介绍如何做到。

8.1 第一步

本节我们开始介绍如何搭建一个网站，介绍如何将 PHP 页面和 MySQL 数据库连接起来，动态生成页面。不过还是有一些内容本章无法详细介绍，需要你自己学习和配置。

- 设置 DNS 和域名。
- 高级 SQL 编写和数据库管理。
- 完整的 HTML 知识。
- 完整的 PHP 知识和开发方法。

¹ 译者注：lamp 英文原意为灯。

- JavaScript 知识。
- CSS 知识。

你可以参考一些书籍或者其他资源来学习这些知识，我们将跳过这些内容，下一节开始我们将正式开始学习 LAMP。

8.2 L: Linux 系统

在开始前，我们需要了解 LAMP 中 L（Linux 系统）的一些基础知识，可以阅读之前的章节学习这些内容。

- 在树莓派上安装并配置 Raspbian 操作系统。
- 将内存分配方案设置为 240/16（运行 `sudo raspi-config` 进行设置）。
- 学习如何使用 Raspbian 系统的命令行。
- 基本的编程知识。
- 将树莓派连接入局域网或互联网。

在树莓派上运行 LAMP 没有太多的限制，也不需要很复杂的配置，几乎所有的操作都会在 shell 命令行环境中完成。我们将在其他计算机上查看网站的页面，所以无需使用树莓派的图形化界面。当然如果你喜欢，你也可以在图形化界面中完成所有操作。

8.3 A: Apache 服务器

网站服务器的最重要组成部分就是 web 服务程序，它接收并处理请求，向用户发回 Web 网页。很多 web 服务器都有很强大的功能，可以向很多用户提供服务。在 LAMP 架构中，我们将使用 Apache web 服务器。

Apache 服务器被认为是史上最早出现的 web 服务器，它于 1995 年首次对外发布，至今仍然是互联网中主要的 web 服务器之一，为很多网站提供简单、快速和免费的基础服务，目前有 54%~58% 的互联网网站使用 Apache 服务器。

Apache 这个名称的由来有两种说法。

- 一种是 Apache 服务器起初在开发时被命名为“A patchy Web server”（读音类似 Apache Web server）。
- 另一种是 Apache 服务器以美洲的阿帕奇（Apache）部落命名的。

就像名字一样，Apache 服务器工作起来的确类似一个部落，它通过创建许多工作进程为大量访问者提供服务。介绍了 Apache 服务器的一些小知识后，让我们首先将它安装到树莓派。

我们使用 `apt-get` 工具来执行安装。

```
$ sudo apt-get install apache2
```

```

Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  apache2-mpm-worker apache2-utils apache2.2-bin apache2.2-common libapr1
  libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-ldap ssl-cert
Suggested packages:
  apache2-doc apache2-suexec apache2-suexec-custom openssl-blacklist
The following NEW packages will be installed:
  apache2 apache2-mpm-worker apache2-utils apache2.2-bin apache2.2-common
  libapr1 libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-ldap ssl-cert
0 upgraded, 10 newly installed, 0 to remove and 71 not upgraded.
Need to get 1,348 kB of archives.
After this operation, 4,990 kB of additional disk space will be used.
Do you want to continue [Y/n]?

```

完成安装后，我们需要确认一下 Apache 服务器是否已经启动，我们使用以下命令进行检查。

```

$ ps -ef | grep apache
root      2306      1  0 Sep17 ?        00:00:09 /usr/sbin/apache2 -k start
www-data  2309    2306  0 Sep17 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  2311    2306  0 Sep17 ?        00:00:00 /usr/sbin/apache2 -k start
www-data  2315    2306  0 Sep17 ?        00:00:00 /usr/sbin/apache2 -k start

```

在输出中可以看到 Apache 服务器已经启动，不过这个 Apache 进程列表还有一些需要注意的，你会发现其中有一个进程和其他的有所不同。

```
root 2306 1 0 Sep17 ? 00:00:09 /usr/sbin/apache2 -k start
```

该进程是 Apache 服务器的主进程，它是由 root 用户启动的（进程列表第一栏显示的用户名称），它的 PID 是 2306。其他 Apache 进程的用户名是 www-data，这是 Apache 服务器的专用账户，它们的 PID 均不相同，不过 PPID 都是 2306，说明这些工作进程都是由 Apache 服务器主进程创建的。

Apache 服务器的基本配置就完成了，我们可以做一个简单的测试，让 Apache 服务器显示默认的启动页面。首先我们需要知道系统的 IP 地址，你可以登录到图形化界面中去查看或者使用 ifconfig 命令查看，在 shell 中使用 ifconfig 命令应该会看到类似下面的输出，我们将 IP 地址加粗展示出来。

```

$ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:8a:46:ba
          inet addr:10.0.0.20  Bcast:10.0.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:213812 errors:0 dropped:0 overruns:0 frame:0
          TX packets:5119 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:19226371 (18.3 MiB)  TX bytes:495394 (483.7 KiB)

lo        Link encap:Local Loopback

```

```

inet addr:127.0.0.1 Mask:255.0.0.0
UP LOOPBACK RUNNING MTU:16436 Metric:1
RX packets:8 errors:0 dropped:0 overruns:0 frame:0
TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
collisions:0 txqueuelen:0
RX bytes:1104 (1.0 KiB) TX bytes:1104 (1.0 KiB)

```

如果你的系统和我们的树莓派一样是连接在局域网中，你应该会看到 eth0 设备上的 IP 地址（10.0.0.20），另一个网络设备 lo 是本地回环设备，该设备一直使用 127.0.0.1 的地址。

知道 IP 地址后，我们可以尝试访问 Apache 服务器的默认起始页面，打开浏览器并在地址栏输入 eth0 设备的 IP 地址，如果你想在树莓派系统中访问，两个 IP 地址均可以使用（eth0 设备或者 lo 设备的）。访问后你应该会看到如图 8-1 所示的页面。

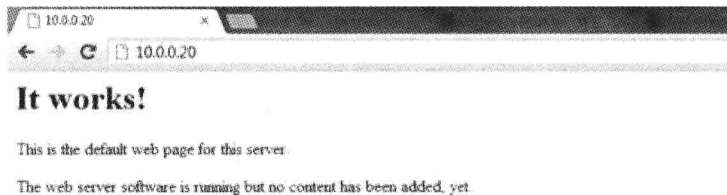


图 8-1 It works

就像你看到的，It works!（它工作得很好！）这说明 Apache 服务器已经运行起来，不过还没有网页可以显示，我们后面会介绍如何添加内容。让我们先来看看 Apache 是如何显示这个默认页面的，了解一些 Apache 服务器的知识。

Apache 服务器配置

Apache 服务器的配置相当复杂，你可以看看/etc/apache2 目录中的内容，其中有多不同作用的子目录，其中的每个文件都不同程度上影响着 Apache 服务器的工作。所以我们先了解一下每个目录中配置文件的作用，这样在调整配置时可以更加容易。

在 Apache 配置目录中首先看到的文件就是 apache2.conf，该文件是主要的配置文件，其中设置了其他配置文件的路径以及读取配置的方式。之后我们需要关注的是 mods-available 和 mods-enabled 目录，mods-available 目录中有许多.load 和.conf 文件，这两种文件分别配置 Apache 服务器从哪里载入模块文件和模块自身的配置。

mods-enabled 目录和 mods-available 目录中的文件类似，如果你使用 ls -l 命令查看该目录中文件的具体情况，你会发现类似以下的状态。

```

$ ls -l /etc/apache2/mods-enabled/
total 0
lrwxrwxrwx 1 root root 28 Sep 17 21:44 alias.conf -> ../mods-available/alias.conf
lrwxrwxrwx 1 root root 28 Sep 17 21:44 alias.load -> ../mods-available/alias.load
lrwxrwxrwx 1 root root 33 Sep 17 21:44 auth_basic.load -> ../mods-available/auth_basic.load
lrwxrwxrwx 1 root root 33 Sep 17 21:44 authn_file.load -> ../mods-available/authn_file.load
...

```

该目录中的文件都是 `mods-available` 目录中对应文件的符号链接，这是因为有很多 Apache 模块可以使用，但是并非每次都载入所有的模块，所以通过创建符号链接，将需要使用的 Apache 模块配置到 `mods-enabled` 目录中，符号链接还有一个优点在于任何一处修改都会两个位置生效。

如果你查看一下 `sites-available` 目录和 `sites-enabled` 目录，你会发现一样的结构。我们刚才看到的默认页面就是在这里配置的，如果修改 `/etc/apache2/sites-available` 目录或者 `/etc/apache2/sites-enabled` 目录中的内容，对于 Apache 服务器的影响是一样的。让我们来看看配置文件具体的内容是什么样的。

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    DocumentRoot /var/www
    <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>
    <Directory /var/www/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
    </Directory>
    ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
    <Directory "/usr/lib/cgi-bin">
        AllowOverride None
        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Order allow,deny
        Allow from all
    </Directory>
    ErrorLog ${APACHE_LOG_DIR}/error.log
    # Possible values include: debug, info, notice, warn, error, crit,
    # alert, emerg.
    LogLevel warn

    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

配置文件的第 1 行说明了该 web 服务器在 80 端口接受请求，你可以通过修改这个端口在同一台计算机上运行多个不同的 Apache 服务器。

 **注意** 如果调整了端口，那么你可以通过在 IP 地址或者域名之后加上 `<port number>` 来访问服务器。

在 `VirtualHost` 中，最重要配置项的就是 `DocumentRoot`，该配置项说明了 Apache 服务器应该从哪里读取页面文件，例如该配置中的 `/var/www`，这表示所有来自 80 端口的请求都使用该目录的内容作为响应。我们打开 `/var/www` 目录，可以看到其中有一个 `index.html` 文件，打开该文件你会看到如下内容。

```
<html><body><h1>It works!</h1>
<p>This is the default web page for this server.</p>
<p>The web server software is running but no content has been added, yet.</p>
</body></html>
```

是不是有点眼熟，这就是之前看到的 Apache 启动页面内容。所以当你访问 Web 服务器时，Apache 实际访问了该目录中的内容，我们来做一些尝试，证明你可以访问该目录中的更多内容。我们执行以下命令。

```
$ echo "Hello World" > /var/www/foo
```

 **注意** 你需要使用 `sudo su` 获得 root 权限来执行该命令。

执行完该命令后，你可以尝试在浏览器中访问 `http://<IP Address>/foo`，你应当会看到刚才我们输入的 Hello World。你可能会问刚才我们访问默认启动页面时为什么没有指定 `index.html`，这是因为 `index` 文件是默认页面，没有其他页面或者没有指定访问页面时将会自动显示其中的内容。

另一个值得关注的是 `Directory` 配置节，这里配置了该目录的访问规则和访问权限。上面的例子配置了两个目录路径，分别是 `/` 目录和 `/var/www` 目录。目录配置可以对许多目录进行设置，如果你希望更多了解如何进行配置，可以访问 Apache 官网并阅读文档。

下一个配置项是 `ScriptAlias`，这里配置了 CGI 脚本的执行程序目录。你可能会想 PHP 脚本是不是通过 CGI 程序执行的？PHP 脚本的确有点像是 BASH 脚本，不过 PHP 脚本并非通过 CGI 程序，而是使用解析模块（`modphp`）解析执行的。

在 `ScriptAlias` 配置项下面又是一些目录配置节，这些目录配置对应于上面的 `ScriptAlias`，例如 `/usr/lib/cgi-bin/`，该目录存储了 `+ExecCGI` 程序，这里还配置了如何通过 CGI 执行请求。配置文件最后是日志文件配置，这里设置了 Apache 服务器应当将日志文件存储在哪个目录以及日志文件的名称。`LogLevel` 则指明需要将什么级别的日志输出，例子中设置的是 `WARN`（警告）级别。

当你使用 PHP 编写网页时，日志在调试时会很有用。下面我们稍微调整一下 Apache 的日志级别配置，看看会有什么变化。修改日志级别为 `notice`，保存文件，之后让 Apache 重新载入配置。由于 Apache 不能自动探测配置的更改，它提供了重新载入的功能，我们在命令行输入以下命令。

```
$ sudo /etc/init.d/apache2 reload
[....] Reloading web server config: apache2apache2: Could not reliably determine the
server's fully qualified domain name, using 127.0.1.1 for ServerName
```

该命令使用 `reload` 参数通知 Apache 服务器重新载入配置文件。下面让我们看一下日志文件的变化，Apache 服务器日志文件位于 `/var/log/apache2` 目录中，打开 `error.log` 文件，你会看到类似以下内容。

```
[Tue Sep 18 22:46:56 2012] [notice] SIGUSR1 received. Doing graceful restart
apache2: Could not reliably determine the server's fully qualified domain name, using 127.0.1.1 for
ServerName
[Tue Sep 18 22:46:56 2012] [notice] Apache/2.2.22 (Debian) configured -- resuming normal operations
```

日志中显示 Apache 服务器收到信号并重启，读取了新的配置文件后立即重新工作起来。我

们还可以查看一下同样位于/var/log/apache2 的 access.log 了解该服务器的访问情况。

```
10.0.0.104 - - [18/Sep/2012:20:31:17 +1000] "GET / HTTP/1.1" 200 482 "-" "Mozilla/5.0 (Windows NT
6.1; WOW64) AppleWebKit/537.10 (KHTML, like Gecko) Chrome/23.0.1262.0 Safari/537.10"
10.0.0.104 - - [18/Sep/2012:22:21:22 +1000] "GET /favicon.ico HTTP/1.1" 404 498 "-" "Mozilla/5.0
(Windows NT 6.1; WOW64) AppleWebKit/537.10 (KHTML, like Gecko) Chrome/23.0.1262.0 Safari/537.10"
10.0.0.104 - - [18/Sep/2012:22:28:36 +1000] "GET /foo HTTP/1.1" 200 274 "-" "Mozilla/5.0 (Windows NT
6.1; WOW64) AppleWebKit/537.10 (KHTML, like Gecko) Chrome/23.0.1262.0 Safari/537.10"
```

这几行日志显示来自 IP 地址为 10.0.0.104 的计算机向 web 服务器的/地址发送了 GET 请求，也就是/var/www 目录的 index.html 文件。除了默认文件的请求，还有对于 foo 文件的请求，这些日志按照时间顺序记录了 web 服务器接受的所有请求，其中还包括了一次尝试访问 favicon.ico 文件，该请求是浏览器试图获得网站的图标，不过我们的文件系统中并不存在该文件。

排错

读到这里你可能会觉得配置 Apache 服务器不太困难，不过当你修改了很多配置之后，就容易出现问題。一旦出现错误，你可以按照以下步骤解决。首先通过 ps -ef 命令检查 Apache 服务器是否已经运行。如果没有，尝试再次启动它。如果仍然失败，检查日志文件中的错误信息，这些错误信息都提供了足够的细节帮助你解决问题。如果仍然没有头绪，可以访问 Apache 官方网站，查询和阅读文档，那里应该有你需要的所有说明。

8.4 M: MySQL

读到这里，我们已经完成了 Web 服务器的设置，你可能会想：“如果把网站的所有内容都写到文件中，通过 Apache 服务器向访问者显示内容，那么一个网站就建成了”。这么想也没错，的确可以完全使用 HTML (HyperText Markup Language) 编写所有的页面，不过这样建立的网站内容是静态的，这意味着每当你需要修改网站内容时，需要打开各个页面文件，修改其中的内容，这样很难创建和管理一个功能较多的网站。

所以本节开始，我们将介绍如何整合 LAMP 架构中的各个组建，以 Apache 作为 Web 服务器，使用 PHP 语言动态生成页面，使用 MySQL 数据库存储呈现给不同用户的不同内容，充分发挥树莓派的性能。

安装 MySQL 数据库

使用数据库系统可以方便地存取数据，可以在其中存储大量的数据，并且通过查询语句得到结构化的输出数据。SQL 数据库的命名来自于 Structured Query Language (结构化查询语言)，所有的 SQL 数据库都使用该语言存储和管理数据。MySQL 数据库是目前最流行的开源关系型数据库 (relational database system)。首先需要在树莓派上安装 MySQL，我们再次使用 apt-get 来安装 mysql-server 软件包。

```

$ sudo apt-get install mysql-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  heirloom-mailx libaio1 libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libmysqlclient16 libnet-daemon-perl libplrpc-perl mysql-client-5.5
  mysql-common mysql-server-5.5 mysql-server-core-5.5
Suggested packages:
  libipc-sharedcache-perl libterm-readkey-perl tinymc
The following NEW packages will be installed:
  heirloom-mailx libaio1 libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libmysqlclient16 libnet-daemon-perl libplrpc-perl mysql-client-5.5
  mysql-common mysql-server mysql-server-5.5 mysql-server-core-5.5
0 upgraded, 13 newly installed, 0 to remove and 84 not upgraded.
Need to get 9,770 kB of archives.
After this operation, 91.5 MB of additional disk space will be used.
Do you want to continue [Y/n]?

```

安装完成后,MySQL 将会向你询问数据库的 root 用户密码¹,这是管理数据库的最重要密码,请记住改密码。完成后,我们通过 ps 命令查看 MySQL 数据库是否已经启动。

```
$ ps -ef | grep mysql
```

你应该会看到类似下面的输出,其中显示了正在运行的 MySQL 守护进程。

```

root      5254      1  0 20:23 ?        00:00:00 /bin/sh /usr/bin/mysqld_safe
mysql     5592  5254  3 20:23 ?        00:00:03 /usr/sbin/mysqld --basedir=/usr --datadir=/var/
lib/mysql --plugin-dir=/usr/lib/mysql/plugin --user=mysql --pid-file=/var/run/mysqld/mysqld.pid
--socket=/var/run/mysqld/mysqld.sock --port=3306
root      5593  5254  0 20:23 ?        00:00:00 logger -t mysqld -p daemon.error

```

可以看到 MySQL 的守护进程已经启动,该进程的启动命令行很长,这也是守护进程的特点。我们不准调整 MySQL 的设置,直接使用默认配置进行后面的操作。下一节我们将介绍如何在数据库中存取数据。

结构化查询语言 SQL

结构化查询语言(SQL)用于在 SQL 数据库中存取和管理数据,可以使用 SQL 编写各种查询语句,SQL 能够写得很简单,也能够写得很复杂,本书不会介绍详细 SQL 的语法,不过我们还是会介绍一些简单的 SQL 语句,能够完成以下的数据库操作功能。

- 基本的数据库管理。
- 存储数据。
- 查找数据。
- 更新数据。

¹ 这里不是 Linux 系统的 root 用户密码,请设置一个新密码。

- 删除数据。

这些例子都很简单，如果你想写出更复杂的 SQL 语句，完成更多的功能，请参考 MySQL 的官方网站文档，学习如何为 MySQL 数据编写复杂的查询语句，或者你也可以阅读更多关于 MySQL 的书籍，更进一步地学习。

下面让我们来建立第一个数据库，创建数据库的 SQL 语法如下：

```
create database <database name>
```

不过，我们首先需要在命令行中启动 mysql 控制台。

```
mysql -uroot -p
```

执行该命令后，MySQL 向你询问 root 密码，输入完成后我们就进入了 mysql 控制台，之后输入 create database pi;，创建一个名为 pi 的数据库。

```
mysql> create database pi;
Query OK, 1 row affected (0.00 sec)
```

我们完成了数据库的创建，不过数据库仅仅是数据库系统的上层结构，我们还需要为数据库创建表 (table)，表是数据存储的逻辑结构，数据在表中按行 (row) 列 (column) 存储，所以我们下一步将会创建 pi 数据库中的表。创建表时，我们需要告诉数据库我们将如何组织数据。

注意 所有的 MySQL 命令都以英文分号 (;) 结尾，所以如果你需要输入多行数据，每次换行不使用分号，而是在整个命令输入完成后输入一个分号。

创建表

我们不准按照常见的软件开发流程来介绍 MySQL 数据库，而是将和数据库相关的所有操作都放在一起介绍，先学习一下如何操作数据库，可以让后面的网站建设事半功倍。

我们将在 MySQL 数据库中存储一个 to-do 任务列表，其中的每个任务条目都包含以下内容。

- 任务的描述 (description)。
- 谁将完成这个任务 (拥有者, owner)。
- 该任务将在何时完成 (日期, date)。
- 任务的地点 (location)。
- 任务的重要性 (importance)。
- 谁创建了该任务 (创建者, creator)。

确定了这些信息后，我们准备创建数据库表结构，在 MySQL 中我们使用以下的语法来创建数据表。

```
create table <tablename> (
  <column name> <column type>,
  <column name> <column type>,
  ...
```

```
<column name> <column type>
);
```

除了基本的表结构，我们还需要为数据表定义一个键（key）。通过键，我们能唯一确定数据表中的某一行数据，数据库表键在网站开发中会非常有用。在数据库中，有多种键，我们将要使用的是主键（primary key），它可以实现唯一确定某一行数据。我们在数据表结构中加上主键 idnumber，得到了以下的 SQL 语句。

```
create table todolist (
  idnumber <column type>,
  description <column type>,
  owner <column type>,
  date <column type>,
  location <column type>,
  importance <column type>,
  creator <column type>
);
```

我们确定了列名称，但仍然没有确定<column type>，这就是每一列的数据类型。MySQL 在我们创建数据表时需要知道每一列的数据类型，这样在存取数据时，MySQL 才能确定是否可以操作相应的数据。MySQL 支持多达 30 种不同的数据类型，不过我们基本只使用其中的 3 种。一是文本类型 VARCHAR，该类型可以定义字符串的最大长度；二是整数类型 INT；最后是时间类型 DATE。根据不同的列，我们使用以上 3 种类型完成我们的 create table 语句。

```
create table todolist (
  idnumber INT,
  description VARCHAR(200),
  owner VARCHAR(40),
  date DATE,
  location VARCHAR(40),
  importance VARCHAR(10),
  creator VARCHAR(40)
);
```

这样基本完成了我们创建数据表的 SQL 语句，不过我们的主键需要保持唯一，并且我们希望它能够自动增长，所以我们为 idnumber 加上 PRIMARY KEY NOT NULL AUTO_INCREMENT 属性，最终我们的 SQL 语句如下：

```
create table todolist (
  idnumber INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
  description VARCHAR(200),
  owner VARCHAR(40),
  date DATE,
  location VARCHAR(40),
  importance VARCHAR(10),
  creator VARCHAR(40)
);
```

不过我们直接执行该语句会发生以下错误。

ERROR 1046 (3D000): No database selected

这是因为我们还没有告诉 MySQL 我们想要操作哪个数据库，MySQL 也不知道应该把表保存到哪个数据库中。我们使用 USE 命令，选择之前建立的 pi 数据库，输入 USE pi; 并回车，MySQL 将告诉我们数据库已被选择。如果你要使用其他的数据库，可以使用 SHOW 命令，通过 SHOW DATABASE; 显示目前 MySQL 中存在的数据库。让我们先完成之前的表创建，再次执行之前的 SQL 语句，你会发现 MySQL 成功地创建了表。

Query OK, 0 rows affected (0.91 sec)

我们通过 SHOW 命令可以查看一下当前 pi 数据库中存在的表。

```
mysql> SHOW TABLES;
+-----+
| Tables_in_pi |
+-----+
| todolist     |
+-----+
1 row in set (0.00 sec)
```

假设你忘记了表结构，可以通过 DESCRIBE 命令让 MySQL 将表结构显示给你。我们输入 DESCRIBE todolist 命令，看看刚刚创建的 todolist 表结构描述。

```
mysql> DESCRIBE todolist;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra      |
+-----+-----+-----+-----+-----+-----+
| idnumber   | int(11)   | NO   | PRI | NULL    | auto_increment |
| description | varchar(200) | YES  |     | NULL    |              |
| owner      | varchar(40) | YES  |     | NULL    |              |
| date       | date      | YES  |     | NULL    |              |
| location   | varchar(40) | YES  |     | NULL    |              |
| importance | varchar(10) | YES  |     | NULL    |              |
| creator    | varchar(40) | YES  |     | NULL    |              |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0.00 sec)
```

可以看到 idnumber 定义为了 PRI（主键），并且拥有自动增长属性，这和我们之前的定义完全相同。到这里我们完成了数据库的创建和数据表的定义，不过在进一步操作数据库之前，我们介绍一些安全性相关的管理命令。假设我们将使用非 root 用户登录并操作数据库，这样可以降低安全风险，不过我们需要给非 root 用户提供数据库的操作权限。这种情况可以使用 GRANT 命令，该命令语法如下：

```
GRANT ALL ON <database>.<table> TO '<username>'@'<user location>' IDENTIFIED BY '<password>';
```

我们为 pi 用户提供 todolist 表的操作权限，该用户的密码设置为 raspberry，我们可以使用以下命令。

```
GRANT ALL ON pi.todolist TO 'pi'@'localhost' IDENTIFIED BY 'raspberry';
```