从零开始：Linux 基础教程之命令的使用

对于 Linux 新手，或者那些想要重新审视或改进自己对基本 Linux 概念（比如：复制和移动文件、创建符号和硬链接、设置文件系统对象所有权和权限以及同管道和重定向一起使用 Linux 的标准文本处理命令）的理解的人来说，本系列连载文章之一是理想的教材。沿着这个方向，我们将一起分享很多心得、技巧和窍门，使该教程甚至对于那些具有丰 富经验的 Linux 老手来说都是“有血有肉”，并且是实用的。

对于初学者而言，本系列文章的许多内容都很新颖，而更有经验的 Linux 用户可能会发现本教程是使他们基本的 Linux 技能“炉火纯青”的有效途径。

介绍 bash

shell

如果您使用过 Linux 系统，那么您知道当登录时，将会看到像这样的提示符：

```
$
```

您所看到的特殊的提示符可能看起来很不一样。它可能包含系统的主机名、当前的工作目录名，或者两者都有。但是不管这个特殊的提示符看起来像什 么，有一件事是肯定的。打印出这个提示符的程序叫“shell”，极有可能您的特殊的 shell 是一个叫 bash 的程序。

您在运行 bash 吗?

您可以通过输入下面的命令来检查您是否正在运行 bash：

```
$ echo $SHELL

/bin/bash
```

如果上面的命令行报错或者不会类似地响应我们的示例，那么您可能正在运行一个不同于 bash 的 shell。

关于 bash

Bash 是“Bourne-again shell”的首字母缩写，它是大多数 Linux 系统缺省的 shell。shell 的任务是执行您的命令，使您能够与 Linux 系统进行交互。当您输完命令，您可以通知 shell 执行 exit 或 logout 命令，在此您将返回到登录提示符。顺便提一下，您还可以通过在 bash 提示符下按 control-D 来注销。

使用“cd”

您可能已经发现，目不转睛地盯着 bash 提示符可不是世界上最让人感到有劲的事。那么，让我们来开始用 bash 来浏览我们的文件系统。在提示符下，输入下面的命令（不包括 $）：

```
$ cd /
```

我们只告诉 bash 您想在 /（也称为根目录）中工作；系统上的所有目录形成一棵树，/ 被认为是这棵树的顶部，或者是根。cd 设置当前您正在工作的目录，也称为“当前工作目录

”。

路径

要看 bash 的当前工作目录，您可以输入：

```
$ pwd

/
```

在上面的示例中，cd 的 / 参数叫做路径。它告诉 cd 我们要转到什么地方。特别是，/ 参数是一个绝对路径，意味着它指定了相对于文件系统树的根的位置。绝对路径这里有几个其它的绝对路径：

```
/dev
/usr
/usr/bin
/usr/local/bin
```

您可以看到，所有绝对路径有一个共同点就是，它们都以/开头。通过路径 /usr/local/bin，我们告诉 cd 进入 / 目录，接着进入这个目录之下的 usr 目录，然后再进入 local 和 bin。绝对路径总是通过是否以 / 开头来判断。

相对路径

另一种路径叫相对路径。在 Bash 中，cd 以及其它命令总是解释那些相对于当前目录的路径。相对路径绝不会以 / 开头。这样，如果我们在 /usr 中：

```
$ cd /usr
```

那么，我们可以使用相对路径来转到 /usr/local/bin 目录：

```
$ cd local/bin
$ pwd
/usr/local/bin
```

使用 “..”

相对路径还可以包含一个或多个 .. 目录。.. 目录是指向父目录的专门目录。那么，继续前面的示例：

```
$ pwd
/usr/local/bin
$ cd ..
$ pwd
/usr/local
```

您可以看到，现在我们的当前目录是 /usr/local。我们能够“后退”到相对于我们所在的当前目录的一个目录。此外，我们还可以将 \\\\\"..\\\\\\" 添加到一个现有的相对路径中，使我们可以进入与我们已在目录并排的目录，例如：

```
$ pwd
/usr/local
$ cd ../share
```

```
$ pwd
/usr/share
```

相对路径示例

相对路径可以变得相当复杂。这里有几个示例，所有的都没有显示出结果的目标路径。请试着推断一下，输入这些命令后，您最终将会转到什么地方：

```
$ cd /bin
$ cd ../usr/share/zoneinfo
$ cd /usr/X11R6/bin
$ cd ../lib/X11
$ cd /usr/bin
$ cd ../bin/../bin
```

现在，试验一次，看看您的推断是否正确。

理解 “.”

在我们结束 cd 的介绍之前，我们还需要讨论一些更多的内容。首先，还有另一个叫 . 的专门的目录。它表示 “当前目录” 。然而该目录不为 cd 命令使用，它通常用来执行一些当前目录中的程序，如下所示：

```
$ ./myprog
```

在上面的示例中，驻留在当前工作目录中的 myprog 可执行文件将被执行。

cd 和主目录

如果我们想要转到主目录，我们可以输入：

```
$ cd
```

没有参数，cd 将转到主目录，对于超级用户来说是 /root，对于一般用户来说通常是 /home/username。但是，如果我们想要指定一个主目录中的文件，将会怎样呢？可能我们想要将一个文件参数 传给 myprog 命令。如果该文件在主目录中，我们可以输入：

```
$ ./myprog /home/drobbins/myfile.txt
```

但是，使用像这样的绝对路径并不总是很方便。幸好，我们可以使用 ~（代字符）字符来完成同样的事：

```
$ ./myprog ~/myfile.txt
```

其他用户的主目录 Bash 将把单独的 ~ 扩展为指向主目录，然而您还可以用它来指向其他用户的主目录。例如，如果我们想要引用 fred 的主目录中的名为 fredsfile.txt 的文件，可以输入：

```
$ ./myprog ~fred/fredsfile.txt
```

使用 Linux 命令

介绍 ls

现在，我们将快速地看一看 ls 命令。很可能，您已经很熟悉 ls，并且知道只输入 ls 本

身将列出当前工作目录的内容： $">[b]通过指定 -a 选项，您可以看到目录中的所有文件，包括隐藏文件 — 那些以 . 开头的文件。您可以在下面的示例中看到，ls -a 将显示 . 和 .. 专门的目录链接：$">[b]递归和索引节点清单。

您可以使用 -d 来查看目录本身，而您还可以用 -R 来完成相反的工作 — 不仅只查看一个目录内部，而且要递归地查看该目录内所有的目录内部！我们将不会有对应该选项的任何示例输出（因为它一般占很大的篇幅），但是为了感觉一下 它是怎样工作的，您可以试几个 ls -R 和 ls -Rl 命令。最后，ls 的 -i 选项可以用来在清单中显示文件系统对象的索引节点号：

```
$ ls -i /usr
1409 X11R6 314258 i686-linux
43090 libexec 13394 sbin
1417 bin 1513 i686-pc-linux-gnu
5120 local 13408 share
8316 distfiles 1517 include
776 man 23779 src
43 doc 1386 info 93892 portage
36737 ssl
70744 gentoo-x86 1585 lib 5132
portage.old 784 tmp
```

理解索引节点，第 1 部分

文件系统的每个对象都分配到一个独一无二的索引，叫做索引节点号。这可能看起来微不足道，但是理解索引节点对于理解许多文件系统操作来说很重 要。例如，请考虑出现在每个目录中的 . 和 .. 链接。为了完全理解 .. 目录实际上是什么，我们将先来看一看 /usr/local 的索引节点号：

```
$ ls -id /usr/local
5120 /usr/local
/usr/local
```

目录有一个 5120 索引节点号。现在，我们来看一 看 /usr/local/bin/.. 的索引节点号：

```
$ ls -id /usr/local/bin/..
5120 /usr/local/bin/..
```

您可以看到，/usr/local/bin/..具有和/usr/local 相同的索引节点号！这就是我们抓住的问题的实质。过去，我们认为 /usr/local 是这个目录本身。

现在，我们发现索引节点 5120 实际上是这个目录，并且我们发现了指向该索引节点的两个目录条目（叫做“链接”）。/usr/local 和 /usr/local/bin/..都链接到索引节点 5120。虽然索引节点 5120 只在磁盘中的一地方存在，但是多个目录条目都链接到它上面。事实上，通过使用 ls -dl 命令，我们可以看到索引节点 5120 被引用的总次数

```
$ ls -dl /usr/local
drwxr-xr-x 8 root root 240 Dec 22 20:
57 /usr/local
```

如果我们看一看从左起的第二栏，我们可以看到目录 /usr/local（索引节点 5120）被引用了 8 次。在我的系统中，引用该索引节点的不同路径有这些：

```
/usr/local
/usr/local/.
/usr/local/bin/..
/usr/local/games/..
/usr/local/lib/..
/usr/local/sbin/..
/usr/local/share/..
/usr/local/src/..
```

mkdir

我们来快速地看一看 mkdir 命令，它可以用来创建新目录。下面的示例创建了三个新目录：tic、tac 和 toe，都在 /tmp 下：

```
$ cd /tmp
$ mkdir tic tac toe
```

缺省情况下，mkdir 不会为您创建父目录；邻接的上一元素的完整路径必须存在。因此，如果您想要创建目录 won/der/ful，您将需要发出三个单独的 mkdir 命令：

```
$ mkdir won/der/ful
mkdir: cannot create directory
`won/der/ful': No such file or directory
$ mkdir won
$ mkdir won/der
$ mkdir won/der/ful
```

mkdir -p

然而，mkdir 有一个很方便的-p 选项，该选项告诉 mkdir 创建所有缺少的父目录，如下所示:

```
$ mkdir -p easy/as/pie
```

总之，非常简单。要学习更多关于 mkdir 命令的知识，请输入 man mkdir 来阅读手册页。除 cd（它内置在 bash 中）之外，这几乎适用于这里所涉及的所有命令（比如 man ls）。

touch

现在，我们将要快速地看一看 cp 和 mv 命令，这些命令用来复制、重命名以及移动文件和目录。为了开始该概述，我们将首先用 touch 命令在 /tmp 中创建一个文件：

```
$ cd /tmp
$ touch copyme
```

如果文件存在，touch 命令将更新文件的“mtime”（请回想 ls -l 输出中的第六栏）。如果文件不存在，那么将创建一个新的空文件。现在您应该有一个大小为零的 /tmp/copyme 文件。

echo 和重定向

既然文件存在，我们来把一些数据添加到文件中。我们可以使用 echo 命令来完成，它

带有自己参数，并且把这些参数打印到标准输出。首先，单独的 echo 命令是这样的：

```
$ echo \"firstfile\"
firstfile
```

带有输出重定向的同样的 echo 命令为：

```
$ echo \"firstfile\" > copyme
```

大于符号告诉 shell 将 echo 的输出写到名为 copyme 的文件中。如果该文件不存在，将创建这个文件；如果该文件存在，将覆盖这个文件。通过输入 ls -l，我们可以看到 copyme 文件为 10 个字节长，因为它包括 firstfile 这个词和换行符：

```
$ ls -l copyme
-rw-r--r-- 1 root root 10 Dec 28 14:13 copyme
```

cat 和 cp

为了在终端显示文件的内容，要使用 cat 命令：

```
$ cat copyme
firstfile
```

现在，我们可以使用 cp 命令的基本调用来由原始的 copyme 文件创建 copiedme 文件：

```
$ cp copyme copiedme
```

通过观察，我们发现它们确实是相互独立的文件；它们的索引节点号不同：

```
$ ls -i copyme copiedme
648284 copiedme 650704 copyme
```

mv

现在，我们来用“mv”命令将“copiedme”重命名为“movedme”。其索引节点号将仍然是同一个；但是，指向该索引节点的文件名将改变。

```
$ mv copiedme movedme
$ ls -i movedme
648284 movedme
```

只要目标文件和源文件驻留在同一文件系统上，被移动的文件的索引节点号就将仍然不变。在本教程系列的第 3 部分，我们将进一步看一下文件系统。

创建链接和删除文件

硬链接

当谈及目录条目和索引节点之间关系时，我们提到了链接这个术语。Linux 实际有两种链接。到此为止我们所讨论的这种链接叫硬链接。一个给定的索引节点可以有任意数目的硬链接，该索引节点一直存在于文件系统，直到所有的硬链接消 失。可以使用 ln 命令来创建新的硬链接

```
$ cd /tmp
$ touch firstlink
```

```
$ ln firstlink secondlink
$ ls -i firstlink secondlink
15782 firstlink 15782 secondlink
```

您可以看到，硬链接工作于索引节点级别，指向特殊的文件。在 Linux 系统上，硬链接有几个局限性。第一，您只能给文件建立硬链接，而不能给目录建立硬链接。的确如此；即便 . 和 .. 是系统给目录创建的硬链接，也不允许您（“root”用户也不行）创建任何您自己的硬链接。

硬链接的第二个局限性是它们不能跨文件系统。这意味着，如果您的 / 和 /usr 存在于不同的文件系统，您不能创建从 /usr/bin/bash 到 /bin/bash 的链接。

符号链接

实际上，符号链接（symbolic link，或“symlinks”）比硬链接更常用到。符号链接是一种专门的文件类型，在这种文件类型中，链接通过名称引用另一个文件，而不是直接引用索引节点。符号链接不阻止文件被删除；如果目标文件消失，那么符号链接仅仅是不可用，或“被破坏”。

通过将 -s 选项传给 ln，可以创建符号链接。

```
$ ln -s secondlink thirdlink
$ ls -l firstlink secondlink thirdlink
-rw-rw-r-- 2 agriffis agriffis 0 Dec 31 19:
08 firstlink
-rw-rw-r-- 2 agriffis agriffis 0 Dec 31 19:
08 secondlink
lrwxrwxrwx 1 agriffis agriffis 10 Dec 31 19:
39 thirdlink -> secondlink
```

在 ls -l 输出中，可以用三种方式区分符号链接和一般文件。第一，请注意第一栏包含一个 l 字符的输出表明是符号链接。第二，符号链接的大小是目标文件（本例是 secondlink）的字符数。第三，输出的最后一栏显示目标文件名。

符号链接通常比硬链接更灵活。您可以给任何类型的文件系统对象（包括目录）创建符号链接。又因为符号链接的实现是基于路径的（而不是索引节 点），所以创建指向另一个文件系统上的对象的符号链接是完全可行的。但是，这一事实也使符号链接理解起来很复杂。请考虑我们想要在/tmp 中创建一个指向 /usr/local/bin 的链接的情况。我们应该输入：

```
$ ln -s /usr/local/bin bin1
$ ls -l bin1
lrwxrwxrwx 1 root root 14 Jan 1 15:
42 bin1 -> /usr/local/bin
```

或者还可以输入：

```
$ ln -s ../usr/local/bin bin2
$ ls -l bin2
lrwxrwxrwx 1 root root 16 Jan 1 15:
43 bin2 -> ../usr/local/bin
```

您可以看到，两个符号链接都指向同一目录。但是，如果我们的第二个符号链接在任何

时刻被移动到另一个目录，由于相对路径的缘故，它将遭到“破坏”。

```
$ ls -l bin2
lrwxrwxrwx 1 root root 16 Jan 1 15:
43 bin2 -> ../usr/local/bin
$ mkdir mynewdir
$ mv bin2 mynewdir
$ cd mynewdir
$ cd bin2
bash: cd: bin2: No such file or directory
```

因为/tmp/usr/local/bin 这个目录不存在，我们不能再把目录转到 bin2；换句话说，bin2 现在被破坏了。

由于这个原因，有时避免用相对路径信息来创建符号链接是个好主意。但是，在许多情况下，相对的符号链接很管用。请考虑一个示例，在这个示例中您想要给 /usr/bin 中的一个程序创建一个别名：

```
# ls -l /usr/bin/keychain
-rwxr-xr-x 1 root root 10150 Dec 12 20:09 /usr/bin/keychain
```

作为 root 用户，您可能想要给“keychain”创建一个别名，比如“kc”。在这个示例中，我们有 root 访问权，由 bash 提示符改变为“#”可以证明。我们之所以需要 root 访问权是因为一般用户不能在 /usr/bin 中创建文件。作为 root 用户，我们可以像下面这样给 keychain 创建一个别名：

```
# cd /usr/bin
# ln -s /usr/bin/keychain kc
```

当这个解决方法起作用时，如果我们想要把两个文件都移到 /usr/local/bin 时，它将会出现问题。

```
# mv /usr/bin/keychain
/usr/bin/kc /usr/local/bin
```

因为在符号链接中，我们使用了绝对路径，而我们的 kc 符号链接仍然指向 /usr/bin/keychain，它已不存在了——另一个被破坏的符 号链接。符号链接中的相对路径和绝对路径都各具优点，您应该使用适合于您的特殊应用的路径类型。一般情况下，相对路径或绝对路径都能工作得很好。在这种情 况下，下面的示例将起作用：

```
# cd /usr/bin
# ln -s keychain kc
# ls -l kc
lrwxrwxrwx 1 root root 8 Jan 5 12:
40 kc -> keychain
```

rm

既然我们知道怎样使用 cp、mv 和 ln，现在我们该学习怎样把对象从文件系统中删除了。通常，这用 rm 命令来完成。要删除文件，只需在命令行中指定它们:

```
$ cd /tmp
$ touch file1 file2
```

```
$ ls -l file1 file2
-rw-r--r-- 1 root root 0 Jan 1 16:41 file1
-rw-r--r-- 1 root root 0 Jan 1 16:41 file2
$ rm file1 file2
$ ls -l file1 file2
ls: file1: No such file or directory
ls: file2: No such file or directory
```

rmdir

要删除目录，您有两种选择。您可以删除目录中所有的对象，然后使用 rmdir 来删除目录本身：

```
$ mkdir mydir
$ touch mydir/file1
$ rm mydir/file1
$ rmdir mydir
```

rm 和目录

或者，您可以使用 rm 命令的 recursive force 选项来告诉 rm 删除您指定的目录以及目录中包含的所有对象：

```
$ rm -rf mydir
```

一般情况下，rm -rf 是删除目录树的首选方法。在使用 rm -rf 时要十分小心，因为它的功能可以被很好地利用，也可能会因使用不当造成恶果。

介绍通配符

在您日常的 Linux 使用中，有很多时候您可能需要一次对多个文件系统对象执行单一操作（比如 rm）。在这些情况下，在命令行中输入许多文件通常让人感到厌烦，为了解决这个问题，您可以利用 Linux 内置的通配符支持。这种支持也叫做“globbing”（由于历史原因），允许您通过使用通配符模式一次指定多个文件。

Bash 和其它 Linux 命令将通过在磁盘上查找并找到任何与之匹配的文件来解释这种模式。因此，如果在当前工作目录中，您有从 file1 到 file8 的文件，那么您可以输入下面的命令来删除这些文件：

```
$ rm file[1-8]
```

或者，如果您只想要删除文件名以 file 开头的所有文件，您可以输入：

```
$ rm file*
```

理解不匹配

或者，如果您想要列出 /etc 中以 g 开头的所有文件系统对象，您可以输入：

```
$ ls -d /etc/g*
/etc/gconf /etc/ggi /etc/gimp /etc/gnome
/etc/gnome-vfs-mime-magic /etc/gpm
/etc/group /etc/group-
```

现在，如果您指定了没有任何文件系统对象与之匹配的模式，会怎么样呢？在下面的示

例中，我们试图列出 /usr/bin 中以 asdf 开头并且以 jkl 结尾的所有文件：

```
$ ls -d /usr/bin/asdf*jkl
ls: /usr/bin/asdf*jkl:
No such file or directory
```

这里是对所发生情况的说明。通常，当我们指定一种模式时，该模式与底层系统上的一个或多个文件匹配，bash 以空格隔开的所有匹配对象的列表来替换该模式。

但是，当模式不能找到匹配对象时，bash 将不理会参数、通配符等等，保留原样。因此，当 “ls”不能找到文件 /usr/bin/asdf*jkl 时，它会报错。此处的有效的规则是：glob 模式只在与文件系统中的对象匹配时才可以进行扩展。