第七章重定向

In this lesson we are going to unleash what may be the coolest feature of the command line. It’s called I/O redirection. The “I/O” stands for input/output and with this facility you can redirect the input and output of commands to and from files, as well as connect multiple commands together into powerful command pipelines. To show off this facility, we will introduce the following commands:

这堂课，我们来介绍可能是命令行最酷的特性。它叫做 I/O 重定向。”I/O”代表输入/输出， 通过这个工具，你可以重定向命令的输入输出，命令的输入来自文件，而输出也存到文件。 也可以把多个命令连接起来组成一个强大的命令管道。为了展示这个工具，我们将叙述 以下命令：

* cat - Concatenate files
* sort - Sort lines of text
* uniq - Report or omit repeated lines
* grep - Print lines matching a pattern
* wc - Print newline, word, and byte counts for each file
* head - Output the first part of a file
* tail - Output the last part of a file
* tee - Read from standard input and write to standard output and files
* cat － 连接文件
* sort － 排序文本行
* uniq － 报道或省略重复行
* grep － 打印匹配行
* wc － 打印文件中换行符，字，和字节个数
* head － 输出文件第一部分
* tail - 输出文件最后一部分
* tee - 从标准输入读取数据，并同时写到标准输出和文件

**标准输入，输出，和错误**

Many of the programs that we have used so far produce output of some kind. This output often consists of two types. First, we have the program’s results; that is, the data the program is designed to produce, and second, we have status and error messages that tell us how the program is getting along. If we look at a command like ls, we can see that it displays its results and its error messages on the screen.

到目前为止，我们用到的许多程序都会产生某种输出。这种输出，经常由两种类型组成。 第一，程序运行结果；这是说，程序要完成的功能。第二，我们得到状态和错误信息， 这些告诉我们程序进展。如果我们观察一个命令，像 ls，会看到它的运行结果和错误信息 显示在屏幕上。

Keeping with the Unix theme of “everything is a file,” programs such as ls actually send their results to a special file called standard output (often expressed as stdout) and their status messages to another file called standard error (stderr). By default, both standard output and standard error are linked to the screen and not saved into a disk file. In addition, many programs take input from a facility called standard input (stdin) which is, by default, attached to the keyboard.

与 Unix 主题“任何东西都是一个文件”保持一致，程序，比方说 ls，实际上把他们的运行结果 输送到一个叫做标准输出的特殊文件（经常用 stdout 表示），而它们的状态信息则送到另一个 叫做标准错误的文件（stderr）。默认情况下，标准输出和标准错误都连接到屏幕，而不是 保存到磁盘文件。除此之外，许多程序从一个叫做标准输入（stdin）的设备得到输入，默认情况下， 标准输入连接到键盘。

I/O redirection allows us to change where output goes and where input comes from. Normally, output goes to the screen and input comes from the keyboard, but with I/O redirection, we can change that.

I/O 重定向允许我们可以更改输出走向和输入来向。一般地，输出送到屏幕，输入来自键盘， 但是通过 I/O 重定向，我们可以改变输入输出方向。

**重定向标准输出**

I/O redirection allows us to redefine where standard output goes. To redirect standard output to another file besides the screen, we use the “>” redirection operator followed by the name of the file. Why would we want to do this? It’s often useful to store the output of a command in a file. For example, we could tell the shell to send the output of the ls command to the file ls-output.txt instead of the screen:

I/O 重定向允许我们来重定义标准输出送到哪里。重定向标准输出到另一个文件除了屏幕，我们使用 “>” 重定向符，其后跟着文件名。为什么我们要这样做呢？因为有时候把一个命令的运行结果存储到 一个文件很有用处。例如，我们可以告诉 shell 把 ls 命令的运行结果输送到文件 ls-output.txt 中去， 由文件代替屏幕。

[me@linuxbox ~]$ ls -l /usr/bin > ls-output.txt

Here, we created a long listing of the /usr/bin directory and sent the results to the file ls-output.txt. Let’s examine the redirected output of the command:

这里，我们创建了一个长长的目录/usr/bin 列表，并且输送程序运行结果到文件 ls-output.txt 中。 我们检查一下重定向的命令输出结果：

[me@linuxbox ~]$ ls -l ls-output.txt

-rw-rw-r-- 1 me me 167878 2008-02-01 15:07 ls-output.txt

Good; a nice, large, text file. If we look at the file with less, we will see that the file ls-output.txt does indeed contain the results from our ls command:

好；一个不错的大型文本文件。如果我们用 less 阅读器来查看这个文件，我们会看到文件 ls-output.txt 的确包含 ls 命令的执行结果。

[me@linuxbox ~]$ less ls-output.txt

Now, let’s repeat our redirection test, but this time with a twist. We’ll change the name of the directory to one that does not exist:

现在，重复我们的重定向测试，但这次有改动。我们把目录换成一个不存在的目录。

[me@linuxbox ~]$ ls -l /bin/usr > ls-output.txt

ls: cannot access /bin/usr: No such file or directory

We received an error message. This makes sense since we specified the non-existent directory /bin/usr, but why was the error message displayed on the screen rather than being redirected to the file ls-output.txt? The answer is that the ls program does not send its error messages to standard output. Instead, like most well-written Unix programs, it sends its error messages to standard error. Since we only redirected standard output and not standard error, the error message was still sent to the screen. We’ll see how to redirect standard error in just a minute, but first, let’s look at what happened to our output file:

我们收到一个错误信息。这讲得通，因为我们指定了一个不存在的目录/bin/usr, 但是为什么这条错误信息显示在屏幕上而不是被重定向到文件 ls-output.txt？答案是， ls 程序不把它的错误信息输送到标准输出。反而，像许多写得不错的 Unix 程序，ls 把 错误信息送到标准错误。因为我们只是重定向了标准输出，而没有重定向标准错误， 所以错误信息被送到屏幕。马上，我们将知道怎样重定向标准错误，但是首先看一下 我们的输出文件发生了什么事情。

me@linuxbox ~]$ ls -l ls-output.txt

-rw-rw-r-- 1 me me 0 2008-02-01 15:08 ls-output.txt

The file now has zero length! This is because, when we redirect output with the “>” redirection operator, the destination file is always rewritten from the beginning. Since our ls command generated no results and only an error message, the redirection operation started to rewrite the file and then stopped because of the error, resulting in its truncation. In fact, if we ever need to actually truncate a file (or create a new, empty file) we can use a trick like this:

文件长度为零！这是因为，当我们使用 “>” 重定向符来重定向输出结果时，目标文件总是从开头被重写。 因为我们 ls 命令没有产生运行结果，只有错误信息，重定向操作开始重写文件，然后 由于错误而停止，导致文件内容删除。事实上，如果我们需要删除一个文件内容（或者创建一个 新的空文件），可以使用这样的技巧：

[me@linuxbox ~]$ > ls-output.txt

Simply using the redirection operator with no command preceding it will truncate an existing file or create a new, empty file.

简单地使用重定向符，没有命令在它之前，这会删除一个已存在文件的内容或是 创建一个新的空文件。

So, how can we append redirected output to a file instead of overwriting the file from the beginning? For that, we use the “>>” redirection operator, like so:

所以，怎样才能把重定向结果追加到文件内容后面，而不是从开头重写文件？为了这个目的， 我们使用”>>“重定向符，像这样：

[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt

Using the “>>” operator will result in the output being appended to the file. If the file does not already exist, it is created just as though the “>” operator had been used. Let’s put it to the test:

使用”>>“操作符，将导致输出结果添加到文件内容之后。如果文件不存在，文件会 被创建，就如使用了”>”操作符。把它放到测试中：

[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt

[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt

[me@linuxbox ~]$ ls -l /usr/bin >> ls-output.txt

[me@linuxbox ~]$ ls -l ls-output.txt

-rw-rw-r-- 1 me me 503634 2008-02-01 15:45 ls-output.txt

We repeated the command three times resulting in an output file three times as large.

我们重复执行命令三次，导致输出文件大小是原来的三倍。

**重定向标准错误**

Redirecting standard error lacks the ease of a dedicated redirection operator. To redirect standard error we must refer to its file descriptor. A program can produce output on any of several numbered file streams. While we have referred to the first three of these file streams as standard input, output and error, the shell references them internally as file descriptors zero, one and two, respectively. The shell provides a notation for redirecting files using the file descriptor number. Since standard error is the same as file descriptor number two, we can redirect standard error with this notation:

重定向标准错误缺乏专用的重定向操作符。重定向标准错误，我们必须参考它的文件描述符。 一个程序可以在几个编号的文件流中的任一个上产生输出。然而我们必须把这些文件流的前 三个看作标准输入，输出和错误，shell 内部参考它们为文件描述符0，1和2，各自地。shell 提供 了一种表示法来重定向文件，使用文件描述符。因为标准错误和文件描述符2一样，我们用这种 表示法来重定向标准错误：

[me@linuxbox ~]$ ls -l /bin/usr 2> ls-error.txt

The file descriptor “2” is placed immediately before the redirection operator to perform the redirection of standard error to the file ls-error.txt.

文件描述符”2”，紧挨着放在重定向操作符之前，来执行重定向标准错误到文件 ls-error.txt 任务。

**重定向标准输出和错误到同一个文件**

There are cases in which we may wish to capture all of the output of a command to a single file. To do this, we must redirect both standard output and standard error at the same time. There are two ways to do this. First, the traditional way, which works with old versions of the shell:

可能有这种情况，我们希望捕捉一个命令的所有输出到一个文件。为了完成这个，我们 必须同时重定向标准输出和标准错误。有两种方法来完成任务。第一个，传统的方法， 在旧版本 shell 中也有效：

[me@linuxbox ~]$ ls -l /bin/usr > ls-output.txt 2>&1

Using this method, we perform two redirections. First we redirect standard output to the file ls-output.txt and then we redirect file descriptor two (standard error) to file descriptor one (standard output) using the notation 2>&1.

使用这种方法，我们完成两个重定向。首先重定向标准输出到文件 ls-output.txt，然后 重定向文件描述符2（标准错误）到文件描述符1（标准输出）使用表示法2>&1。

Notice that the order of the redirections is significant. The redirection of standard error must always occur after redirecting standard output or it doesn’t work. In the example above,

注意重定向的顺序安排非常重要。标准错误的重定向必须总是出现在标准输出 重定向之后，要不然它不起作用。上面的例子，

>ls-output.txt 2>&1

redirects standard error to the file ls-output.txt, but if the order is changed to

重定向标准错误到文件 ls-output.txt，但是如果命令顺序改为：

2>&1 >ls-output.txt

standard error is directed to the screen.

则标准错误定向到屏幕。

Recent versions of bash provide a second, more streamlined method for performing this combined redirection:

现在的 bash 版本提供了第二种方法，更精简合理的方法来执行这种联合的重定向。

[me@linuxbox ~]$ ls -l /bin/usr &> ls-output.txt

In this example, we use the single notation &> to redirect both standard output and standard error to the file ls-output.txt.

在这个例子里面，我们使用单单一个表示法 &> 来重定向标准输出和错误到文件 ls-output.txt。

**处理不需要的输出**

Sometimes “silence is golden,” and we don’t want output from a command, we just want to throw it away. This applies particularly to error and status messages. The system provides a way to do this by redirecting output to a special file called “/dev/null”. This file is a system device called a bit bucket which accepts input and does nothing with it. To suppress error messages from a command, we do this:

有时候“沉默是金”，我们不想要一个命令的输出结果，只想把它们扔掉。这种情况 尤其适用于错误和状态信息。系统为我们提供了解决问题的方法，通过重定向输出结果 到一个特殊的叫做”/dev/null”的文件。这个文件是系统设备，叫做位存储桶，它可以 接受输入，并且对输入不做任何处理。为了隐瞒命令错误信息，我们这样做：

[me@linuxbox ~]$ ls -l /bin/usr 2> /dev/null

/dev/null in Unix Culture

Unix 文化中的/dev/null

The bit bucket is an ancient Unix concept and due to its universality, has appeared in many parts of Unix culture. When someone says he/she is sending your comments to /dev/null, now you know what it means. For more examples, see the Wikipedia article on “/dev/null”.

位存储桶是个古老的 Unix 概念，由于它的普遍性，它的身影出现在 Unix 文化的 许多部分。当有人说他/她正在发送你的评论到/dev/null，现在你应该知道那是 什么意思了。更多的例子，可以阅读 Wikipedia 关于”/dev/null”的文章。

**重定向标准输入**

Up to now, we haven’t encountered any commands that make use of standard input (actually we have, but we’ll reveal that surprise a little bit later), so we need to introduce one.

到目前为止，我们还没有遇到一个命令是利用标准输入的（实际上我们遇到过了，但是 一会儿再揭晓谜底），所以我们需要介绍一个。

**cat － 连接文件**

The cat command reads one or more files and copies them to standard output like so:

cat 命令读取一个或多个文件，然后复制它们到标准输出，就像这样:

cat [file]

In most cases, you can think of cat as being analogous to the TYPE command in DOS. You can use it to display files without paging, for example:

在大多数情况下，你可以认为 cat 命令相似于 DOS 中的 TYPE 命令。你可以使用 cat 来显示 文件而没有分页，例如：

[me@linuxbox ~]$ cat ls-output.txt

will display the contents of the file ls-output.txt. cat is often used to display short text files. Since cat can accept more than one file as an argument, it can also be used to join files together. Say we have downloaded a large file that has been split into multiple parts (multimedia files are often split this way on USENET), and we want to join them back together. If the files were named:

将会显示文件 ls-output.txt 的内容。cat 经常被用来显示简短的文本文件。因为 cat 可以 接受不只一个文件作为参数，所以它也可以用来把文件连接在一起。比方说我们下载了一个 大型文件，这个文件被分离成多个部分（USENET 中的多媒体文件经常以这种方式分离）， 我们想把它们连起来。如果文件命名为：

movie.mpeg.001 movie.mpeg.002 … movie.mpeg.099

we could join them back together with this command:

我们能用这个命令把它们连接起来：

cat movie.mpeg.0\* > movie.mpeg

Since wildcards always expand in sorted order, the arguments will be arranged in the correct order.

因为通配符总是以有序的方式展开，所以这些参数会以正确顺序安排。

This is all well and good, but what does this have to do with standard input? Nothing yet, but let’s try something else. What happens if we type “cat” with no arguments:

这很好，但是这和标准输入有什么关系呢？没有任何关系，让我们试着做些其他的工作。 如果我们输入不带参数的”cat”命令，会发生什么呢：

[me@linuxbox ~]$ cat

Nothing happens, it just sits there like it’s hung. It may seem that way, but it’s really doing exactly what it’s supposed to.

没有发生任何事情，它只是坐在那里，好像挂掉了一样。看起来是那样，但是它正在做它该做的事情：

If cat is not given any arguments, it reads from standard input and since standard input is, by default, attached to the keyboard, it’s waiting for us to type something! Try this:

如果 cat 没有给出任何参数，它会从标准输入读入数据，因为标准输入，默认情况下，连接到键盘。 它正在等待我们输入数据！试试这个：

[me@linuxbox ~]$ cat

The quick brown fox jumped over the lazy dog.

Next, type a Ctrl-d (i.e., hold down the Ctrl key and press “d”) to tell cat that it has reached end of file (EOF) on standard input:

下一步，输入 Ctrl-d（按住 Ctrl 键同时按下”d”），来告诉 cat，在标准输入中， 它已经到达文件末尾（EOF）：

[me@linuxbox ~]$ cat

The quick brown fox jumped over the lazy dog.

The quick brown fox jumped over the lazy dog.

In the absence of filename arguments, cat copies standard input to standard output, so we see our line of text repeated. We can use this behavior to create short text files. Let’s say that we wanted to create a file called “lazy\_dog.txt” containing the text in our example. We would do this:

由于没有文件名参数，cat 复制标准输入到标准输出，所以我们看到文本行重复出现。 我们可以使用这种行为来创建简短的文本文件。比方说，我们想创建一个叫做”lazy\_dog.txt” 的文件，这个文件包含例子中的文本。我们这样做：

[me@linuxbox ~]$ cat > lazy\_dog.txt

The quick brown fox jumped over the lazy dog.

Type the command followed by the text we want in to place in the file. Remember to type Ctrl-d at the end. Using the command line, we have implemented the world’s dumbest word processor! To see our results, we can use cat to copy the file to stdout again:

输入命令，其后输入要放入文件中的文本。记住，最后输入 Ctrl-d。通过使用这个命令，我们 实现了世界上最低能的文字处理器！看一下运行结果，我们使用 cat 来复制文件内容到 标准输出：

[me@linuxbox ~]$ cat lazy\_dog.txt

The quick brown fox jumped over the lazy dog.

Now that we know how cat accepts standard input, in addition to filename arguments, let’s try redirecting standard input:

现在我们知道怎讲接受标准输入，除了文件名参数，让我们试着重定向标准输入：

[me@linuxbox ~]$ cat < lazy\_dog.txt

The quick brown fox jumped over the lazy dog.

Using the “<” redirection operator, we change the source of standard input from the keyboard to the file lazy\_dog.txt. We see that the result is the same as passing a single filename argument. This is not particularly useful compared to passing a filename argument, but it serves to demonstrate using a file as a source of standard input. Other commands make better use of standard input, as we shall soon see.

使用“<”重定向操作符，我们把标准输入源从键盘改到文件 lazy\_dog.tx。我们看到结果 和传递单个文件名作为参数的执行结果一样。把这和传递一个文件名参数作比较，不是特别有意义， 但它是用来说明把一个文件作为标准输入源。有其他的命令更好地利用了标准输入，我们不久将会看到。

Before we move on, check out the man page for cat, as it has several interesting options.

在我们继续之前，查看 cat 的手册页，因为它有几个有趣的选项。

**管道线**

The ability of commands to read data from standard input and send to standard output is utilized by a shell feature called pipelines. Using the pipe operator “|” (vertical bar), the standard output of one command can be piped into the standard input of another:

命令可以从标准输入读取数据，然后再把数据输送到标准输出，命令的这种能力被 一个 shell 特性所利用，这个特性叫做管道线。使用管道操作符”|”（竖杠），一个命令的 标准输出可以管道到另一个命令的标准输入：

command1 | command2

To fully demonstrate this, we are going to need some commands. Remember how we said there was one we already knew that accepts standard input? It’s less. We can use less to display, page-by-page, the output of any command that sends its results to standard output:

为了全面地说明这个命令，我们需要一些命令。是否记得我们说过，我们已经知道有一个 命令接受标准输入？它是 less 命令。我们用 less 来一页一页地显示任何命令的输出，命令把 它的运行结果输送到标准输出：

[me@linuxbox ~]$ ls -l /usr/bin | less

This is extremely handy! Using this technique, we can conveniently examine the output of any command that produces standard output.

这极其方便！使用这项技术，我们可以方便地检测会产生标准输出的任一命令的运行结果。

**过滤器**

Pipelines are often used to perform complex operations on data. It is possible to put several commands together into a pipeline. Frequently, the commands used this way are referred to as filters. Filters take input, change it somehow and then output it. The first one we will try is sort. Imagine we wanted to make a combined list of all of the executable programs in /bin and /usr/bin, put them in sorted order and view it:

管道线经常用来对数据完成复杂的操作。有可能会把几个命令放在一起组成一个管道线。 通常，以这种方式使用的命令被称为过滤器。过滤器接受输入，以某种方式改变它，然后 输出它。第一个我们想试验的过滤器是 sort。想象一下，我们想把目录/bin 和/usr/bin 中 的可执行程序都联合在一起，再把它们排序，然后浏览执行结果：

[me@linuxbox ~]$ ls /bin /usr/bin | sort | less

Since we specified two directories (/bin and /usr/bin), the output of ls would have consisted of two sorted lists, one for each directory. By including sort in our pipeline, we changed the data to produce a single, sorted list.

因为我们指定了两个目录（/bin 和/usr/bin），ls 命令的输出结果由有序列表组成， 各自针对一个目录。通过在管道线中包含 sort，我们改变输出数据，从而产生一个 有序列表。

**uniq - 报道或忽略重复行**

The uniq command is often used in conjunction with sort. uniq accepts a sorted list of data from either standard input or a single filename argument (see the uniq man page for details) and, by default, removes any duplicates from the list. So, to make sure our list has no duplicates (that is, any programs of the same name that appear in both the /bin and /usr/bin directories) we will add uniq to our pipeline:

uniq 命令经常和 sort 命令结合在一起使用。uniq 从标准输入或单个文件名参数接受数据有序 列表（详情查看 uniq 手册页），默认情况下，从数据列表中删除任何重复行。所以，为了确信 我们的列表中不包含重复句子（这是说，出现在目录/bin 和/usr/bin 中重名的程序），我们添加 uniq 到我们的管道线中：

[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq | less

In this example, we use uniq to remove any duplicates from the output of the sort command. If we want to see the list of duplicates instead, we add the “-d” option to uniq like so:

在这个例子中，我们使用 uniq 从 sort 命令的输出结果中，来删除任何重复行。如果我们想看到 重复的数据列表，让 uniq 命令带上”-d”选项，就像这样：

[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq -d | less

**wc － 打印行，字和字节数**

The wc (word count) command is used to display the number of lines, words, and bytes contained in files. For example:

wc（字计数）命令是用来显示文件所包含的行，字和字节数。例如：

[me@linuxbox ~]$ wc ls-output.txt

7902 64566 503634 ls-output.txt

In this case it prints out three numbers: lines, words, and bytes contained in ls- output.txt. Like our previous commands, if executed without command line arguments, wc accepts standard input. The “-l” option limits its output to only report lines. Adding it to a pipeline is a handy way to count things. To see the number of programs we have in our sorted list, we can do this:

在这个例子中，wc 打印出来三个数字：包含在文件 ls-output.txt 中的行数，单词数和字节数， 正如我们先前的命令，如果 wc 不带命令行参数，它接受标准输入。”-l”选项限制命令输出只能 报道行数。添加 wc 到管道线来统计数据，是个很便利的方法。查看我们的有序列表中程序个数， 我们可以这样做：

[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq | wc -l

2728

**grep － 打印匹配行**

grep is a powerful program used to find text patterns within files. It’s used like this:

grep 是个很强大的程序，用来找到文件中的匹配文本。这样使用 grep 命令：

grep pattern [file...]

When grep encounters a “pattern” in the file, it prints out the lines containing it. The patterns that grep can match can be very complex, but for now we will concentrate on simple text matches. We’ll cover the advanced patterns, called regular expressions in a later chapter.

当 grep 遇到一个文件中的匹配”模式”，它会打印出包含这个类型的行。grep 能够匹配的模式可以 很复杂，但是现在我们把注意力集中在简单文本匹配上面。在后面的章节中，我们将会研究 高级模式，叫做正则表达式。

Let’s say we want to find all the files in our list of programs that had the word “zip” embedded in the name. Such a search might give us an idea of some of the programs on our system that had something to do with file compression. We would do this:

比如说，我们想在我们的程序列表中，找到文件名中包含单词”zip”的所有文件。这样一个搜索， 可能让我们了解系统中的一些程序与文件压缩有关系。这样做：

[me@linuxbox ~]$ ls /bin /usr/bin | sort | uniq | grep zip

bunzip2

bzip2

gunzip

...

There are a couple of handy options for grep: “-i” which causes grep to ignore case when performing the search (normally searches are case sensitive) and “-v” which tells grep to only print lines that do not match the pattern.

grep 有－对方便的选项：”-i”导致 grep 忽略大小写当执行搜索时（通常，搜索是大小写 敏感的），”-v”选项会告诉 grep 只打印不匹配的行。

**head / tail － 打印文件开头部分/结尾部分**

Sometimes you don’t want all of the output from a command. You may only want the first few lines or the last few lines. The head command prints the first ten lines of a file and the tail command prints the last ten lines. By default, both commands print ten lines of text, but this can be adjusted with the “-n” option:

有时候你不需要一个命令的所有输出。可能你只想要前几行或者后几行的输出内容。 head 命令打印文件的前十行，而 tail 命令打印文件的后十行。默认情况下，两个命令 都打印十行文本，但是可以通过”-n”选项来调整命令打印的行数。

[me@linuxbox ~]$ head -n 5 ls-output.txt

total 343496

...

[me@linuxbox ~]$ tail -n 5 ls-output.txt

...

These can be used in pipelines as well:

它们也能用在管道线中：

[me@linuxbox ~]$ ls /usr/bin | tail -n 5

znew

...

tail has an option which allows you to view files in real-time. This is useful for watching the progress of log files as they are being written. In the following example, we will look at the messages file in /var/log. Superuser privileges are required to do this on some Linux distributions, since the /var/log/messages file may contain security information:

tail 有一个选项允许你实时的浏览文件。当观察日志文件的进展时，这很有用，因为 它们同时在被写入。在以下的例子里，我们要查看目录/var/log 里面的信息文件。在 一些 Linux 发行版中，要求有超级用户权限才能阅读这些文件，因为文件/var/log/messages 可能包含安全信息。

[me@linuxbox ~]$ tail -f /var/log/messages

Feb 8 13:40:05 twin4 dhclient: DHCPACK from 192.168.1.1

....

Using the “-f” option, tail continues to monitor the file and when new lines are appended, they immediately appear on the display. This continues until you type Ctrl-c.

使用”-f”选项，tail 命令继续监测这个文件，当新的内容添加到文件后，它们会立即 出现在屏幕上。这会一直继续下去直到你输入 Ctrl-c。

**tee － 从 Stdin 读取数据，并同时输出到 Stdout 和文件**

In keeping with our plumbing metaphor, Linux provides a command called tee which creates a “tee” fitting on our pipe. The tee program reads standard input and copies it to both standard output (allowing the data to continue down the pipeline) and to one or more files. This is useful for capturing a pipeline’s contents at an intermediate stage of processing. Here we repeat one of our earlier examples, this time including tee to capture the entire directory listing to the file ls.txt before grep filters the pipeline’s contents:

为了和我们的管道隐喻保持一致，Linux 提供了一个叫做 tee 的命令，这个命令制造了 一个”tee”，安装到我们的管道上。tee 程序从标准输入读入数据，并且同时复制数据 到标准输出（允许数据继续随着管道线流动）和一个或多个文件。当在某个中间处理 阶段来捕捉一个管道线的内容时，这很有帮助。这里，我们重复执行一个先前的例子， 这次包含 tee 命令，在 grep 过滤管道线的内容之前，来捕捉整个目录列表到文件 ls.txt：

[me@linuxbox ~]$ ls /usr/bin | tee ls.txt | grep zip

bunzip2

bzip2

....

**总结归纳**

As always, check out the documentation of each of the commands we have covered in this chapter. We have only seen their most basic usage. They all have a number of interesting options. As we gain Linux experience, we will see that the redirection feature of the command line is extremely useful for solving specialized problems. There are many commands that make use of standard input and output, and almost all command line programs use standard error to display their informative messages.

一如既往，查看这章学到的每一个命令的文档。我们已经知道了他们最基本的用法。 它们还有很多有趣的选项。随着我们 Linux 经验的积累，我们会了解命令行重定向特性 在解决特殊问题时非常有用处。有许多命令利用标准输入和输出，而几乎所有的命令行 程序都使用标准错误来显示它们的详细信息。

Linux Is About Imagination

Linux 可以激发我们的想象

When I am asked to explain the difference between Windows and Linux, I often use a toy analogy.

当我被要求解释 Windows 与 Linux 之间的差异时，我经常拿玩具来作比喻。

Windows is like a Game Boy. You go to the store and buy one all shiny new in the box. You take it home, turn it on and play with it. Pretty graphics, cute sounds. After a while though, you get tired of the game that came with it so you go back to the store and buy another one. This cycle repeats over and over. Finally, you go back to the store and say to the person behind the counter, “I want a game that does this!” only to be told that no such game exists because there is no “market demand” for it. Then you say, “But I only need to change this one thing!” The person behind the counter says you can’t change it. The games are all sealed up in their cartridges. You discover that your toy is limited to the games that others have decided that you need and no more.

Windows 就像一个游戏机。你去商店，买了一个包装在盒子里面的全新的游戏机。 你把它带回家，打开盒子，开始玩游戏。精美的画面，动人的声音。玩了一段时间之后， 你厌倦了它自带的游戏，所以你返回商店，又买了另一个游戏机。这个过程反复重复。 最后，你玩腻了游戏机自带的游戏，你回到商店，告诉售货员，“我想要一个这样的游戏！” 但售货员告诉你没有这样的游戏存在，因为它没有“市场需求”。然后你说，“但是我只 需要修改一下这个游戏！“，售货员又告诉你不能修改它。所有游戏都被封装在它们的 存储器中。到头来，你发现你的玩具只局限于别人为你规定好的游戏。

Linux, on the other hand, is like the world’s largest Erector Set. You open it up and it’s just a huge collection of parts. A lot of steel struts, screws, nuts, gears, pulleys, motors, and a few suggestions on what to build. So you start to play with it. You build one of the suggestions and then another. After a while you discover that you have your own ideas of what to make. You don’t ever have to go back to the store, as you already have everything you need. The Erector Set takes on the shape of your imagination. It does what you want.

另一方面，Linux 就像一个全世界上最大的建造模型。你打开它，发现它只是一个巨大的 部件集合。有许多钢支柱，螺钉，螺母，齿轮，滑轮，发动机，和一些怎样来建造它的说明书。 然后你开始摆弄它。你建造了一个又一个样板模型。过了一会儿，你发现你要建造自己的模型。 你不必返回商店，因为你已经拥有了你需要的一切。建造模型以你构想的形状为模板，搭建 你想要的模型。

Your choice of toys is, of course, a personal thing, so which toy would you find more satisfying?

当然，选择哪一个玩具，是你的事情，那么你觉得哪个玩具更令人满意呢？