1. 键盘高级操作技巧

I often kiddingly describe Unix as “the operating system for people who like to type.” Of course, the fact that it even has a command line is a testament to that. But command line users don’t like to type that much. Why else would so many commands have such short names like cp, ls, mv, and rm? In fact, one of the most cherished goals of the command line is laziness; doing the most work with the fewest number of keystrokes. Another goal is never having to lift your fingers from the keyboard, never reaching for the mouse. In this chapter, we will look at bash features that make keyboard use faster and more efficient.

开玩笑地说，我经常把 Unix 描述为“这个操作系统是为喜欢敲键盘的人们服务的。” 当然，Unix 甚至还有一个命令行，这个事实是个确凿的证据，证明了我所说的话。 但是命令行用户不喜欢敲入那么多字。那又为什么如此多的命令会有这样简短的命令名，像 cp，ls，mv，和 rm？事实上，命令行最为珍视的目标之一就是懒惰；用最少的击键次数 来完成最多的工作。另一个目标是你的手指永远不必离开键盘，永不触摸鼠标。在这一 章节，我们将看一下 bash 特性，这些特性使键盘使用起来更加迅速，更加高效。

The following commands will make an appearance:

以下命令将会露面：

* clear - Clear the screen
* history - Display the contents of the history list
* clear － 清空屏幕
* history － 显示历史列表内容

**命令行编辑**

bash uses a library (a shared collection of routines that different programs can use) called Readline to implement command line editing. We have already seen some of this. We know, for example, that the arrow keys move the cursor but there are many more features. Think of these as additional tools that we can employ in our work. It’s not important to learn all of them, but many of them are very useful. Pick and choose as desired.

Bash 使用了一个名为 Readline 的库（共享的线程集合，可以被不同的程序使用）， 来实现命令行编辑。我们已经看到一些例子。我们知道，例如，箭头按键可以移动光标， 此外还有许多特性。想想这些额外的工具，我们可以在工作中使用。学会所有的特性 并不重要，但许多特性非常有帮助。选择自己需要的特性。

Note: Some of the key sequences below (particularly those which use the Alt key) may be intercepted by the GUI for other functions. All of the key sequences should work properly when using a virtual console.

注意：下面一些按键组合（尤其使用 Alt 键的组合），可能会被 GUI 拦截来触发其它的功能。 当使用虚拟控制台时，所有的按键组合都应该正确地工作。

**移动光标**

The following table lists the keys used to move the cursor:

下表列出了移动光标所使用的按键：

|  |  |
| --- | --- |
| *Table 9-1: Cursor Movement Commands* | |
| Key | Action |
| Ctrl-a | Move cursor to the beginning of the line. |
| Ctrl-e | Move cursor to the end of the line. |
| Ctrl-f | Move cursor forward one character;same as the right arrow key. |
| Ctrl-b | Move cursor backward one character;same as the left arrow key. |
| Alt-f | Move cursor forward one word. |
| Alt-b | Move cursor backward one word. |
| Ctrl-l | Clear the screen and move the cursor to the top left corner. The clear command does the same thing. |

|  |  |
| --- | --- |
| *表9-1: 光标移动命令* | |
| 按键 | 行动 |
| Ctrl-a | 移动光标到行首。 |
| Ctrl-e | 移动光标到行尾。 |
| Ctrl-f | 光标前移一个字符；和右箭头作用一样。 |
| Ctrl-b | 光标后移一个字符；和左箭头作用一样。 |
| Alt-f | 光标前移一个字。 |
| Alt-b | 光标后移一个字。 |
| Ctrl-l | 清空屏幕，移动光标到左上角。clear 命令完成同样的工作。 |

**修改文本**

Table 9-2 lists keyboard commands that are used to edit characters on the command line.

表9－2列出了键盘命令，这些命令用来在命令行中编辑字符。

|  |  |
| --- | --- |
| *Table 9-2: Text Editing Commands* | |
| Key | Action |
| Ctrl-d | Delete the character at the cursor location |
| Ctrl-t | Transpose(exchange)the character at the cursor location with the one preceding it. |
| Alt-t | Transpose the word at the cursor location with the one preceding it. |
| Alt-l | Convert the characters from the cursor location to the end of the word to lowercase. |
| Alt-u | Convert the characters from the cursor location to the end of the word to uppercase. |

|  |  |
| --- | --- |
| *表9-2: 文本编辑命令* | |
| 按键 | 行动 |
| Ctrl-d | 删除光标位置的字符。 |
| Ctrl-t | 光标位置的字符和光标前面的字符互换位置。 |
| Alt-t | 光标位置的字和其前面的字互换位置。 |
| Alt-l | 把从光标位置到字尾的字符转换成小写字母。 |
| Alt-u | 把从光标位置到字尾的字符转换成大写字母。 |

**剪切和粘贴文本**

The Readline documentation uses the terms killing and yanking to refer to what we would commonly call cutting and pasting. Items that are cut are stored in a buffer called the kill-ring.

Readline 的文档使用术语 killing 和 yanking 来指我们平常所说的剪切和粘贴。 剪切下来的本文被存储在一个叫做剪切环(kill-ring)的缓冲区中。

|  |  |
| --- | --- |
| *Table 9-3: Cut And Paste Commands* | |
| Key | Action |
| Ctrl-k | Kill text from the cursor location to the end of line. |
| Ctrl-u | Kill text from the cursor location to the beginning of the line. |
| Alt-d | Kill text from the cursor location to the end of the current word. |
| Alt-Backspace | Kill text from the cursor location to the beginning of the word. If the cursor is at the beginning of a word, kill the previous word. |
| Ctrl-y | Yank text from the kill-ring and insert it at the cursor location. |

|  |  |
| --- | --- |
| *表9-3: 剪切和粘贴命令* | |
| 按键 | 行动 |
| Ctrl-k | 剪切从光标位置到行尾的文本。 |
| Ctrl-u | 剪切从光标位置到行首的文本。 |
| Alt-d | 剪切从光标位置到词尾的文本。 |
| Alt-Backspace | 剪切从光标位置到词头的文本。如果光标在一个单词的开头，剪切前一个单词。 |
| Ctrl-y | 把剪切环中的文本粘贴到光标位置。 |

The Meta Key

元键

If you venture into the Readline documentation, which can be found in the READLINE section of the bash man page, you will encounter the term “meta key.” On modern keyboards this maps to the Alt key but it wasn’t always so.

如果你冒险进入到 Readline 的文档中，你会在 bash 手册页的 READLINE 段落， 遇到一个术语”元键”（meta key）。在当今的键盘上，这个元键是指 Alt 键，但 并不总是这样。

Back in the dim times (before PCs but after Unix) not everybody had their own computer. What they might have had was a device called a terminal. A terminal was a communication device that featured a text display screen and a keyboard and just enough electronics inside to display text characters and move the cursor around. It was attached (usually by serial cable) to a larger computer or the communication network of a larger computer. There were many different brands of terminals and they all had different keyboards and display feature sets. Since they all tended to at least understand ASCII, software developers wanting portable applications wrote to the lowest common denominator. Unix systems have a very elaborate way of dealing with terminals and their different display features. Since the developers of Readline could not be sure of the presence of a dedicated extra control key, they invented one and called it “meta.” While the Alt key serves as the meta key on modern keyboards, you can also press and release the Esc key to get the same effect as holding down the Alt key if you’re still using a terminal (which you can still do in Linux!).

回到昏暗的年代（在 PC 之前 Unix 之后），并不是每个人都有他们自己的计算机。 他们可能有一个叫做终端的设备。一个终端是一种通信设备，它以一个文本显示 屏幕和一个键盘作为其特色，它里面有足够的电子器件来显示文本字符和移动光标。 它连接到（通常通过串行电缆）一个更大的计算机或者是一个大型计算机的通信 网络。有许多不同的终端产品商标，它们有着不同的键盘和特征显示集。因为它们 都倾向于至少能理解 ASCII，所以软件开发者想要符合最低标准的可移植的应用程序。 Unix 系统有一个非常精巧的方法来处理各种终端产品和它们不同的显示特征。因为 Readline 程序的开发者们，不能确定一个专用多余的控制键的存在，他们发明了一个 控制键，并把它叫做”元”（”meta”）。然而在现代的键盘上，Alt 键作为元键来服务。 如果你仍然在使用终端（在 Linux 中，你仍然可以得到一个终端），你也可以按下和 释放 Esc 键来得到如控制 Alt 键一样的效果。

**自动补全**

Another way that the shell can help you is through a mechanism called completion. Completion occurs when you press the tab key while typing a command. Let’s see how this works. Given a home directory that looks like this:

shell 能帮助你的另一种方式是通过一种叫做自动补全的机制。当你敲入一个命令时， 按下 tab 键，自动补全就会发生。让我们看一下这是怎样工作的。给出一个看起来 像这样的家目录：

[me@linuxbox ~]$ ls

Desktop ls-output.txt Pictures Templates Videos

....

Try typing the following but don’t press the Enter key:

试着输入下面的命令，但不要按下 Enter 键：

[me@linuxbox ~]$ ls l

Now press the tab key:

现在按下 tab 键：

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

See how the shell completed the line for you? Let’s try another one. Again, don’t press Enter:

看一下 shell 是怎样补全这一行的？让我们再试试另一个例子。这回，也 不要按下 Enter:

[me@linuxbox ~]$ ls D

Press tab:

按下 tab:

[me@linuxbox ~]$ ls D

No completion, just a beep. This happened because “D” matches more than one entry in the directory. For completion to be successful, the “clue” you give it has to be unambiguous. If we go further:

没有补全，只是嘟嘟响。因为”D”不止匹配目录中的一个条目。为了自动补全执行成功， 你给它的”线索”必须不模棱两可。如果我们继续输入：

[me@linuxbox ~]$ ls Do

Then press tab:

然后按下 tab：

[me@linuxbox ~]$ ls Documents

The completion is successful.

自动补全成功了。

While this example shows completion of pathnames, which is its most common use, completion will also work on variables (if the beginning of the word is a “$”), user names (if the word begins with “~”), commands (if the word is the first word on the line.) and host names (if the beginning of the word is “@”). Host name completion only works for host names listed in /etc/hosts.

这个实例展示了路径名自动补全，这是最常用的形式。自动补全也能对变量起作用（如果 字的开头是一个”$”），用户名字（单词以”~”开始），命令（如果单词是一行的第一个单词）， 和主机名（如果单词的开头是”@”）。主机名自动补全只对包含在文件/etc/hosts 中的主机名有效。

There are a number of control and meta key sequences that are associated with completion:

有一系列的控制和元键序列与自动补全相关联：

|  |  |
| --- | --- |
| *Table 9-4: Completion Commands* | |
| Key | Action |
| Alt-? | Display list of possible completions. On most systems you can also do this by pressing the tab key a second time, which is much easier. |
| Alt-\* | Insert all possible completions. This is useful when you want to use more than one possible match. |

|  |  |
| --- | --- |
| *表9-4: 自动补全命令* | |
| 按键 | 行动 |
| Alt-? | 显示可能的自动补全列表。在大多数系统中，你也可以完成这个通过按 两次 tab 键，这会更容易些。 |
| Alt-\* | 插入所有可能的自动补全。当你想要使用多个可能的匹配项时，这个很有帮助。 |

Programmable Completion

可编程自动补全

Recent versions of bash have a facility called programmable completion. Programmable completion allows you (or more likely, your distribution provider) to add additional completion rules. Usually this is done to add support for specific applications. For example it is possible to add completions for the option list of a command or match particular file types that an application supports. Ubuntu has a fairly large set defined by default. Programmable completion is implemented by shell functions, a kind of mini shell script that we will cover in later chapters. If you are curious, try:

目前的 bash 版本有一个叫做可编程自动补全工具。可编程自动补全允许你（更可能是，你的 发行版提供商）来加入额外的自动补全规则。通常需要加入对特定应用程序的支持，来完成这个 任务。例如，有可能为一个命令的选项列表，或者一个应用程序支持的特殊文件类型加入自动补全。 默认情况下，Ubuntu 已经定义了一个相当大的规则集合。可编程自动补全是由 shell 函数实现的，shell 函数是一种小巧的 shell 脚本，我们会在后面的章节中讨论到。如果你感到好奇，试一下：

**set | less**

and see if you can find them. Not all distributions include them by default.

查看一下如果你能找到它们的话。默认情况下，并不是所有的发行版都包括它们。

**利用历史命令**

As we discovered in Chapter 2, bash maintains a history of commands that have been entered. This list of commands is kept in your home directory in a file called .bash\_history. The history facility is a useful resource for reducing the amount of typing you have to do, especially when combined with command line editing.

正如我们在第二章中讨论到的，bash 维护着一个已经执行过的命令的历史列表。这个命令列表 被保存在你家目录下，一个叫做 .bash\_history 的文件里。这个 history 工具是个有用资源， 因为它可以减少你敲键盘的次数，尤其当和命令行编辑联系起来时。

**搜索历史命令**

At any time, we can view the contents of the history list by:

在任何时候，我们都可以浏览历史列表的内容，通过：

[me@linuxbox ~]$ history | less

By default, bash stores the last five hundred commands you have entered. We will see how to adjust this value in a later chapter. Let’s say we want to find the commands we used to list /usr/bin. One way we could do this:

在默认情况下，bash 会存储你所输入的最后 500 个命令。在随后的章节里，我们会知道 怎样调整这个数值。比方说我们想要找到列出目录 /usr/bin 内容的命令。一种方法，我们可以这样做：

[me@linuxbox ~]$ history | grep /usr/bin

And let’s say that among our results we got a line containing an interesting command like this:

比方说在我们的搜索结果之中，我们得到一行，包含了有趣的命令，像这样；

88 ls -l /usr/bin > ls-output.txt

The number “88” is the line number of the command in the history list. We could use this immediately using another type of expansion called history expansion. To use our discovered line we could do this:

数字 “88” 是这个命令在历史列表中的行号。随后在使用另一种展开类型时，叫做 历史命令展开，我们会用到这个数字。我们可以这样做，来使用我们所发现的行：

[me@linuxbox ~]$ !88

bash will expand “!88” into the contents of the eighty-eighth line in the history list. There are other forms of history expansion that we will cover a little later. bash also provides the ability to search the history list incrementally. This means that we can tell bash to search the history list as we enter characters, with each additional character further refining our search. To start incremental search type Ctrl-r followed by the text you are looking for. When you find it, you can either type Enter to execute the command or type Ctrl-j to copy the line from the history list to the current command line. To find the next occurrence of the text (moving “up” the history list), type Ctrl-r again. To quit searching, type either Ctrl-g or Ctrl-c. Here we see it in action:

bash 会把 “!88” 展开成为历史列表中88行的内容。还有其它的历史命令展开形式，我们一会儿 讨论它们。bash 也具有按递增顺序来搜索历史列表的能力。这意味着随着字符的输入，我们 可以告诉 bash 去搜索历史列表，每一个附加字符都进一步提炼我们的搜索。启动递增搜索， 输入 Ctrl-r，其后输入你要寻找的文本。当你找到它以后，你可以敲入 Enter 来执行命令， 或者输入 Ctrl-j，从历史列表中复制这一行到当前命令行。再次输入 Ctrl-r，来找到下一个 匹配项（向上移动历史列表）。输入 Ctrl-g 或者 Ctrl-c，退出搜索。实际来体验一下：

[me@linuxbox ~]$

First type Ctrl-r:

首先输入 Ctrl-r:

(reverse-i-search)`':

The prompt changes to indicate that we are performing a reverse incremental search. It is “reverse” because we are searching from “now” to some time in the past. Next, we start typing our search text. In this example “/usr/bin”:

提示符改变，显示我们正在执行反向递增搜索。搜索过程是”反向的”，因为我们按照从”现在”到过去 某个时间段的顺序来搜寻。下一步，我们开始输入要查找的文本。在这个例子里是 “/usr/bin”：

(reverse-i-search)`/usr/bin': ls -l /usr/bin > ls-output.txt

Immediately, the search returns our result. With our result, we can execute the command by pressing Enter, or we can copy the command to our current command line for further editing by typing Ctrl-j. Let’s copy it. Type Ctrl-j:

即刻，搜索返回我们需要的结果。我们可以执行这个命令，按下 Enter 键，或者我们可以复制 这个命令到我们当前的命令行，来进一步编辑它，输入 Ctrl-j。复制它，输入 Ctrl-j：

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

Our shell prompt returns and our command line is loaded and ready for action! The table below lists some of the keystrokes used to manipulate the history list:

我们的 shell 提示符重新出现，命令行加载完毕，正准备行动！下表列出了一些按键组合， 这些按键用来操作历史列表：

|  |  |
| --- | --- |
| *Table 9-5: History Commands* | |
| Key | Action |
| Ctrl-p | Move to the previous history entry. Same action as the up arrow. |
| Ctrl-n | Move to the next history entry. Same action as the down arrow. |
| Alt-< | Move to the beginning (top) of the history list. |
| Alt-> | Move to the end (bottom) of the history list, i.e., the current command line. |
| Ctrl-r | Reverse incremental search. Searches incrementally from the current command line up the history list. |
| Alt-p | Reverse search, non-incremental. With this key, type in the search string and press enter before the search is performed. |
| Alt-n | Forward search, non-incremental. |
| Ctrl-o | Execute the current item in the history list and advance to the next one. This is handy if you are trying to re-execute a sequence of commands in the history list. |

|  |  |
| --- | --- |
| *表9-5: 历史命令* | |
| 按键 | 行为 |
| Ctrl-p | 移动到上一个历史条目。类似于上箭头按键。 |
| Ctrl-n | 移动到下一个历史条目。类似于下箭头按键。 |
| Alt-< | 移动到历史列表开头。 |
| Alt-> | 移动到历史列表结尾，即当前命令行。 |
| Ctrl-r | 反向递增搜索。从当前命令行开始，向上递增搜索。 |
| Alt-p | 反向搜索，不是递增顺序。输入要查找的字符串，然后按下 Enter，执行搜索。 |
| Alt-n | 向前搜索，非递增顺序。 |
| Ctrl-o | 执行历史列表中的当前项，并移到下一个。如果你想要执行历史列表中一系列的命令，这很方便。 |

**历史命令展开**

The shell offers a specialized type of expansion for items in the history list by using the “!” character. We have already seen how the exclamation point can be followed by a number to insert an entry from the history list. There are a number of other expansion features:

通过使用 “!” 字符，shell 为历史列表中的命令，提供了一个特殊的展开类型。我们已经知道一个感叹号 ，其后再加上一个数字，可以把来自历史列表中的命令插入到命令行中。还有一些其它的展开特性：

|  |  |
| --- | --- |
| *Table 9-6: History Expansion Commands* | |
| Sequence | Action |
| !! | Repeat the last command. It is probably easier to press up arrow and enter. |
| !number | Repeat history list item number. |
| !string | Repeat last history list item starting with string. |
| !?string | Repeat last history list item containing string. |

|  |  |
| --- | --- |
| *表9-6: 历史展开命令* | |
| 序列 | 行为 |
| !! | 重复最后一次执行的命令。可能按下上箭头按键和 enter 键更容易些。 |
| !number | 重复历史列表中第 number 行的命令。 |
| !string | 重复最近历史列表中，以这个字符串开头的命令。 |
| !?string | 重复最近历史列表中，包含这个字符串的命令。 |

I would caution against using the “!string” and “!?string” forms unless you are absolutely sure of the contents of the history list items.

应该小心谨慎地使用 “!string” 和 “!?string” 格式，除非你完全确信历史列表条目的内容。

There are many more elements available in the history expansion mechanism, but this subject is already too arcane and our heads may explode if we continue. The HISTORY EXPANSION section of the bash man page goes into all the gory details. Feel free to explore!

在历史展开机制中，还有许多可利用的特点，但是这个题目已经太晦涩难懂了， 如果我们再继续讨论的话，我们的头可能要爆炸了。bash 手册页的 HISTORY EXPANSION 部分详尽地讲述了所有要素。

script

脚本

In addition to the command history feature in bash, most Linux distributions include a program called script that can be used to record an entire shell session and store it in a file. The basic syntax of the command is:

除了 bash 中的命令历史特性，许多 Linux 发行版包括一个叫做 script 的程序， 这个程序可以记录整个 shell 会话，并把 shell 会话存在一个文件里面。这个命令的基本语法是：

**script [file]**

where file is the name of the file used for storing the recording. If no file is specified, the file typescript is used. See the script man page for a complete list of the program’s options and features.

命令中的 file 是指用来存储 shell 会话记录的文件名。如果没有指定文件名，则使用文件 typescript。查看脚本的手册页，可以得到一个关于 script 程序选项和特点的完整列表。

**总结归纳**

In this chapter we have covered some of the keyboard tricks that the shell provides to help hardcore typists reduce their workloads. I suspect that as time goes by and you become more involved with the command line, you will refer back to this chapter to pick up more of these tricks. For now, consider them optional and potentially helpful.

在这一章中，我们已经讨论了一些由 shell 提供的键盘操作技巧，这些技巧是来帮助打字员减少工作量的。 随着时光流逝，你和命令行打交道越来越多，我猜想你会重新翻阅这一章的内容，学会更多的技巧。 目前，你就认为它们是可选的，潜在地有帮助的。