第十二章 shell环境  
As we discussed earlier, the shell maintains a body of information during our shell session called the environment. Data stored in the environment is used by programs to determine facts about our configuration. While most programs use configuration files to store program settings, some programs will also look for values stored in the environment to adjust their behavior. Knowing this, we can use the environment to customize our shell experience.

正如我们之前所讨论到的，shell 在 shell 会话中维护着大量的信息，这些信息称为 (shell) 环境。 存储在 shell 环境中的数据被程序用来确定配置属性。虽然大多数程序用配置文件来存储程序设置， 某些程序也会查找存储在 shell 环境中的数值来调整他们的行为。知道了这些，我们就可以用 shell 环境 来自定制 shell 经历。

In this chapter, we will work with the following commands:

* printenv – Print part or all of the environment
* set – Set shell options
* export – Export environment to subsequently executed programs
* alias – Create an alias for a command

在这一章，我们将用到以下命令：

* printenv - 打印部分或所有的环境变量
* set - 设置 shell 选项
* export — 导出环境变量，让随后执行的程序知道。
* alias - 创建命令别名

**什么存储在环境变量中？**

The shell stores two basic types of data in the environment, though, with bash, the types are largely indistinguishable. They are environment variables and shell variables. Shell variables are bits of data placed there by bash, and environment variables are basically everything else. In addition to variables, the shell also stores some programmatic data, namely aliases and shell functions. We covered aliases in Chapter 6, and shell functions (which are related to shell scripting) will be covered in Part 5.

shell 在环境中存储了两种基本类型的数据，虽然对于 bash 来说，很大程度上这些类型是不可 辨别的。它们是环境变量和 shell 变量。Shell 变量是由 bash 存放的少量数据，而剩下的基本上 都是环境变量。除了变量，shell 也存储了一些可编程的数据，命名为别名和 shell 函数。我们 已经在第六章讨论了别名，而 shell 函数（涉及到 shell 脚本）将会在第五部分叙述。

**检查环境变量**

We can use either the set builtin in bash or the printenv program to see what is stored in the environment. The set command will show both the shell and environment variables, while printenv will only display the latter. Since the list of environment contents will be fairly long, it is best to pipe the output of either command into less:

我们既可以用 bash 的内部命令 set，或者是 printenv 程序来查看什么存储在环境当中。set 命令可以 显示 shell 和环境变量两者，而 printenv 只是显示环境变量。因为环境变量内容列表相当长，所以最好 把每个命令的输出结果管道到 less 命令：

[me@linuxbox ~]$ printenv | less

Doing so, we should get something that looks like this:

执行以上命令之后，我们应该能得到类似以下内容：

KDE\_MULTIHEAD=false

SSH\_AGENT\_PID=6666

HOSTNAME=linuxbox

GPG\_AGENT\_INFO=/tmp/gpg-PdOt7g/S.gpg-agent:6689:1

SHELL=/bin/bash

TERM=xterm

XDG\_MENU\_PREFIX=kde-

HISTSIZE=1000

XDG\_SESSION\_COOKIE=6d7b05c65846c3eaf3101b0046bd2b00-1208521990.996705

-1177056199

GTK2\_RC\_FILES=/etc/gtk-2.0/gtkrc:/home/me/.gtkrc-2.0:/home/me/.kde/sh

are/config/gtkrc-2.0

GTK\_RC\_FILES=/etc/gtk/gtkrc:/home/me/.gtkrc:/home/me/.kde/share/confi

g/gtkrc

GS\_LIB=/home/me/.fonts

WINDOWID=29360136

QTDIR=/usr/lib/qt-3.3

QTINC=/usr/lib/qt-3.3/include

KDE\_FULL\_SESSION=true

USER=me

LS\_COLORS=no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33;01

:cd=40;33;01:or=01;05;37;41:mi=01;05;37;41:ex=00;32:\\*.cmd=00;32:\\*.exe:

What we see is a list of environment variables and their values. For example, we see a variable called USER, which contains the value “me”. The printenv command can also list the value of a specific variable:

我们所看到的是环境变量及其数值的列表。例如，我们看到一个叫做 USER 的变量，这个变量值是 “me”。printenv 命令也能够列出特定变量的数值：

[me@linuxbox ~]$ printenv USER

me

The set command, when used without options or arguments, will display both the shell and environment variables, as well as any defined shell functions. Unlike printenv, its output is courteously sorted in alphabetical order:

当使用没有带选项和参数的 set 命令时，shell 和环境变量二者都会显示，同时也会显示定义的 shell 函数。不同于 printenv 命令，set 命令的输出结果很礼貌地按照字母顺序排列：

[me@linuxbox ~]$ set | less

It is also possible to view the contents of a variable using the echo command, like this:

也可以通过 echo 命令来查看一个变量的内容，像这样：

[me@linuxbox ~]$ echo $HOME

/home/me

One element of the environment that neither set nor printenv displays is aliases. To see them, enter the alias command without arguments:

如果 shell 环境中的一个成员既不可用 set 命令也不可用 printenv 命令显示，则这个变量是别名。 输入不带参数的 alias 命令来查看它们:

[me@linuxbox ~]$ alias

alias l.='ls -d .\* --color=tty'

alias ll='ls -l --color=tty'

alias ls='ls --color=tty'

alias vi='vim'

alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'

**一些有趣的变量**

The environment contains quite a few variables, and though your environment may differ from the one presented here, you will likely see the following variables in your environment:

shell 环境中包含相当多的变量，虽然你的 shell 环境可能不同于这里展示的，但是你可能会看到 以下变量在你的 shell 环境中：

|  |  |
| --- | --- |
| *Table 12-1: Environment Variables* | |
| Variable | Contents |
| DISPLAY | The name of your display if you are running a graphical environment. Usually this is ":0", meaning the first display generated by the X server. |
| EDITOR | The name of the program to be used for text editing. |
| SHELL | The name of your shell program. |
| HOME | The pathname of your home directory. |
| LANG | Defines the character set and collation order of your language. |
| OLD\_PWD | The previous working directory. |
| PAGER | The name of the program to be used for paging output. This is often set to /usr/bin/less. |
| PATH | A colon-separated list of directories that are searched when you enter the name of a executable program. |
| PS1 | Prompt String 1. This defines the contents of your shell prompt. As we will later see, this can be extensively customized. |
| PWD | The current working directory. |
| TERM | The name of your terminal type. Unix-like systems support many terminal protocols; this variable sets the protocol to be used with your terminal emulator. |
| TZ | Specifies your timezone. Most Unix-like systems maintain the computer's internal clock in Coordinated Universal Time (UTC) and then displays the local time by applying an offset specified by this variable. |
| USER | Your user name. |

|  |  |
| --- | --- |
| *表12-1: 环境变量* | |
| 变量 | 内容 |
| DISPLAY | 如果你正在运行图形界面环境，那么这个变量就是你显示器的名字。通常，它是 ":0"， 意思是由 X 产生的第一个显示器。 |
| EDITOR | 文本编辑器的名字。 |
| SHELL | shell 程序的名字。 |
| HOME | 用户家目录。 |
| LANG | 定义了字符集以及语言编码方式。 |
| OLD\_PWD | 先前的工作目录。 |
| PAGER | 页输出程序的名字。这经常设置为/usr/bin/less。 |
| PATH | 由冒号分开的目录列表，当你输入可执行程序名后，会搜索这个目录列表。 |
| PS1 | Prompt String 1. 这个定义了你的 shell 提示符的内容。随后我们可以看到，这个变量 内容可以全面地定制。 |
| PWD | 当前工作目录。 |
| TERM | 终端类型名。类 Unix 的系统支持许多终端协议；这个变量设置你的终端仿真器所用的协议。 |
| TZ | 指定你所在的时区。大多数类 Unix 的系统按照协调时间时 (UTC) 来维护计算机内部的时钟 ，然后应用一个由这个变量指定的偏差来显示本地时间。 |
| USER | 你的用户名 |

Don’t worry if some of these values are missing. They vary by distribution.

如果缺失了一些变量，不要担心，这些变量会因发行版本的不同而不同。

**如何建立 shell 环境？**

When we log on to the system, the bash program starts, and reads a series of configuration scripts called startup files, which define the default environment shared by all users. This is followed by more startup files in our home directory that define our personal environment. The exact sequence depends on the type of shell session being started. There are two kinds: a login shell session and a non-login shell session.

当我们登录系统后，启动 bash 程序，并且会读取一系列称为启动文件的配置脚本， 这些文件定义了默认的可供所有用户共享的 shell 环境。然后是读取更多位于我们自己家目录中 的启动文件，这些启动文件定义了用户个人的 shell 环境。精确的启动顺序依赖于要运行的 shell 会话 类型。有两种 shell 会话类型：一个是登录 shell 会话，另一个是非登录 shell 会话。

A login shell session is one in which we are prompted for our user name and password; when we start a virtual console session, for example. A non-login shell session typically occurs when we launch a terminal session in the GUI.

登录 shell 会话会提示用户输入用户名和密码；例如，我们启动一个虚拟控制台会话。当我们在 GUI 模式下 运行终端会话时，非登录 shell 会话会出现。

Login shells read one or more startup files as shown in Table 12-2:

登录 shell 会读取一个或多个启动文件，正如表12－2所示：

|  |  |
| --- | --- |
| *Table 12-2: Startup Files For Login Shell Sessions* | |
| File | Contents |
| /etc/profile | A global configuration script that applies to all users. |
| ~/.bash\_profile | A user's personal startup file. Can be used to extend or override settings in the global configuration script. |
| ~/.bash\_login | If ~/.bash\_profile is not found, bash attempts to read this script. |
| ~/.profile | If neither ~/.bash\_profile nor ~/.bash\_login is found, bash attempts to read this file. This is the default in Debian-based distributions, such as Ubuntu. |

|  |  |
| --- | --- |
| *表12-2: 登录 shell 会话的启动文件* | |
| 文件 | 内容 |
| /etc/profile | 应用于所有用户的全局配置脚本。 |
| ~/.bash\_profile | 用户私人的启动文件。可以用来扩展或重写全局配置脚本中的设置。 |
| ~/.bash\_login | 如果文件 ~/.bash\_profile 没有找到，bash 会尝试读取这个脚本。 |
| ~/.profile | 如果文件 ~/.bash\_profile 或文件 ~/.bash\_login 都没有找到，bash 会试图读取这个文件。 这是基于 Debian 发行版的默认设置，比方说 Ubuntu。 |

Non-login shell sessions read the following startup files:

非登录 shell 会话会读取以下启动文件：

|  |  |
| --- | --- |
| *Table 12-3: Startup Files For Non-Login Shell Sessions* | |
| File | Contents |
| /etc/bash.bashrc | A global configuration script that applies to all users. |
| ~/.bashrc | A user's personal startup file. Can be used to extend or override settings in the global configuration script. |

|  |  |
| --- | --- |
| *表12-3: 非登录 shell 会话的启动文件* | |
| 文件 | 内容 |
| /etc/bash.bashrc | 应用于所有用户的全局配置文件。 |
| ~/.bashrc | 用户私有的启动文件。可以用来扩展或重写全局配置脚本中的设置。 |

In addition to reading the startup files above, non-login shells also inherit the environment from their parent process, usually a login shell.

除了读取以上启动文件之外，非登录 shell 会话也会继承它们父进程的环境设置，通常是一个登录 shell。

Take a look at your system and see which of these startup files you have. Remember— since most of the filenames listed above start with a period (meaning that they are hidden), you will need to use the “-a” option when using ls.

浏览一下你的系统，看一看系统中有哪些启动文件。记住－因为上面列出的大多数文件名都以圆点开头 （意味着它们是隐藏文件），你需要使用带”-a”选项的 ls 命令。

The ~/.bashrc file is probably the most important startup file from the ordinary user’s point of view, since it is almost always read. Non-login shells read it by default and most startup files for login shells are written in such a way as to read the ~/.bashrc file as well.

在普通用户看来，文件 ~/.bashrc 可能是最重要的启动文件，因为它几乎总是被读取。非登录 shell 默认 会读取它，并且大多数登录 shell 的启动文件会以能读取 ~/.bashrc 文件的方式来书写。

**一个启动文件的内容**

If we take a look inside a typical .bash\_profile (taken from a CentOS 4 system), it looks something like this:

如果我们看一下典型的 .bash\_profile 文件（来自于 CentOS 4 系统），它看起来像这样：

# .bash\_profile

# Get the aliases and functions

if [ -f ~/.bashrc ]; then

. ~/.bashrc

fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH

Lines that begin with a “#” are comments and are not read by the shell. These are there for human readability. The first interesting thing occurs on the fourth line, with the following code:

以”#”开头的行是注释，shell 不会读取它们。它们在那里是为了方便人们阅读。第一件有趣的事情 发生在第四行，伴随着以下代码：

if [ -f ~/.bashrc ]; then

. ~/.bashrc

fi

This is called an if compound command, which we will cover fully when we get to shell scripting in Part 5, but for now we will translate:

这叫做一个 if 复合命令，我们将会在第五部分详细地介绍它，现在我们对它翻译一下：

If the file ~/.bashrc exists, then

read the ~/.bashrc file.

We can see that this bit of code is how a login shell gets the contents of .bashrc. The next thing in our startup file has to do with the PATH variable.

我们可以看到这一小段代码就是一个登录 shell 得到 .bashrc 文件内容的方式。在我们启动文件中， 下一件有趣的事与 PATH 变量有关系。

Ever wonder how the shell knows where to find commands when we enter them on the command line? For example, when we enter ls, the shell does not search the entire computer to find /bin/ls (the full pathname of the ls command), rather, it searches a list of directories that are contained in the PATH variable.

曾经是否感到迷惑 shell 是怎样知道到哪里找到我们在命令行中输入的命令的？例如，当我们输入 ls 后， shell 不会查找整个计算机系统，来找到 /bin/ls（ls 命令的绝对路径名），而是，它查找一个目录列表， 这些目录包含在 PATH 变量中。

The PATH variable is often (but not always, depending on the distribution) set by the /etc/profile startup file and with this code:

PATH 变量经常（但不总是，依赖于发行版）在 /etc/profile 启动文件中设置，通过这些代码：

PATH=$PATH:$HOME/bin

PATH is modified to add the directory $HOME/bin to the end of the list. This is an example of parameter expansion, which we touched on in Chapter 8. To demonstrate how this works, try the following:

修改 PATH 变量，添加目录 $HOME/bin 到目录列表的末尾。这是一个参数展开的实例， 参数展开我们在第八章中提到过。为了说明这是怎样工作的，试试下面的例子：

[me@linuxbox ~]$ foo="This is some"

[me@linuxbox ~]$ echo $foo

This is some

[me@linuxbox ~]$ foo="$foo text."

[me@linuxbox ~]$ echo $foo

This is some text.

Using this technique, we can append text to the end of a variable’s contents. By adding the string $HOME/bin to the end of the PATH variable’s contents, the directory $HOME/bin is added to the list of directories searched when a command is entered. This means that when we want to create a directory within our home directory for storing our own private programs, the shell is ready to accommodate us. All we have to do is call it bin, and we’re ready to go.

使用这种技巧，我们可以把文本附加到一个变量值的末尾。通过添加字符串 $HOME/bin 到 PATH 变量值 的末尾，则目录 $HOME/bin 就添加到了命令搜索目录列表中。这意味着当我们想要在自己的家目录下， 创建一个目录来存储我们自己的私人程序时，shell 已经给我们准备好了。我们所要做的事就是 把创建的目录叫做 bin，赶快行动吧。

Note: Many distributions provide this PATH setting by default. Some Debian based distributions, such as Ubuntu, test for the existence of the ~/bin directory at login, and dynamically add it to the PATH variable if the directory is found.

注意：很多发行版默认地提供了这个 PATH 设置。一些基于 Debian 的发行版，例如 Ubuntu，在登录 的时候，会检测目录 ~/bin 是否存在，若找到目录则把它动态地加到 PATH 变量中。

Lastly, we have:

最后，有下面一行代码：

export PATH

The export command tells the shell to make the contents of PATH available to child processes of this shell.

这个 export 命令告诉 shell 让这个 shell 的子进程可以使用 PATH 变量的内容。

**修改 shell 环境**

Since we know where the startup files are and what they contain, we can modify them to customize our environment.

既然我们知道了启动文件所在的位置和它们所包含的内容，我们就可以修改它们来定制自己的 shell 环境。

**我们应该修改哪个文件？**

As a general rule, to add directories to your PATH, or define additional environment variables, place those changes in .bash\_profile (or equivalent, according to your distribution. For example, Ubuntu uses .profile.) For everything else, place the changes in .bashrc. Unless you are the system administrator and need to change the defaults for all users of the system, restrict your modifications to the files in your home directory. It is certainly possible to change the files in /etc such as profile, and in many cases it would be sensible to do so, but for now, let’s play it safe.

按照通常的规则，添加目录到你的 PATH 变量或者是定义额外的环境变量，要把这些更改放置到 .bash\_profile 文件中（或者其替代文件中，根据不同的发行版。例如，Ubuntu 使用 .profile 文件）。 对于其它的更改，要放到 .bashrc 文件中。除非你是系统管理员，需要为系统中的所有用户修改 默认设置，那么则限定你只能对自己家目录下的文件进行修改。当然，有可能会更改 /etc 目录中的 文件，比如说 profile 文件，而且在许多情况下，修改这些文件也是明智的，但是现在，我们要 安全起见。

**文本编辑器**

To edit (i.e., modify) the shell’s startup files, as well as most of the other configuration files on the system, we use a program called a text editor. A text editor is a program that is, in some ways, like a word processor in that it allows you to edit the words on the screen with a moving cursor. It differs from a word processor by only supporting pure text, and often contains features designed for writing programs. Text editors are the central tool used by software developers to write code, and by system administrators to manage the configuration files that control the system.

为了编辑（例如，修改）shell 的启动文件，还有系统中大多数其它配置文件，我们使用一个叫做文本 编辑器的程序。文件编辑器是一个，在某些方面，类似于文字处理器的程序，比如说随着鼠标的移动， 它允许你在屏幕上编辑文字。只有一点，文本编辑器不同于文字处理器，就是它只能支持纯文本，并且 经常包含为便于写程序而设计的特性。文本编辑器是软件开发人员用来写代码，和系统管理原员用来管理 系统配置文件的重要工具。

There are a lot of different text editors available for Linux; your system probably has several installed. Why so many different ones? Probably because programmers like writing them, and since programmers use them extensively, they write editors to express their own desires as to how they should work.

Linux 系统有许多不同类型的文本编辑器可用；你的系统中可能已经安装了几个。为什么会有这么 多种呢？可能因为程序员喜欢编写它们，又因为程序员们会频繁地使用它们，所以程序员编写编辑器让 它们按照程序员自己的愿望工作。

Text editors fall into two basic categories: graphical and text based. GNOME and KDE both include some popular graphical editors. GNOME ships with an editor called gedit, which is usually called “Text Editor” in the GNOME menu. KDE usually ships with three which are (in order of increasing complexity) kedit, kwrite, and kate.

文本编辑器分为两种基本类型：图形化的和基于文本的编辑器。GNOME 和 KDE 两者都包含一些流行的 图形编辑器。GNOME 自带了一个叫做 gedit 的编辑器，这个编辑器通常在 GNOME 菜单中称为”文本编辑器”。 KDE 通常自带了三种编辑器，分别是（按照复杂度递增的顺序排列）kedit，kwrite，kate。

There are many text-based editors. The popular ones you will encounter are nano, vi, and emacs. The nano editor is a simple, easy-to-use editor designed as a replacement for the pico editor supplied with the PINE email suite. The vi editor (on most Linux systems replaced by a program named vim, which is short for “Vi IMproved”) is the traditional editor for Unix-like systems. It will be the subject of our next chapter. The emacs editor was originally written by Richard Stallman. It is a gigantic, all-purpose, does-everything programming environment. While readily available, it is seldom installed on most Linux systems by default.

有许多基于文本的编辑器。你将会遇到一些流行的编辑器，它们是 nano，vi，和 emacs。这个 nano 编辑器 是一个简单的，容易使用的编辑器，它是 pico 编辑器的替代物，pico 编辑器由 PINE 邮件套件提供。vi 编辑器 （在大多数 Linux 系统中被 vim 替代，vim 是 “Vi IMproved”的简写）是类 Unix 操作系统的传统编辑器。 vim 是我们下一章节的讨论对象。emacs 编辑器最初由 Richard Stallman 写成。emacs 是一个庞大的，多用途的， 可做任何事情的编程环境。虽然 emacs 很容易获取，但是大多数 Linux 系统很少默认安装它。

**使用文本编辑器**

All text editors can be invoked from the command line by typing the name of the editor followed by the name of the file you want to edit. If the file does not already exist, the editor will assume that you want to create a new file. Here is an example using gedit:

所有的文本编辑器都可以通过在命令行中输入编辑器的名字，加上你所想要编辑的文件来唤醒。如果所 输入的文件名不存在，编辑器则会假定你想要创建一个新文件。下面是一个使用 gedit 的例子：

[me@linuxbox ~]$ gedit some\_file

This command will start the gedit text editor and load the file named “some\_file”, if it exists.

这条命令将会启动 gedit 文本编辑器，同时加载名为 “some\_file” 的文件，如果这个文件存在的话。

All graphical text editors are pretty self-explanatory, so we won’t cover them here. Instead, we will concentrate on our first text-based text editor, nano. Let’s fire up nano and edit the .bashrc file. But before we do that, let’s practice some “safe computing.” Whenever we edit an important configuration file, it is always a good idea to create a backup copy of the file first. This protects us in case we mess the file up while editing. To create a backup of the .bashrc file, do this:

所有的图形文本编辑器都相当不言自明的，所以我们在这里不会介绍它们。反之，我们将集中精力在 我们第一个基于文本的文本编辑器，nano。让我们启动 nano，并且编辑文件 .bashrc。但是在我们这样 做之前，先练习一些”安全计算”。当我们编辑一个重要的配置文件时，首先创建一个这个文件的备份 总是一个不错的主意。这样能避免我们在编辑文件时弄乱文件。创建文件 .bashrc 的备份文件，这样做：

[me@linuxbox ~]$ cp .bashrc .bashrc.bak

It doesn’t matter what you call the backup file, just pick an understandable name. The extensions “.bak”, “.sav”, “.old”, and “.orig” are all popular ways of indicating a backup file. Oh, and remember that cp will overwrite existing files silently.

备份文件的名字无关紧要，只要选择一个容易理解的文件名。扩展名 “.bak”，”.sav”， “.old”，和 “.orig” 都是用来指示备份文件的流行方法。哦，记住 cp 命令会默默地重写存在的文件。

Now that we have a backup file, we’ll start the editor:

现在我们有了一个备份文件，我们启动 nano 编辑器吧：

[me@linuxbox ~]$ nano .bashrc

Once nano starts, we’ll get a screen like this:

一旦 nano 编辑器启动后，我们将会得到一个像下面一样的屏幕：

GNU nano 2.0.3

....

Note: If your system does not have nano installed, you may use a graphical editor instead.

注意：如果你的系统中没有安装 nano 编辑器，你可以用一个图形化的编辑器代替。

The screen consists of a header at the top, the text of the file being edited in the middle and a menu of commands at the bottom. Since nano was designed to replace the text editor supplied with an email client, it is rather short on editing features. The first command you should learn in any text editor is how to exit the program. In the case of nano, you type Ctrl-x to exit. This is indicated in the menu at the bottom of the screen. The notation “^X” means Ctrl-x. This is a common notation for control characters used by many programs.

这个屏幕由上面的标头，中间正在编辑的文件文本和下面的命令菜单组成。因为设计 nano 是为了 代替由电子邮件客户端提供的编辑器的，所以它相当缺乏编辑特性。在任一款编辑器中，你应该 学习的第一个命令是怎样退出程序。以 nano 为例，你输入 Ctrl-x 来退出 nano。在屏幕底层的菜单中 说明了这个命令。”^X” 表示法意思是 Ctrl-x。这是控制字符的常见表示法，许多程序都使用它。

The second command we need to know is how to save our work. With nano it’s Ctrl- o. With this knowledge under our belts, we’re ready to do some editing. Using the down arrow key and / or the PageDown key, move the cursor to the end of the file, then add the following lines to the .bashrc file:

第二个我们需要知道的命令是怎样保存我们的劳动成果。对于 nano 来说是 Ctrl-o。尽然我们 已经获得了这些知识，接下来我们准备做些编辑工作。使用下箭头按键和 / 或下翻页按键，移动 鼠标到文件的最后一行，然后添加以下几行到文件 .bashrc 中：

umask 0002

export HISTCONTROL=ignoredups

export HISTSIZE=1000

alias l.='ls -d .\* --color=auto'

alias ll='ls -l --color=auto'

Note: Your distribution may already include some of these, but duplicates won’t hurt anything.

注意：你的发行版可能已经包含其中的一些行，但是复制没有任何伤害。

Here is the meaning of our additions:

下表是所添加行的意义：

|  |  |
| --- | --- |
| *Table 12-4:* | |
| Line | Meaning |
| umask 0002 | Sets the umask to solve the problem with shared directories |
| export HISTCONTROL=ignoredups | Causes the shell's history recording feature to ignore a command if the same command was just recorded. |
| export HISTSIZE=1000 | Increases the size of the command history from the default of 500 lines to 1000 lines. |
| alias l.='ls -d .\* --color=auto' | Creates a new command called “l.” which displays all directory entries that begin with a dot. |
| alias ll='ls -l --color=auto' | Creates a new command called “ll” which displays a long format directory listing. |

|  |  |
| --- | --- |
| *表12-4:* | |
| 文本行 | 含义 |
| umask 0002 | 设置掩码来解决共享目录的问题。 |
| export HISTCONTROL=ignoredups | 使得 shell 的历史记录功能忽略一个命令，如果相同的命令已被记录。 |
| export HISTSIZE=1000 | 增加命令历史的大小，从默认的 500 行扩大到 1000 行。 |
| alias l.='ls -d .\* --color=auto' | 创建一个新命令，叫做'l.'，这个命令会显示所有以点开头的目录项。 |
| alias ll='ls -l --color=auto' | 创建一个叫做'll'的命令，这个命令会显示长格式目录列表。 |

As we can see, many of our additions are not intuitively obvious, so it would be a good idea to add some comments to our .bashrc file to help explain things to the humans. Using the editor, change our additions to look like this:

正如我们所看到的，我们的许多附加物意思直觉上并不是明显的，所以添加注释到我们的文件 .bashrc 中是 一个好主意，可以帮助人们理解。使用编辑器，更改我们的附加物，让它们看起来像这样：

# Change umask to make directory sharing easier

umask 0002

# Ignore duplicates in command history and increase

# history size to 1000 lines

export HISTCONTROL=ignoredups

export HISTSIZE=1000

# Add some helpful aliases

alias l.='ls -d .\* --color=auto'

alias ll='ls -l --color=auto'

Ah, much better! With our changes complete, type Ctrl-o to save our modified .bashrc file, and Ctrl-x to exit nano.

啊，看起来好多了! 当我们完成修改后，输入 Ctrl-o 来保存我们修改的 .bashrc 文件，输入 Ctrl-x 退出 nano。

Why Comments Are Important

**为什么注释很重要？**

Whenever you modify configuration files it’s a good idea to add some comments to document your changes. Sure, you will remember what you changed tomorrow, but what about six months from now? Do yourself a favor and add some comments. While you’re at it, it’s not a bad idea to keep a log of what changes you make.

不管什么时候你修改配置文件时，给你所做的更改加上注释都是一个好主意。的确，明天你会 记得你修改了的内容，但是六个月之后会怎样呢？帮自己一个忙，加上一些注释吧。当你意识 到这一点后，对你所做的修改做个日志是个不错的主意。

Shell scripts and bash startup files use a “#” symbol to begin a comment. Other configuration files may use other symbols. Most configuration files will have comments. Use them as a guide.

Shell 脚本和 bash 启动文件都使用 “#” 符号来开始注释。其它配置文件可能使用其它的符号。 大多数配置文件都有注释。把它们作为指南。

You will often see lines in configuration files that are commented out to prevent them from being used by the affected program. This is done to give the reader suggestions for possible configuration choices or examples of correct configuration syntax. For example, the .bashrc file of Ubuntu 8.04 contains these lines:

你会经常看到配置文件中的一些行被注释掉，以此防止它们被受影响的程序使用。这样做 是为了给读者在可能的配置选项方面一些建议，或者给出正确的配置语法实例。例如，Ubuntu 8.04 中的 .bashrc 文件包含这些行：

# some more ls aliases

#alias ll='ls -l'

#alias la='ls -A'

#alias l='ls -CF'

The last three lines are valid alias definitions that have been commented out. If you remove the leading “#” symbols from these three lines, a technique called uncommenting, you will activate the aliases. Conversely, if you add a “#” symbol to the beginning of a line, you can deactivate a configuration line while preserving the information it contains.

最后三行是有效的被注释掉的别名定义。如果你删除这三行开头的 “#” 符号，此技术程称为 uncommenting (不注释)，这样你就会激活这些别名。相反地，如果你在一行的开头加上 “#” 符号， 你可以注销掉这一行，但会保留它所包含的信息。

**激活我们的修改**

The changes we have made to our .bashrc will not take affect until we close our terminal session and start a new one, since the .bashrc file is only read at the beginning of a session. However, we can force bash to re-read the modified .bashrc file with the following command:

我们对于文件 .bashrc 的修改不会生效，直到我们关闭终端会话，再重新启动一个新的会话， 因为 .bashrc 文件只是在刚开始启动终端会话时读取。然而，我们可以强迫 bash 重新读取修改过的 .bashrc 文件，使用下面的命令：

[me@linuxbox ~]$ source .bashrc

After doing this, we should be able to see the effect of our changes. Try out one of the new aliases:

运行上面命令之后，我们就应该能够看到所做修改的效果了。试试其中一个新的别名：

[me@linuxbox ~]$ ll

**总结**

In this chapter we learned an essential skill—editing configuration files with a text editor. Moving forward, as we read man pages for commands, take note of the environment variables that commands support. There may be a gem or two. In later chapters, we will learn about shell functions, a powerful feature that you can also include in the bash startup files to add to your arsenal of custom commands.

在这一章中，我们学到了用文本编辑器来编辑配置文件的必要技巧。随着继续学习，当我们 读到命令的手册页时，记录下命令所支持的环境变量。可能会有一个或两个宝贝。在随后的章节 里面，我们将会学习 shell 函数，一个很强大的特性，你可以把它包含在 bash 启动文件里面，以此 来添加你自定制的命令宝库。