##### ****3.1 输入输出重定向****

既然我们已经在上一章学完了几乎所有基础且常用的Linux命令，那么接下来的任务就是把多个Linux命令适当地组合到一起，使其协同工作，以便我们更加高效地处理数据。要做到这一点，就必须搞明白命令的输入重定向和输出重定向的原理。

简而言之，输入重定向是指把文件导入到命令中，而输出重定向则是指把原本要输出到屏幕的数据信息写入到指定文件中。在日常的学习和工作中，相较于输入重定向，我们使用输出重定向的频率更高，所以又将输出重定向分为了标准输出重定向和错误输出重定向两种不同的技术，以及清空写入与追加写入两种模式。听起来就很玄妙？[刘遄](https://www.linuxprobe.com/" \o "刘遄" \t "https://www.linuxprobe.com/_blank)老师接下来将慢慢道来。

标准输入重定向（STDIN，文件描述符为0）：默认从键盘输入，也可从其他文件或命令中输入。

标准输出重定向（STDOUT，文件描述符为1）：默认输出到屏幕。

错误输出重定向（STDERR，文件描述符为2）：默认输出到屏幕。

比如我们分别查看两个文件的属性信息，其中第二个文件是不存在的，虽然针对这两个文件的操作都分别会在屏幕上输出一些数据信息，但这两个操作的差异其实很大：

[root@linuxprobe ~]# touch linuxprobe

[root@linuxprobe ~]# ls -l linuxprobe

-rw-r--r--. 1 root root 0 Aug 5 05:35 linuxprobe

[root@linuxprobe ~]# ls -l xxxxxx

ls: cannot access xxxxxx: No such file or directory

在上述命令中，名为linuxprobe的文件是存在的，输出信息是该文件的一些相关权限、所有者、所属组、文件大小及修改时间等信息，这也是该命令的标准输出信息。而名为xxxxxx的第二个文件是不存在的，因此在执行完ls命令之后显示的报错提示信息也是该命令的错误输出信息。那么，要想把原本输出到屏幕上的数据转而写入到文件当中，就要区别对待这两种输出信息。

对于输入重定向来讲，用到的符号及其作用如表3-1所示。

表3-1                                         输入重定向中用到的符号及其作用

|  |  |
| --- | --- |
| 符号 | 作用 |
| 命令 < 文件 | 将文件作为命令的标准输入 |
| 命令 << 分界符 | 从标准输入中读入，直到遇见分界符才停止 |
| 命令 < 文件1 > 文件2 | 将文件1作为命令的标准输入并将标准输出到文件2 |

对于输出重定向来讲，用到的符号及其作用如表3-2所示。

表3-2                                         输出重定向中用到的符号及其作用

|  |  |
| --- | --- |
| 符号 | 作用 |
| 命令 > 文件 | 将标准输出重定向到一个文件中（清空原有文件的数据） |
| 命令 2> 文件 | 将错误输出重定向到一个文件中（清空原有文件的数据） |
| 命令 >> 文件 | 将标准输出重定向到一个文件中（追加到原有内容的后面） |
| 命令 2>> 文件 | 将错误输出重定向到一个文件中（追加到原有内容的后面） |
| 命令 >> 文件 2>&1  或 命令 &>> 文件 | 将标准输出与错误输出共同写入到文件中（追加到原有内容的后面） |

对于重定向中的标准输出模式，可以省略文件描述符1不写，而错误输出模式的文件描述符2是必须要写的。我们先来小试牛刀。通过标准输出重定向将man bash命令原本要输出到屏幕的信息写入到文件readme.txt中，然后显示readme.txt文件中的内容。具体命令如下：

[root@linuxprobe ~]# man bash > readme.txt

[root@linuxprobe ~]# cat readme.txt

BASH(1)                     General Commands Manual                    BASH(1)

NAME

bash - GNU Bourne-Again [Shell](https://www.linuxcool.com/" \o "shell" \t "https://www.linuxprobe.com/_blank)

SYNOPSIS

bash [options] [file]

COPYRIGHT

Bash is Copyright (C) 1989-2011 by the Free Software Foundation, Inc.

DESCRIPTION

Bash  is  an  sh-compatible  command language interpreter that executes

commands read from the standard input or from a file.  Bash also incor‐

porates useful features from the Korn and C shells (ksh and csh).

Bash  is  intended  to  be a conformant implementation of the Shell and

Utilities portion  of  the  IEEE  POSIX  specification  (IEEE  Standard

1003.1).  Bash can be configured to be POSIX-conformant by default.

………………省略部分输出信息………………

有没有感觉到很方便呢？我们接下来尝试输出重定向技术中的覆盖写入与追加写入这两种不同模式带来的变化。首先通过覆盖写入模式向readme.txt文件写入一行数据（该文件中包含上一个实验的man命令信息），然后再通过追加写入模式向文件再写入一次数据，其命令如下：

[root@linuxprobe ~]# echo "Welcome to LinuxProbe.Com" > readme.txt

[root@linuxprobe ~]# echo "Quality linux learning materials" >> readme.txt

在执行cat命令之后，可以看到如下所示的文件内容：

[root@linuxprobe ~]# cat readme.txt

Welcome to LinuxProbe.Com

Quality linux learning materials

虽然都是输出重定向技术，但是不同命令的标准输出和错误输出还是有区别的。例如查看当前目录中某个文件的信息，这里以linuxprobe文件为例。因为这个文件是真实存在的，因此使用标准输出即可将原本要输出到屏幕的信息写入到文件中，而错误的输出重定向则依然把信息输出到了屏幕上。

[root@linuxprobe ~]# ls -l linuxprobe

-rw-r--r--. 1 root root 0 Mar  1 13:30 linuxprobe

[root@linuxprobe ~]# ls -l linuxprobe > /root/stderr.txt

[root@linuxprobe ~]# ls -l linuxprobe 2> /root/stderr.txt

-rw-r--r--. 1 root root 0 Mar  1 13:30 linuxprobe

如果想把命令的报错信息写入到文件，该怎么操作呢？当用户在执行一个自动化的Shell脚本时，这个操作会特别有用，而且特别实用，因为它可以把整个脚本执行过程中的报错信息都记录到文件中，便于安装后的排错工作。接下来我们以一个不存在的文件进行实验演示：

[root@linuxprobe ~]# ls -l xxxxxx

cannot access xxxxxx: No such file or directory

[root@linuxprobe ~]# ls -l xxxxxx > /root/stderr.txt

cannot access xxxxxx: No such file or directory

[root@linuxprobe ~]# ls -l xxxxxx 2> /root/stderr.txt

[root@linuxprobe ~]# cat /root/stderr.txt

ls: cannot access xxxxxx: No such file or directory

输入重定向相对来说有些冷门，在工作中遇到的概率会小一点。输入重定向的作用是把文件直接导入到命令中。接下来使用输入重定向把readme.txt文件导入给wc -l命令，统计一下文件中的内容行数。

[root@linuxprobe ~]# wc -l < readme.txt

2

上述命令实际上等同于接下来要学习的cat readme.txt | wc -l的管道符命令组合。

##### ****3.2 管道命令符****

细心的读者肯定还记得在2.6节学习tr命令时曾经见到过一个名为管道符的东西。同时按下键盘上的Shift+\键即可输入管道符，其执行格式为“命令A | 命令B”。管道命令符的作用也可以用一句话来概括“把前一个命令原本要输出到屏幕的标准正常数据当作是后一个命令的标准输入”。在2.8节讲解grep文本搜索命令时，我们通过匹配关键词/sbin/nologin找出了所有被限制登录系统的用户。在学完本节内容后，完全可以把下面这两条命令合并为一条：

找出被限制登录用户的命令是grep "/sbin/nologin" /etc/passwd；

统计文本行数的命令则是wc -l。

现在要做的就是把搜索命令的输出值传递给统计命令，即把原本要输出到屏幕的用户信息列表再交给wc命令作进一步的加工，因此只需要把管道符放到两条命令之间即可，具体如下。这简直是太方便了！

[root@linuxprobe ~]# grep "/sbin/nologin" /etc/passwd | wc -l

33

这个管道符就像一个法宝，我们可以将它套用到其他不同的命令上，比如用翻页的形式查看/etc目录中的文件列表及属性信息（这些内容默认会一股脑儿地显示到屏幕上，根本看不清楚）：

[root@linuxprobe ~]# ls -l /etc/ | more

total 1400

drwxr-xr-x. 3 root root 97 Jul 10 17:26 abrt

-rw-r--r--. 1 root root 16 Jul 10 17:36 adjtime

-rw-r--r--. 1 root root 1518 Jun 7 2013 aliases

-rw-r--r--. 1 root root 12288 Jul 10 09:38 aliases.db

drwxr-xr-x. 4 root root 94 Jul 10 17:26 avahi--More--

在修改用户密码时，通常都需要输入两次密码以进行确认，这在编写自动化脚本时将成为一个非常致命的缺陷。通过把管道符和passwd命令的--stdin参数相结合，我们可以用一条命令来完成密码重置操作：

[root@linuxprobe ~]# echo "linuxprobe" | passwd --stdin root

Changing password for user root.

passwd: all authentication tokens updated successfully.

大家是不是觉得管道符命令有些相见恨晚？管道符的玩法还有很多，比如，在发送电子邮件时，默认采用交互式的方式来进行，我们完全可以利用一条结合了管道符的命令语句，把编辑好的内容与标题一起“打包”，最终用这一条命令实现邮件的发送。

[root@linuxprobe ~]# echo "Content" | mail -s "Subject" linuxprobe

[root@linuxprobe ~]# su - linuxprobe

Last login: Fri Jul 10 09:44:07 CST 2017 on :0

[linuxprobe@linuxprobe ~]$ mail

Heirloom Mail version 12.5 7/5/10. Type ? for help.

"/var/spool/mail/linuxprobe": 1 message 1 new

>N 1 root Sun Aug 30 17:33 18/578 "Subject"

下面这条自造的命令就结合使用了mail邮件命令与输入重定向的分界符，其目的是让用户一直输入内容，直到用户输入了其自定义的分界符时，才结束输入。

[root@linuxprobe ~]# mail -s "Readme" root@linuxprobe.com << over****>**** I think linux is very practical****>**** I hope to learn more****>**** can you teach me ?****>**** over

[root@linuxprobe ~]#

当然，大家千万不要误以为管道命令符只能在一个命令组合中使用一次，我们完全可以这样使用：“命令A | 命令B | 命令C”。

##### ****3.3 命令行的通配符****

大家可能都遇到过提笔忘字的尴尬，作为Linux运维人员，我们有时候也会遇到明明一个文件的名称就在嘴边但就是想不起来的情况。如果就记得一个文件的开头几个字母，想遍历查找出所有以这个关键词开头的文件，该怎么操作呢？又比如，假设想要批量查看所有硬盘文件的相关权限属性，一种方式是这样的：

[root@linuxprobe ~]# ls -l /dev/sda

brw-rw----. 1 root disk 8, 0 May 4 15:55 /dev/sda

[root@linuxprobe ~]# ls -l /dev/sda1

brw-rw----. 1 root disk 8, 1 May 4 15:55 /dev/sda1

[root@linuxprobe ~]# ls -l /dev/sda2

brw-rw----. 1 root disk 8, 2 May 4 15:55 /dev/sda2

[root@linuxprobe ~]# ls -l /dev/sda3

ls: cannot access /dev/sda3: No such file or directory

幸亏我的硬盘文件和分区只有3个，要是有几百个，估计需要花费一天的时间来忙这个事情了。由此可见，这种方式的效率确实很低。虽然我们在第6章才会讲解Linux系统的存储结构和FHS，但现在我们应该能看出一些简单规律了。比如，这些硬盘设备文件都是以sda开头并且存放到了/dev目录中，这样一来，即使我们不知道硬盘的分区编号和具体分区的个数，也可以使用通配符来搞定。顾名思义，通配符就是通用的匹配信息的符号，比如星号（\*）代表匹配零个或多个字符，问号（?）代表匹配单个字符，中括号内加上数字[0-9]代表匹配0～9之间的单个数字的字符，而中括号内加上字母[abc]则是代表匹配a、b、c三个字符中的任意一个字符。俗话讲“百闻不如一见，看书不如实验”，下面我们就来匹配所有在/dev目录中且以sda开头的文件：

[root@linuxprobe ~]# ls -l /dev/sda\*

brw-rw----. 1 root disk 8, 0 May 4 15:55 /dev/sda

brw-rw----. 1 root disk 8, 1 May 4 15:55 /dev/sda1

brw-rw----. 1 root disk 8, 2 May 4 15:55 /dev/sda2

如果只想查看文件名为sda开头，但是后面还紧跟其他某一个字符的文件的相关信息，该怎么操作呢？这时就需要用到问号来进行通配了。

[root@linuxprobe ~]# ls -l /dev/sda?

brw-rw----. 1 root disk 8, 1 May 4 15:55 /dev/sda1

brw-rw----. 1 root disk 8, 2 May 4 15:55 /dev/sda2

除了使用[0-9]来匹配0~9之间的单个数字，也可以用[135]这样的方式仅匹配这三个指定数字中的一个，若没有匹配到，则不会显示出来：

[root@linuxprobe ~]# ls -l /dev/sda[0-9]

brw-rw----. 1 root disk 8, 1 May 4 15:55 /dev/sda1

brw-rw----. 1 root disk 8, 2 May 4 15:55 /dev/sda2

[root@linuxprobe ~]# ls -l /dev/sda[135]

brw-rw----. 1 root disk 8, 1 May 4 15:55 /dev/sda1

##### ****3.4 常用的转义字符****

为了能够更好地理解用户的表达，Shell解释器还提供了特别丰富的转义字符来处理输入的特殊数据。刘遄老师以近十年的工作和培训为基础，愣是用了两周时间从数十个转义字符中提炼出了4个最常用的转义字符！这件事情也让我深刻反省了很长时间。原本认为图书写的越厚，作者越是大牛，现在发现这种观念完全是错误的，希望读者在读完本书后能体会到刘遄老师的用心付出。

4个最常用的转义字符如下所示。

反斜杠（\）：使反斜杠后面的一个变量变为单纯的字符串。

单引号（''）：转义其中所有的变量为单纯的字符串。

双引号（""）：保留其中的变量属性，不进行转义处理。

反引号（``）：把其中的命令执行后返回结果。

我们先定义一个名为PRICE的变量并赋值为5，然后输出以双引号括起来的字符串与变量信息：

[root@linuxprobe ~]# PRICE=5

[root@linuxprobe ~]# echo "Price is $PRICE"

Price is 5

接下来，我们希望能够输出“Price is $5”，即价格是5美元的字符串内容，但碰巧美元符号与变量提取符号合并后的$$作用是显示当前程序的进程ID号码，于是命令执行后输出的内容并不是我们所预期的：

[root@linuxprobe ~]# echo "Price is $$PRICE"

Price is 3767PRICE

要想让第一个“$”乖乖地作为美元符号，那么就需要使用反斜杠（\）来进行转义，将这个命令提取符转义成单纯的文本，去除其特殊功能。

[root@linuxprobe ~]# echo "Price is \$$PRICE"

Price is $5

而如果只需要某个命令的输出值时，可以像`命令`这样，将命令用反引号括起来，达到预期的效果。例如，将反引号与uname -a命令结合，然后使用echo命令来查看本机的Linux版本和内核信息：

[root@linuxprobe ~]# echo `uname -a`

Linux linuxprobe.com 3.10.0-123.el7.x86\_64 #1 SMP Mon May 5 11:16:57 EDT 2017 x86\_64 x86\_64 x86\_64 GNU/Linux

##### ****3.5 重要的环境变量****

变量是计算机系统用于保存可变值的数据类型。在Linux系统中，变量名称一般都是大写的，这是一种约定俗成的规范。我们可以直接通过变量名称来提取到对应的变量值。Linux系统中的环境变量是用来定义系统运行环境的一些参数，比如每个用户不同的家目录、邮件存放位置等。

细心的读者应该发现了，本节和上一节的标题名都分别加了形容词—重要的、常见的。原因其实不言而喻—要想让Linux系统能够正常运行并且为用户提供服务，需要数百个环境变量来协同工作，我们没有必要逐一查看、学习每一个变量，而是应该在有限的篇幅中精讲最重要的内容。

为了通过环境变量帮助Linux系统构建起能够为用户提供服务的工作运行环境，需要数百个变量协同工作才能完成。您当然没有必要去把每一个变量都看一遍，而应该在最宝贵的书籍中为读者精讲最重要的内容。为了更好地帮助大家理解变量的作用，刘遄老师给大家举个例子。前文中曾经讲到，在Linux系统中一切都是文件，Linux命令也不例外。那么，在用户执行了一条命令之后，Linux系统中到底发生了什么事情呢？简单来说，命令在Linux中的执行分为4个步骤。

****第1步****：判断用户是否以绝对路径或相对路径的方式输入命令（如/bin/ls），如果是的话则直接执行。

****第2步****：Linux系统检查用户输入的命令是否为“别名命令”，即用一个自定义的命令名称来替换原本的命令名称。可以用alias命令来创建一个属于自己的命令别名，格式为“alias 别名=命令”。若要取消一个命令别名，则是用unalias命令，格式为“unalias 别名”。我们之前在使用rm命令删除文件时，Linux系统都会要求我们再确认是否执行删除操作，其实这就是Linux系统为了防止用户误删除文件而特意设置的rm别名命令，接下来我们把它取消掉：

[root@linuxprobe ~]# ls

anaconda-ks.cfg Documents initial-setup-ks.cfg Pictures Templates

Desktop Downloads Music Public Videos

[root@linuxprobe ~]# rm anaconda-ks.cfg

rm: remove regular file ‘anaconda-ks.cfg’? y

[root@linuxprobe~]# alias rm

alias rm='rm -i'

[root@linuxprobe ~]# unalias rm

[root@linuxprobe ~]# rm initial-setup-ks.cfg

[root@linuxprobe ~]#

****第3步****：Bash解释器判断用户输入的是内部命令还是外部命令。内部命令是解释器内部的指令，会被直接执行；而用户在绝大部分时间输入的是外部命令，这些命令交由步骤4继续处理。可以使用“type命令名称”来判断用户输入的命令是内部命令还是外部命令。

****第4步****：系统在多个路径中查找用户输入的命令文件，而定义这些路径的变量叫作PATH，可以简单地把它理解成是“解释器的小助手”，作用是告诉Bash解释器待执行的命令可能存放的位置，然后Bash解释器就会乖乖地在这些位置中逐个查找。PATH是由多个路径值组成的变量，每个路径值之间用冒号间隔，对这些路径的增加和删除操作将影响到Bash解释器对Linux命令的查找。

[root@linuxprobe ~]# echo $PATH

/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin

[root@linuxprobe ~]# PATH=$PATH:****/root/bin****[root@linuxprobe ~]# echo $PATH

/usr/local/bin:/usr/local/sbin:/usr/bin:/usr/sbin:/bin:/sbin:****/root/bin****

这里有比较经典的问题：“为什么不能将当前目录（.）添加到PATH中呢? ” 原因是，尽管可以将当前目录（.）添加到PATH变量中，从而在某些情况下可以让用户免去输入命令所在路径的麻烦。但是，如果黑客在比较常用的公共目录/tmp中存放了一个与ls或cd命令同名的木马文件，而用户又恰巧在公共目录中执行了这些命令，那么就极有可能中招了。

所以，作为一名态度谨慎、有经验的运维人员，在接手了一台Linux系统后一定会在执行命令前先检查PATH变量中是否有可疑的目录，另外读者从前面的PATH变量示例中是否也感觉到环境变量特别有用呢。我们可以使用env命令来查看到Linux系统中所有的环境变量，而刘遄老师为您精挑细选出了最重要的10个环境变量，如表3-3所示。

表3-3                                       Linux系统中最重要的10个环境变量

|  |  |
| --- | --- |
| 变量名称 | 作用 |
| HOME | 用户的主目录（即家目录） |
| SHELL | 用户在使用的Shell解释器名称 |
| HISTSIZE | 输出的历史命令记录条数 |
| HISTFILESIZE | 保存的历史命令记录条数 |
| MAIL | 邮件保存路径 |
| LANG | 系统语言、语系名称 |
| RANDOM | 生成一个随机数字 |
| PS1 | Bash解释器的提示符 |
| PATH | 定义解释器搜索用户执行命令的路径 |
| EDITOR | 用户默认的文本编辑器 |

Linux作为一个多用户多任务的操作系统，能够为每个用户提供独立的、合适的工作运行环境，因此，一个相同的变量会因为用户身份的不同而具有不同的值。例如，我们使用下述命令来查看HOME变量在不同用户身份下都有哪些值（su是用于切换用户身份的命令，将在第5章跟大家见面）：

[root@linuxprobe ~]# echo $HOME

/root

[root@linuxprobe ~]# su - linuxprobe

Last login: Fri Feb 27 19:49:57 CST 2017 on pts/0

[linuxprobe@linuxprobe ~]$ echo $HOME

/home/linuxprobe

其实变量是由固定的变量名与用户或系统设置的变量值两部分组成的，我们完全可以自行创建变量，来满足工作需求。例如设置一个名称为WORKDIR的变量，方便用户更轻松地进入一个层次较深的目录：

[root@linuxprobe ~]# mkdir /home/workdir

[root@linuxprobe ~]# WORKDIR=/home/workdir

[root@linuxprobe ~]# cd $WORKDIR

[root@linuxprobe workdir]# pwd

/home/workdir

但是，这样的变量不具有全局性，作用范围也有限，默认情况下不能被其他用户使用。如果工作需要，可以使用export命令将其提升为全局变量，这样其他用户也就可以使用它了：

[root@linuxprobe workdir]# su linuxprobe

Last login: Fri Mar 20 20:52:10 CST 2017 on pts/0

[linuxprobe@linuxprobe ~]$ cd $WORKDIR

[linuxprobe@linuxprobe ~]$ echo $WORKDIR

[linuxprobe@linuxprobe ~]$ exit

[root@linuxprobe ~]# ****export WORKDIR****[root@linuxprobe ~]# su linuxprobe

Last login: Fri Mar 20 21:52:10 CST 2017 on pts/0

[linuxprobe@linuxprobe ~]$ cd $WORKDIR

[linuxprobe@linuxprobe workdir]$ pwd

/home/workdir