

23 流和输出重定向，心之所向

更新时间：2019-07-15 15:18:37



“

每个人的生命都是一只小船，理想是小船的风帆。

——张海迪

”

内容简介

1. 前言
2. “> 和 >>”：重定向到文件
3. “2>, 2>>, 2>&1”：重定向错误输出
4. 总结

1. 前言

上一课数据处理，慢条斯理，我们学习了不少文件和数据处理的命令。

这一课和下一课，我们将一起来学一些非常有用的内容：流、管道、重定向，三管齐下。

这两课会相当有意思，内容也很多，而且有可能颠覆你的三“观”（毕竟三管齐下，不颠覆三观也难）。

这三个名称，听上去就怪怪的。什么流，管道，重定向，都是什么啊。不过相信学完这节课和下一课，大家能够有拨云见雾的感觉。

到目前为止，我们已经学习了不少 Linux 的命令了，也已经比较熟悉命令行的用法了。其最基本用法是这样的：

1. 在终端输入命令（比如输入 ls 命令）
2. 命令的运行结果显示在终端中

但是我们还不知道的是：其实我们可以重定向命令的运行结果。

重定向，是什么意思呢？简单来说，就是我们可以把本来要显示在终端的命令结果，输送到别的地方：到文件中或者作为其他命令的输入（命令的链接，或者叫命令管道）。

把两个命令连起来使用，一个命令的输出作为另一个命令的输入，这就构成了管道。

管道的英语是 `pipeline`。你可以想象一个个水管，连接起来，一个水管流出来的水（输出），如果接上另一个水管，水是不是就流入另一个水管，成为另一个水管的输入了？

当然了，在计算机科学中，流（英语是 `stream`）的含义是比较难理解的，也比较丰富，不同的情况下含义也不太一样，如果把它比作水流可能不完全。

知乎上就有一篇帖子，大家可以看看：[如何理解编程语言中「流」（stream）的概念？](#)

我们常可以看到这样的字样：位元流，字节流，资料流，视频流，音频流，流媒体，流算法，流处理，数据流挖掘等等。

在维基百科中，流的简单定义是这样的，供大家参考：

In computer science, a stream is a sequence of data elements made available over time. A stream can be thought of as items on a conveyor belt being processed one at a time rather than in large batches.

以上这段英语翻译过来大致意思是（有点难翻译... 我尽力了）：

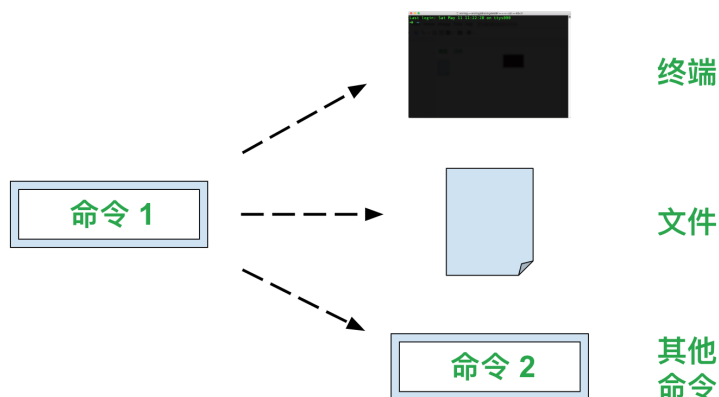
在计算机科学中，流是时间上可用的一系列数据元素。我们可以把流比喻成传送带上的物件，每个时间点传输一个，而不是多个打包传输。

还是不太容易理解对吧？没事，学完这课和下一课应该就理解了，因为有大量实例和图解帮助你。

前面说了，我们会学习如何把命令的输出结果重定向到其他地方：

1. 哪里：文件或者另一个命令的输入
2. 如何实现：通过在命令间插入特定的符号（这些符号可以被称为“重定向流”符号）。下面我们会学到，有好几种符号。

可以用下图来做一个小结：



可以看到，命令的输出可以有三个去向：

- 终端
- 文件
- 其他命令的输入

到目前为止，我们只用过第一种形式：把命令的输出结果显示在终端。

我们岂能就此罢休呢，肯定要乘胜追击啊，一不做二不休啊。

重定向流从 Unix 时代就已经是很重要的概念了，后来 Linux 出现，重定向流的原理依旧被沿用。

重定向流将会改变我们看待终端命令行的方式，所以这课很重要。

借着这一课，我们将把我们的命令行的造诣提升到另一个层级。如果之前是命令行的“小学”，那么这一课学完就基本可以“初中毕业”了。

你也许看到过一些 Linux 高手输入命令，经常会输入一大串，然后回车。这一大串命令中很可能就用到了我们学的流、管道和重定向。

2. “> 和 >>”：重定向到文件

我们先从最简单的开始。最简单的操作就是把命令的输出结果重定向到文件中，就不会在终端显示命令运行结果了。

准备工作：再谈 cut 命令

在开始学习 > 和 >> 这两个符号的用法之前，我们需要创建一些文件。

在创建文件之前，我们来谈谈 cut 命令的进阶用法。cut 是英语“剪切”的意思，用于从文件中剪切出一部分内容。

上一课中我们学习了 cut 命令，但没有深入讲解，只讲了基本的用法（-c 参数：根据字符数来剪切）。

现在让我们学习一下 cut 命令的其他参数。

cut 命令进阶：根据分隔符来剪切

首先，我们来看一种特殊的文件形式：CSV 格式。

CSV 是 Comma Separated Values 的缩写，翻成中文是“逗号分隔值”。

以下摘自[百度百科](#)：

逗号分隔值（Comma-Separated Values，CSV，有时也称为字符分隔值，因为分隔字符也可以不是逗号），其文件以纯文本形式存储表格数据（数字和文本）。

纯文本意味着该文件是一个字符序列，不含必须像二进制数字那样被解读的数据。

CSV 文件由任意数目的记录组成，记录间以某种换行符分隔；每条记录由字段组成，字段间的分隔符是其它字符或字符串，最常见的是逗号或制表符。

通常，所有记录都有完全相同的字段序列。

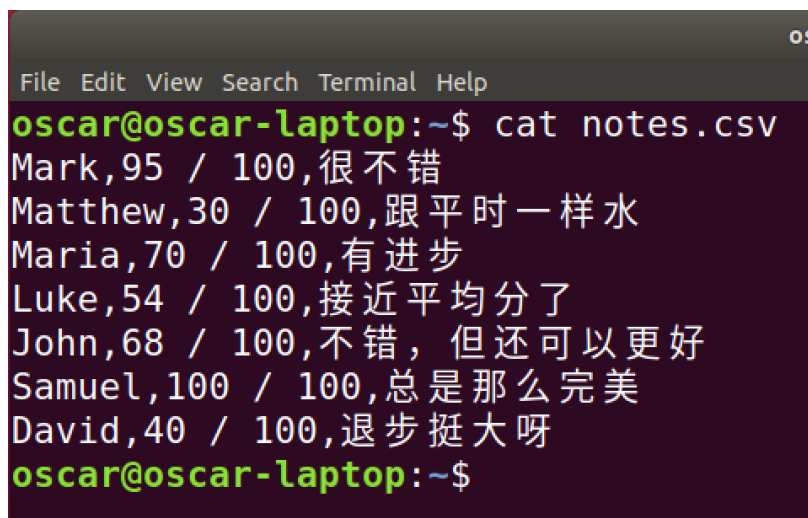
CSV 文件的后缀名是 .csv，通常可以被 Excel 等软件打开，打开之后会把分隔符隔开的各个数值填充到表格里。

我们来构建一个 CSV 文件，因为要用逗号作为分隔符，来学习 cut 命令的进阶使用：根据分隔符来剪切。

假设我们有一个人数不多的班级，作为老师我们需要统计新近一次考试的成绩。为此我们制作了一个表格，并按照顺序把数据写入表格。

我们的 CSV 文件的内容可以如下，我们用文本编辑器来编写这个 CSV 文件，并且取名：notes.csv（note 是英语“成绩”的意思）。

```
Mark,95 / 100,很不错
Matthew,30 / 100,跟平时一样水
Maria,70 / 100,有进步
Luke,54 / 100,接近平均分
John,68 / 100,不错，但还可以更好
Samuel,100 / 100,总是那么完美
David,40 / 100,退步挺大呀
```

A terminal window with a dark background and light green text. The window title is "os". The menu bar shows "File Edit View Search Terminal Help". The prompt is "oscar@oscar-laptop:~\$". The command "cat notes.csv" has been executed, displaying the contents of the file: "Mark,95 / 100,很不错", "Matthew,30 / 100,跟平时一样水", "Maria,70 / 100,有进步", "Luke,54 / 100,接近平均分", "John,68 / 100,不错，但还可以更好", "Samuel,100 / 100,总是那么完美", "David,40 / 100,退步挺大呀". The prompt is now "oscar@oscar-laptop:~\$".

```
oscar@oscar-laptop:~$ cat notes.csv
Mark,95 / 100,很不错
Matthew,30 / 100,跟平时一样水
Maria,70 / 100,有进步
Luke,54 / 100,接近平均分
John,68 / 100,不错，但还可以更好
Samuel,100 / 100,总是那么完美
David,40 / 100,退步挺大呀
oscar@oscar-laptop:~$
```

上面的“跟平时一样水”中的“水”表示“不怎么样，很一般，有点糟糕”。

如上图所示，我们的 notes.csv 文件中每一行由三部分组成，每部分由一个逗号分隔：

- 学生名字
- 成绩（满分是 100 分）
- 评语

现在假如我们要从 notes.csv 文件中提取名字那一列，怎么办呢？我们不能用 cut 命令的 -c 参数啊，毕竟每个名字的字符数不相等。

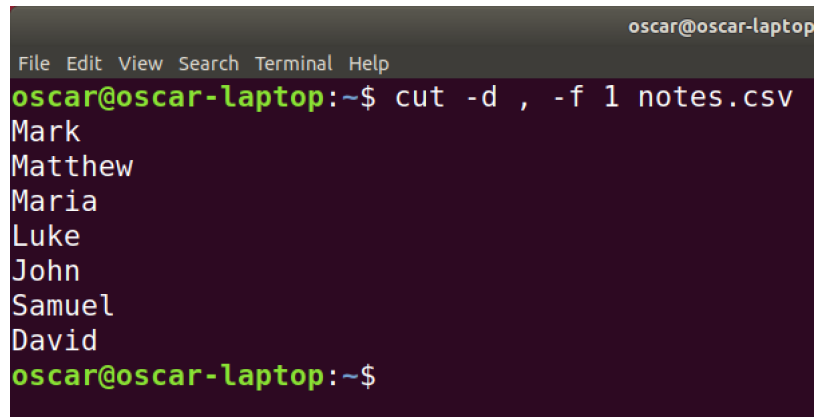
聪明如你应该想到了。是的，我们看到文件中每一行的每一部分是用分隔符来隔开的，所以我们可以这样做：

需要用到两个参数：

- -d 参数：d 是 delimiter 的缩写，是英语“分隔符”的意思。用于指定用什么分隔符（比如逗号、分号、双引号等等）。
- -f 参数：f 是 field 的缩写，是英语“区域”的意思。表示剪切下用分隔符分隔的哪一块或哪几块区域。

我们的 notes.csv 文件是用逗号来分隔三个部分的，我们要剪切下来的是名字那一列，也就是第一部分。因此我们可以这样使用：

```
cut -d , -f 1 notes.csv
```

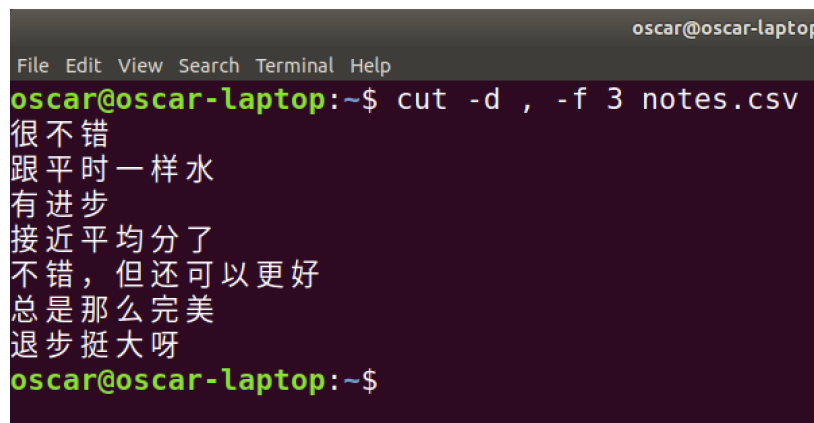


```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ cut -d , -f 1 notes.csv  
Mark  
Matthew  
Maria  
Luke  
John  
Samuel  
David  
oscar@oscar-laptop:~$
```

怎么样，很不错吧。我们通过 `cut` 命令的两个参数就实现了从 `notes.csv` 文件中剪切下第一部分（“名字”那一列）的想法。

那如果我们只想剪切下评语部分呢？评语是第三部分，可以用以下命令：

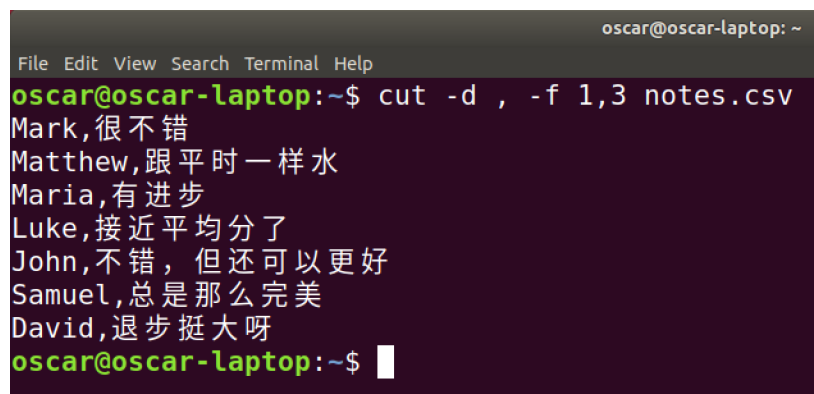
```
cut -d , -f 3 notes.csv
```



```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ cut -d , -f 3 notes.csv  
很不错  
跟平时一样水  
有进步  
接近平均分了  
不错，但还可以更好  
总是那么完美  
退步挺大呀  
oscar@oscar-laptop:~$
```

那如果我们要第一和第三部分呢？可以这样：

```
cut -d , -f 1,3 notes.csv
```



```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ cut -d , -f 1,3 notes.csv  
Mark,很不错  
Matthew,跟平时一样水  
Maria,有进步  
Luke,接近平均分了  
John,不错，但还可以更好  
Samuel,总是那么完美  
David,退步挺大呀  
oscar@oscar-laptop:~$
```

同样地，我们可以用

```
cut -d , -f 2- notes.csv
```

来剪切第二部分直到最后的内容：

```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ cut -d , -f 2- notes.csv  
95 / 100,很不错  
30 / 100,跟平时一样水  
70 / 100,有进步  
54 / 100,接近平均分  
68 / 100,不错,但还可以更好  
100 / 100,总是那么完美  
40 / 100,退步挺大呀  
oscar@oscar-laptop:~$
```

好了，现在准备工作做好了，我们可以来学习重定向流符号了。

>：重定向到新的文件

我们知道虽然刚才用 `cut` 命令从 `notes.csv` 文件中剪切出来一些部分，但原始的 `notes.csv` 文件是不变的。

我们现在想要将剪切出来的部分储存到一个文件中，而不是像之前那样显示在终端里。

为了方便演示，我们可以在家目录下新建一个 `redirect` 目录，`redirect` 是英语“重定向”的意思。将之前的 `notes.csv` 文件放到这个目录下，再用 `cd redirect` 命令定位到这个目录。

```
cd ~ # 单独用 cd 也可以定位到用户的家目录  
mkdir redirect  
mv notes.csv redirect  
cd redirect
```

我们需要用到 `>` 这个神奇的符号。如果你是美式键盘，那么可以在句号那个键找到这个符号，在句号的上面。所以要输入这个符号，可以用 `Shift` 加上句号那个键。

这个符号可以将命令的输出结果重定向到你选择的文件中。例如：

```
cut -d , -f 1 notes.csv > students.txt
```

`student` 是英语“学生”的意思，`students` 就是复数啦。

如果你运行上述命令，那么终端不会有任何显示。因为我们将 `cut` 命令的运行结果（剪切了名字那一列）重定向到 `students.txt` 文件中了。

```
oscar@oscar-laptop: ~/redirect  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~/redirect$ cut -d , -f 1 notes.csv > students.txt  
oscar@oscar-laptop:~/redirect$ cat students.txt  
Mark  
Matthew  
Maria  
Luke  
John  
Samuel  
David  
oscar@oscar-laptop:~/redirect$ ls  
notes.csv  students.txt  
oscar@oscar-laptop:~/redirect$
```

可以看到，我们用 `cat` 命令打印了 `student.txt` 文件的内容，正是 `cut -d , -f 1 notes.csv` 这句命令的运行结果。

我们用 `ls` 命令查看 `redirect` 目录下的文件，发现多了一个 `students.txt` 文件。

怎么样，> 这个符号很有用吧。

使用时要小心，因为 > 符号会把输出重定向到文件中。如果此文件不存在，则新建一个文件；如果此文件已经存在，那就会把文件内容覆盖掉（清除原有内容，然后写入文件），而且是不会征求用户确认的！

有时候，我们既不想将命令的输出显示在终端，又不想将其储存到文件中，怎么办呢？

Linux 中有一个俗称“黑洞”的文件，就是

```
/dev/null
```

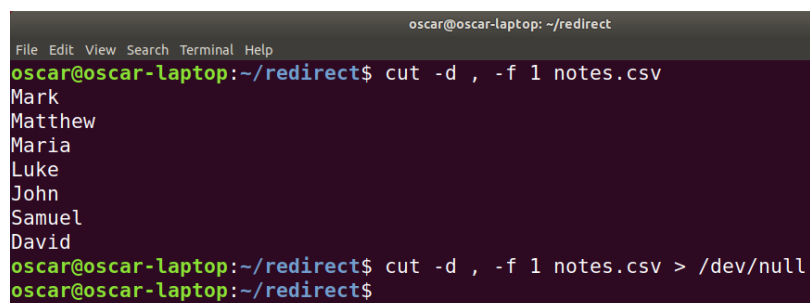
null 是英语“无，空”的意思。

/dev/null 文件是特殊文件，不是一个目录，此文件具有唯一的属性：它总是空的。它能使发送到 /dev/null 的任何数据作废，就好像这些数据掉进了无底的黑洞一般。

因此，假如我们不需要在终端显示刚才那个 cut 命令的结果，也不想存储到文件里，那么可以这么做：

```
cut -d , -f 1 notes.csv > /dev/null
```

然后，整个世界都清净了...



A terminal window titled 'oscar@oscar-laptop: ~/redirect' showing the execution of the 'cut' command. The first command is 'cut -d , -f 1 notes.csv', which outputs a list of names: Mark, Matthew, Maria, Luke, John, Samuel, and David. The second command is 'cut -d , -f 1 notes.csv > /dev/null', which produces no visible output, demonstrating redirection to the null device.

>> : 重定向到文件末尾

我们已经知道，单独一个 > 符号可以实现重定向到新的文件（覆盖文件内容），那么两个连在一起的 > 符号有什么作用呢？

>> 的作用与 > 是类似的，不过它不会像 > 那么危险（如果文件已经存在，> 符号会覆盖文件内容），而是将重定向的内容写入到文件末尾，起到追加的作用。如果文件不存在，文件也会被创建。

我们就来实践一下：

```
cut -d , -f 1 notes.csv >> students.txt
```

因为我们上一个例子中已经用 > 符号来重定向名字那列的内容到 students.txt 文件中了，所以上面的命令会追加同样内容到 students.txt 的末尾。


```
File Edit View Search Terminal Help
oscar@oscar-laptop: ~/redirect
oscar@oscar-laptop:~/redirect$ cut -d , -f 1 notes.csv >> students.txt
oscar@oscar-laptop:~/redirect$ cat students.txt
Mark
Matthew
Maria
Luke
John
Samuel
David
Mark
Matthew
Maria
Luke
John
Samuel
David
oscar@oscar-laptop:~/redirect$
```

我们用 `cat` 命令打印了 `students.txt` 文件的内容，可以看到有两遍 `notes.csv` 中名字列的内容。

`>>` 符号在很多情况下非常有用，比如你不在电脑前，而你又想让终端为你记录程序运行的结果，就可以在一个日志文件的末尾一直写入。例如：

```
command >> results.log
```

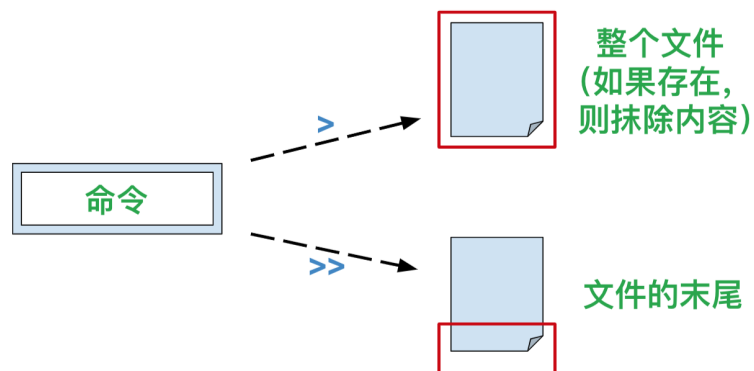
上面的 `command` 代表某个命令或一行命令。

小结

我们方才学习了两个重定向流符号：

- `>`：重定向到文件中。如果文件已存在，则覆盖文件内容；文件不存在，则创建文件。
- `>>`：重定向到文件末尾。如果文件不存在，则创建文件。

可以用下图演示两个符号的原理：



3. “2>, 2>>, 2>&1”：重定向错误输出

怎么样，流的重定向是不是很有趣？那么我们继续学习其他的重定向流符号。

我们首先来学点新知识。

stdin, stdout, stderr：标准输入，标准输出，标准错误输出

这三个又是什么呢？这一课新东西果然多。

一般学编程，到某个阶段，总应该会碰到这三位“仁兄”的。既然逃不了，不如把它学好。

对于我们的终端命令行，我们从键盘向终端输入数据，这是标准输入，也就是 `stdin`。

终端接收键盘输入的命令，会产生两种输出：

- 标准输出：`stdout`。指终端输出的信息（不包括错误信息）。
- 标准错误输出：`stderr`。指终端输出的错误信息。

第一个 `stdout` 就是我们到目前为止看到的那些 Linux 命令的正常运行结果，比如在终端中运行 `ls` 命令，我们以前也看到了它列出当前目录下所有文件。

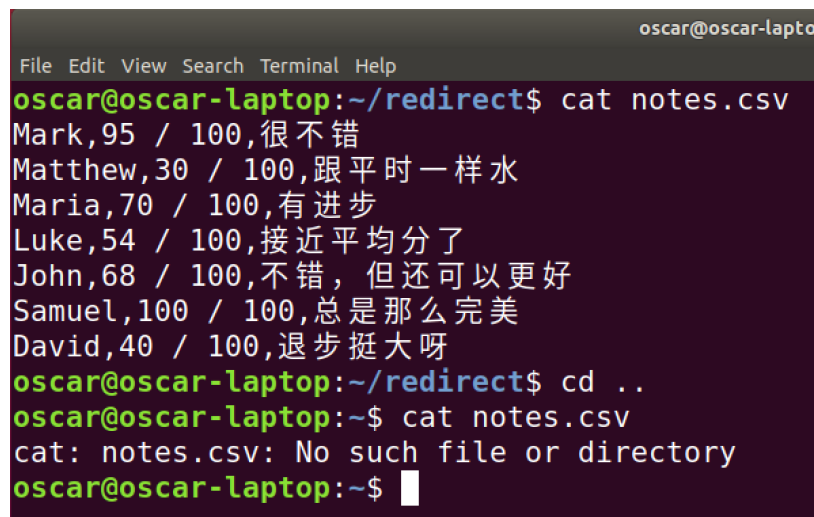
那什么是标准错误输出呢？其实我们以前也看到过，只是见得不多而已。

我们用一个例子来说明。

假设，我们运行 `cat notes.csv` 命令，想要显示 `notes.csv` 文件的内容。将会有两种结果：

如果一切顺利（`notes.csv` 文件存在于当前目录，而且我们有权限这么做），那么终端就会显示其内容。这是标准输出。

如果出错（比如 `notes.csv` 文件不存在），那么终端就会显示错误信息。这是标准错误输出。

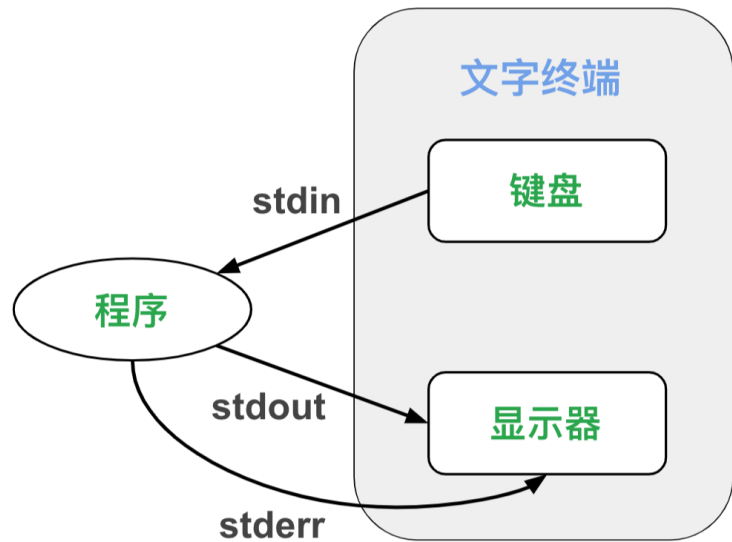
A terminal window titled 'oscar@oscar-laptop' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt is 'oscar@oscar-laptop:~/redirect\$'. The first command is 'cat notes.csv', which outputs a list of names and scores: 'Mark,95 / 100,很不错', 'Matthew,30 / 100,跟平时一样水', 'Maria,70 / 100,有进步', 'Luke,54 / 100,接近平均分了', 'John,68 / 100,不错, 但还可以更好', 'Samuel,100 / 100,总是那么完美', 'David,40 / 100,退步挺大呀'. The second command is 'cd ..', changing the directory to '~'. The third command is 'cat notes.csv', which outputs an error message: 'cat: notes.csv: No such file or directory'. The prompt returns to 'oscar@oscar-laptop:~\$'.

如上图所示，我们在 `redirect` 这个目录中运行 `cat notes.csv` 时，因为文件存在，而且我们有权限对其执行 `cat` 命令，所以打印出了文件内容。这就是标准输出。

接着，我们用 `cd ..` 命令定位到了上一级目录，也就是用户的家目录，在这个目录中并没有 `notes.csv` 文件，所以 `cat notes.csv` 命令自然就输出了错误信息：`No such file or directory`，意思是“不存在此文件或目录”。这就是标准错误输出。

默认情况下，标准输出和标准错误输出都会显示在终端，这也是为什么我们之前对它们的区别并没有那么在意的原因，因为长得挺像的，都在终端输出。

我们可以用下图来演示：



这三个你也可以把它们看作流：

- **stdin**：标准输入流。英语 standard input 的缩写（standart 是英语“标准”的意思，input 是英语“输入”的意思）。标准输入是指输入至程序的数据（通常是文件），程序要求以读（read）操作来传输数据。并非所有程序都要求输入，如 dir 或 ls 程序运行时不用任何输入。除非重定向，输入是预期由键盘获取的，标准输入的文件描述符为 0（零）。
- **stdout**：标准输出流。英语 standard output 的缩写（output 是英语“输出”的意思）。标准输出是指程序输出的数据，程序要求数据传输使用写的运算。并非所有程序都要求输出，如 mv 或 ren 程序在成功完成时是没有输出的。除非重定向，输出是预期显示在终端上的。标准输出的文件描述符为 1（一）。
- **stderr**：标准错误输出流。英语 standard error 的缩写（error 是英语“错误”的意思）。标准错误输出是另一个输出流，用于输出错误消息或诊断。它独立于标准输出，且标准输出和标准错误输出可以分别被重定向。标准错误输出的文件描述符为 2（二）。

文件描述符	名字	解释
0	stdin	标准输入
1	stdout	标准输出
2	stderr	标准错误输出

那什么是文件描述符呢？

文件描述符的英语是 File Descriptor，简称 fd。File 是英语“文件”的意思，而 Descriptor 是英语“描述符”的意思。

文件描述符是计算机科学中的一个术语，要完全讲清楚可能要用单独的一课，我们就不深究了。

文件描述符是一个用于表述指向文件的引用的抽象化概念。这定义本身也有点抽象，我们不需要太深入了解，大致只需要知道：

文件描述符在形式上是一个非负整数。

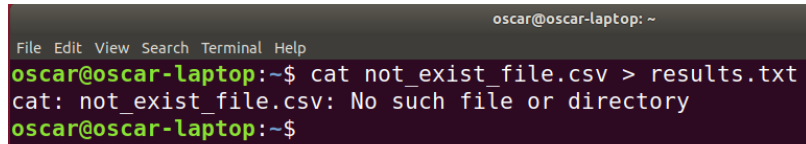
实际上，它是一个索引值，指向操作系统内核为每一个进程所维护的该进程打开文件的记录表。

当程序打开一个现有文件或者创建一个新文件时，内核向进程返回一个文件描述符。

文件描述符通常是 Unix, Linux 等系统的概念。在 Windows 中, 也有类似的概念, 但是 Windows 中称为“句柄”, 就是 handle。

好了, 重新回到我们的话题。刚才我们已经学习了用 `>` 和 `>>` 两个符号将输出重定向到文件中, 那么我们在出错的情况下是不是也可以用 `>` 和 `>>` 将标准错误输出也重定向到文件中呢? 我们来试试:

```
cat not_exist_file.csv > results.txt
```



```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ cat not_exist_file.csv > results.txt  
cat: not_exist_file.csv: No such file or directory  
oscar@oscar-laptop:~$
```

如上图所示, 因为 `not_exist_file.csv` 这个文件不存在 (正如其名。not 是“不”的意思, exist 是“存在”的意思, file 是“文件”的意思, 请别在意此处的英语语法问题), 所以产生了错误信息。但是这个错误信息却并没有如我们所愿的写入到 `results.txt` 文件中, 而是仍旧在终端输出了。这是为什么呢? 不是说好的 `>` 符号用于重定向输出到文件的吗?

其实, `>` 和 `>>` 符号只是将标准输出重定向到文件, 并不能将标准错误输出重定向到文件。

那么我们要重定向标准错误输出, 该怎么办呢?

我们就要用到 `2>` 这个符号。是的, 就是在 `>` 这个符号左边紧挨着写一个 2。

为什么是 2 呢? 记得上面说的吗? 标准错误输出的文件描述符是 2, 所以这里的 2 表示标准错误输出。如果没有 2, 单独的 `>` 符号就是重定向标准输出 (文件描述符为 1)。

我们补充一下刚才的命令:

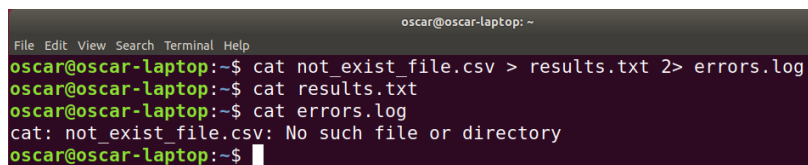
```
cat not_exist_file.csv > results.txt 2> errors.log
```

这个命令里有两个重定向:

- `> results.txt`: 将标准输出重定向到 `results.txt` 文件中;
- `2> errors.log`: 将标准错误输出重定向到 `errors.log` 文件中。

也就是说:

- 假如 `not_exist_file.csv` 这个文件确实存在, 将其内容写入 `results.txt` 文件中。
- 假如 `not_exist_file.csv` 这个文件不存在, 将错误信息写入 `errors.log` 文件中。



```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ cat not_exist_file.csv > results.txt 2> errors.log  
oscar@oscar-laptop:~$ cat results.txt  
oscar@oscar-laptop:~$ cat errors.log  
cat: not_exist_file.csv: No such file or directory  
oscar@oscar-laptop:~$
```

如上图所示, 因为 `not_exist_file.csv` 这个文件不存在, 所以 `results.txt` 文件是空的, 而 `errors.log` 文件的内容是错误信息 (`cat: not_exist_file.csv: No such file or directory`)。

类似的, `2>>` 符号用于将标准错误输出重定向到文件末尾。

合并输出

上面我们学习了如何将标准输出和标准错误输出分别重定向到不同文件。但是有的时候，我们比较“任性”，就想把标准输出和标准错误输出都重定向到同一个地方。怎么做呢？

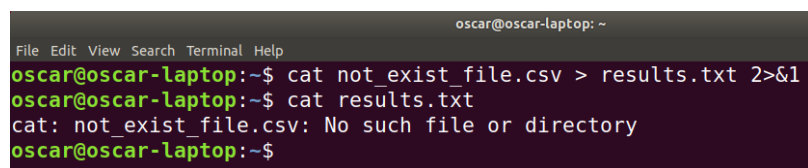
须要使用 `2>&1` 这个组合符号。

看着怪怪的对吧？仅由四个字符组成。这个符号的作用是：将标准错误输出重定向到与标准输出相同的地方。

我们用实例演示一下：

```
cat not_exist_file.csv > results.txt 2>&1
```

上面的命令的作用是：将 `cat not_exist_file.csv` 这个命令的所有输出（标准输出和标准错误输出）都重定向到 `results.txt` 文件中。



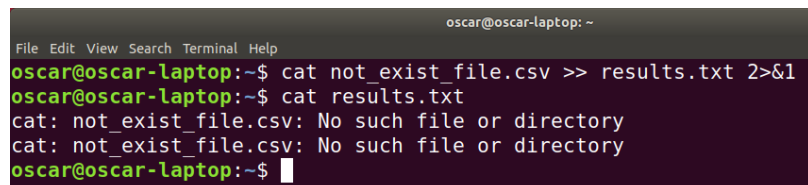
```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ cat not_exist_file.csv > results.txt 2>&1  
oscar@oscar-laptop:~$ cat results.txt  
cat: not_exist_file.csv: No such file or directory  
oscar@oscar-laptop:~$
```

如上图所示，因为 `not_exist_file.csv` 这个文件不存在，但因为加了 `2>&1` 这个符号，所以标准输出（为空）和标准错误输出（`cat: not_exist_file.csv: No such file or directory`）都重定向到 `results.txt` 文件中了。

大家是否觉得要将标准输出和标准错误输出都重定向到文件末尾，应该是这样写：`2>>&1` 呢？

其实不然，这样是不对的。我们还是保持 `2>&1` 这个组合不变，只改变前面的符号就行了。例如：

```
cat not_exist_file.csv >> results.txt 2>&1
```



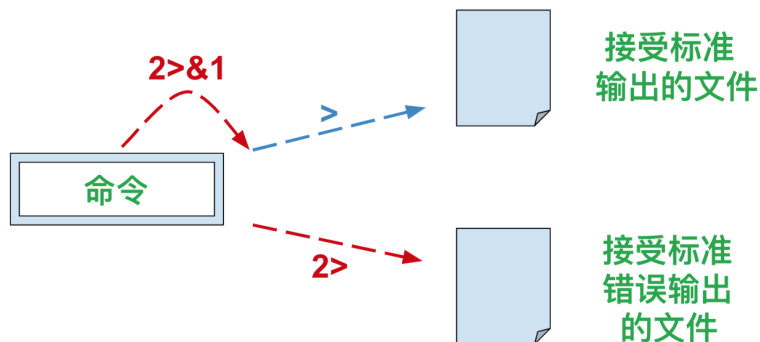
```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ cat not_exist_file.csv >> results.txt 2>&1  
oscar@oscar-laptop:~$ cat results.txt  
cat: not_exist_file.csv: No such file or directory  
cat: not_exist_file.csv: No such file or directory  
oscar@oscar-laptop:~$
```

可以看到，追加了一条错误信息到 `results.txt` 文件中。

小结

- `2>`：将标准错误输出重定向到文件。如果文件已经存在，则覆盖文件内容；如果不存在，则创建文件。
- `2>>`：将标准错误输出重定向到文件末尾。如果文件不存在，则创建文件。
- `2>&1`：将标准输出和标准错误输出重定向到同一个地方。

用下图来演示：



上图中，我故意没有加 >> 和 2>>，不然这图就太复杂了。

其实 > 和 >>，2> 和 2>> 的区别就是前者覆盖文件内容，后者追加内容到文件。

小结

1. Linux 命令的结果（标准输出），我们不一定要显示在终端里，也可以将其存放在一个文件里。只需要在命令后加上 > 符号，然后接文件名就可以了。例如 `ls > file_list.txt` 就会把 ls 命令的结果（当前目录所有文件的列表）存放到 file_list.txt 文件中，不在终端显示了。
2. >> 符号可以追加内容。> 符号会把文件内容清空，再写入。如果文件已经存在，那么 >> 符号会在文件末尾追加写入内容。
3. 2> 和 2>> 符号用于重定向标准错误输出到文件中。2>&1 符号用于将标准错误输出和标准输出重定向到相同地方。

今天的课就到这里，一起加油吧！