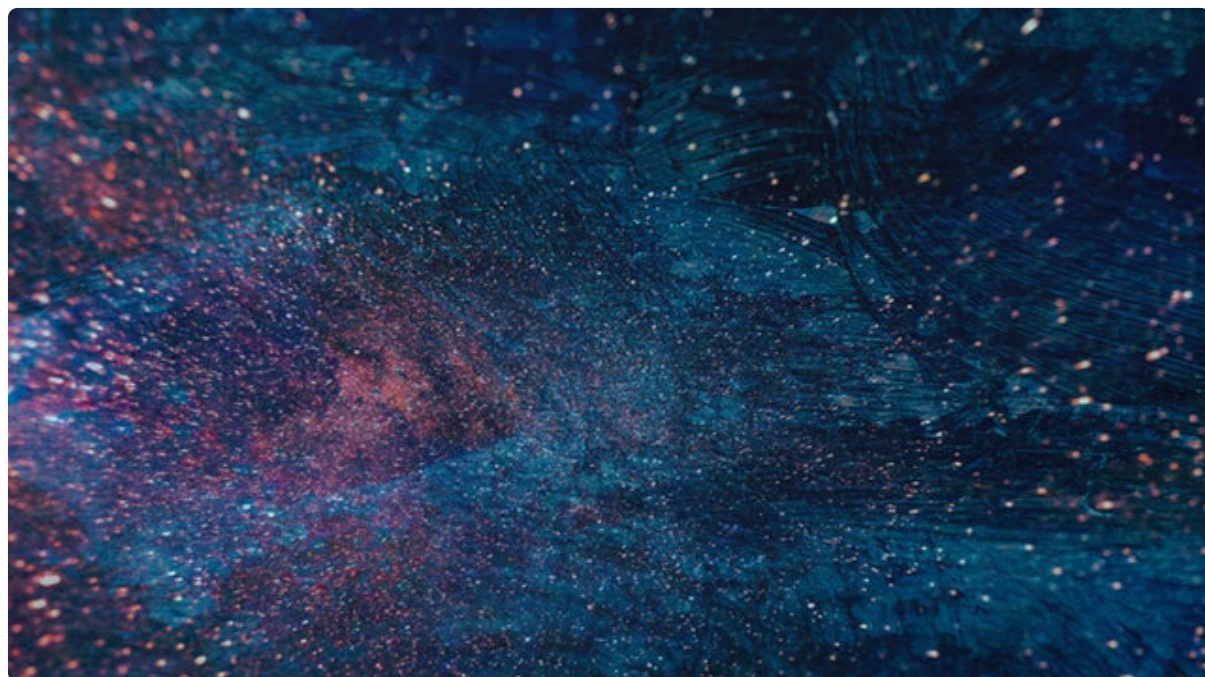


41 一入Shell深似海，酷炫外壳惹人爱

更新时间：2019-08-08 14:40:31



“

没有智慧的头脑，就象没有蜡烛的灯笼。

——列夫·托尔斯泰

”

内容简介

1. 前言
2. Shell 是什么？
3. 我们的第一个 Shell 脚本
4. 运行 Shell 脚本
5. 总结

1. 前言

上一课是 [带你玩转Linux和Shell编程 | 第五部分第二课：Vim的标准和高级操作，配置Vim](#)。

现在，我们已经学习了 Vim 这样强大的文本编辑器。相信我，Vim 对我们之后的课程会非常有用。

这一课我们可以进入第五部分的重心了：**Shell 编程**。

什么是 Shell 呢？

首先，shell 是英语“壳，外壳”的意思。你可以把它想象成嵌入在 Linux 这样的操作系统中的一个“微型编程语言”。

Shell 不像 C 语言，C++，Java 等编程语言那么完整，但是 Shell 这门语言可以帮我们完成很多自动化任务，例如：保存数据，监测系统的负载等等。

我们当然也可以写一个 C 语言的程序来完成以上提到的任务。但是 Shell 相比 C 语言的优势在于它是完全嵌入在 Linux 中的：不需要安装，不需要编译，而且我们不需要学习太多新的东西。

实际上，我们到目前为止在 Linux 中用的那些命令，之后我们也可以用在 Shell 语言中。例如：

```
ls
cd
grep
```

等等。

既然我们第五部分要聊很多 Shell 的事，那么 Shell 到底是什么呢？

我们首先要回答这个问题，然后来写我们的第一个 Shell 脚本。接下来的几课我们会深入 Shell 编程的学习。

脚本（Script）是批处理文件的延伸，是一种纯文本保存的程序。一般来说计算机脚本程序是确定的一系列控制计算机进行运算操作动作的组合，在其中可以实现一定的逻辑分支等。

2. Shell 是什么？

从这个系列课程的一开始，我们就把 Linux 中的两个不同的环境分开来看：

- 终端命令行环境
- 图形界面环境

在大多数的时候，我们使用的是图形界面环境，因为更加直观。不同的图形界面环境的差异总是那么容易辨认，例如菜单项不一样、图标不一样、配色不一样等等。

然而，在终端命令行环境中，我们可以实现很多在图形界面环境中不能完成的复杂任务。

我们之前的课程中说到，Linux 有不少图形界面环境：GNOME、Unity、KDE、XFCE 等等，但是终端命令行环境“貌似”长得一样，只有一种。

其实这说法不准确。终端命令行环境也有好多种，对应的就是不同的 Shell。

不同终端命令行之间的区别不像图形界面那么明显，因为终端命令行一般都是黑底白字。但是根据 Shell 不同，命令行所能提供的功能也不同。

“因此，我们可以把不同的终端命令行环境称为不同的 Shell 咯？”
“是这样的。”

以下列出几种主流的 Shell：

Sh：Bourne Shell 的缩写。可以说是目前所有 Shell 的祖先。

Bash：Bourne Again Shell 的缩写，可以看到比 Bourne Shell 多了一个 again。again 在英语中是“又，再，此外”的意思，说明 Bash 是 Sh 的一个进阶版本，比 Sh 更优秀。Bash 是目前大多数 Linux 发行版和苹果的 macOS 操作系统的默认 Shell。

Ksh：Korn Shell 的缩写。一般在收费的 Unix 版本上比较多见，但也有免费版本的。

Csh : C Shell 的缩写。此 Shell 的语法有点类似 C 语言。

Tcsh : Tenex C Shell 的缩写。Csh 的优化版本。

Zsh : Z Shell 的缩写。比较新近的一个 Shell，集 Bash，Ksh 和 Tcsh 各家之大成。我非常喜欢 Zsh。Github 上有一个 Zsh 的轻松配置程序叫作 [oh-my-zsh](#)，你值得拥有。

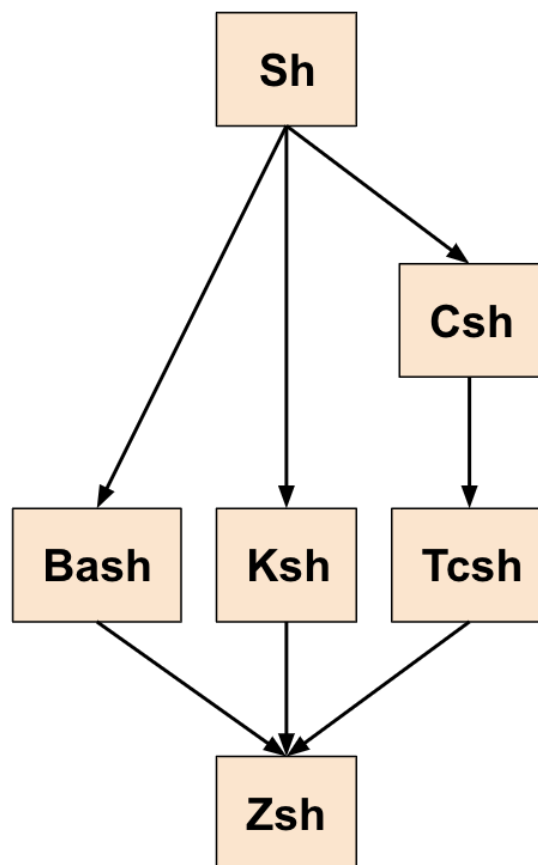
当然了，还有不少其它的 Shell，但我们列出了最主要的一些。

关于这些品类繁多的 Shell 我们需要知道些什么呢？

首先，Sh（Bourne Shell）是所有这些 Shell 的祖先。这是最古老的 Shell，被安装在几乎所有发源于 Unix 的操作系统上。比之于它的后代们，这位“老者”显得有点“技艺不足”。

Bash（Bourne Again Shell）是非常有名的 Shell，是大多数 Linux 发行版的默认 Shell，也是苹果的 macOS 操作系统的默认 Shell。很有可能你目前在你的 Linux 中使用的 Shell 就是 Bash。

我们可以用下图来展示各个 Shell 的演化关系：



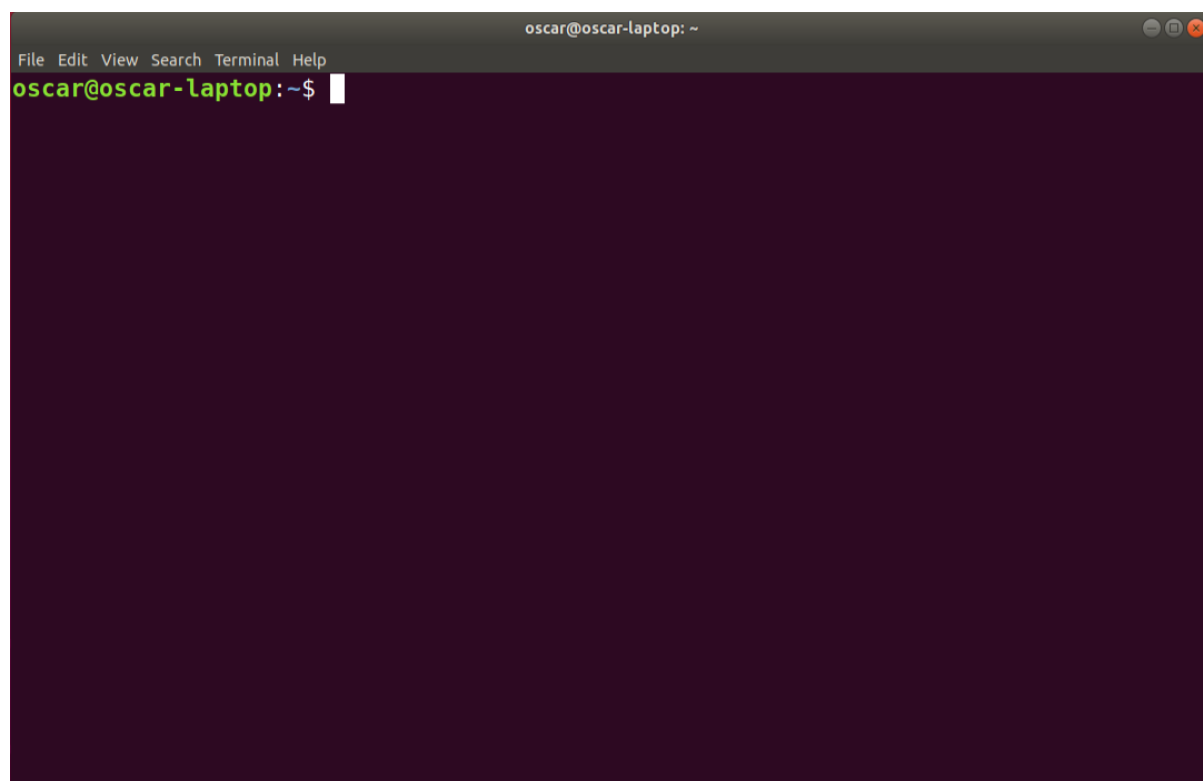
既然 Bash 已经是大多数 Linux 发行版的默认 Shell 了，那么“老古董” Sh 还有什么用呢？

实际上，Sh 始终比 Bash 使用面更广。在本课程的第一课里，我们就介绍了 Linux 是有点模仿 Unix 而创建的。几乎所有源自于 Unix 的操作系统（Linux 也算）都有 Sh，但不是每一个都有 Bash，比如 AIX 和 Solaris 这样的源自于 Unix 的收费操作系统就着用其它的 Shell。

Shell 可以做什么呢？

Shell 是管理命令行的程序。

因此，其实是 Shell 这个程序在等待你输入那些命令。如下图：

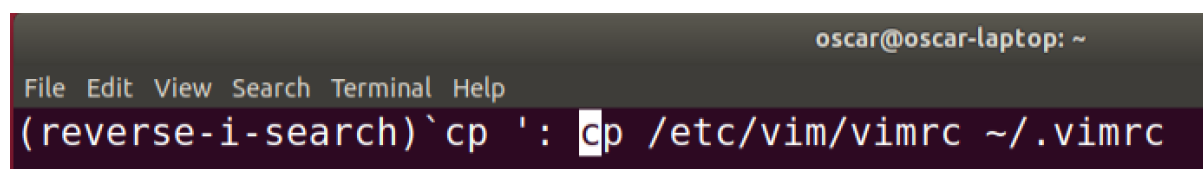
A terminal window titled 'oscar@oscar-laptop: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The prompt 'oscar@oscar-laptop:~\$' is shown with a cursor.

可以看到命令行提示符之后，一个光标一直在一闪一闪的，你就可以输入命令了。

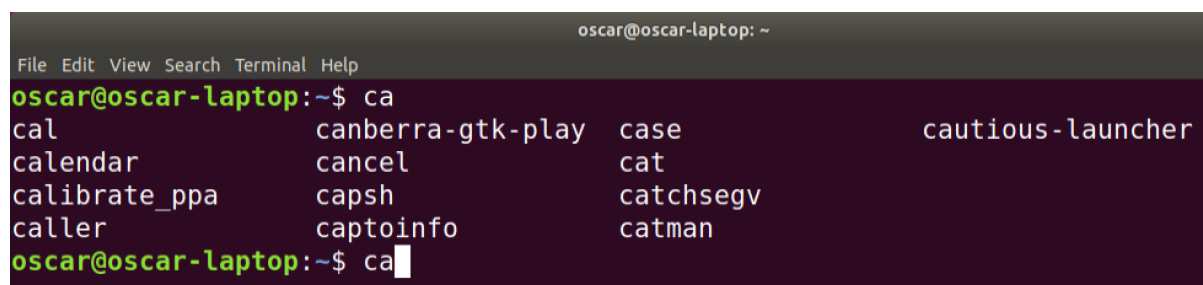
Shell 还可以帮你做很多事情，例如：

记住你之前在终端里输入过的命令：用键盘上的向上键可以回退到你之前执行过的命令。

用组合键 Ctrl + R 在终端的历史记录中搜索执行过的命令：

A terminal window showing a search for '(reverse-i-search)`cp`': cp /etc/vim/vimrc ~/.vimrc. The prompt is 'oscar@oscar-laptop: ~'.

- 用 Tab 键自动补全我们要输入的命令：

A terminal window showing command completion for 'ca'. The prompt is 'oscar@oscar-laptop:~\$'. The completion list includes: cal, calendar, calibrate_ppa, caller, canberra-gtk-play, cancel, capsh, captinfo, case, cat, catchsegv, catman, and cautious-launcher.

上图中，我们输入 `ca`，然后按一下 Tab 键，Shell 就会提示我们可用的候选项。用 Tab 键也可以自动补全文件的路径。

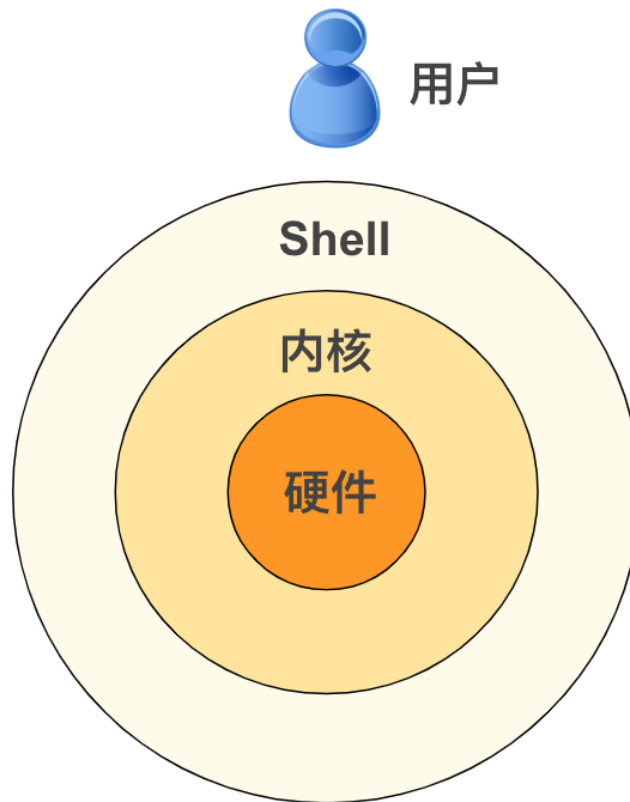
控制进程（把进程放到后台，暂停进程（用之前的课中学过的 Ctrl + z 就可以做到）等等）。

重定向命令（用到 <, >, |, 等符号）。

定义别名：例如可以将很长的一串命令定义为很短的一个别名（alias）。例如 `ll` 可以被定义为 `ls -al` 的别名。

简而言之，Shell 提供了所有可以让你运行命令的基础功能。

可以用下图来简单地表示用户、Shell、操作系统内核和硬件的关系：



可以看到，Shell 就像用户和操作系统之间的一个中介或桥梁一样，这也是它的名字（shell）的由来：很像包裹操作系统内核（内核的英语是 kernel）的一个外壳（shell），就像鸡蛋的外壳一样。

还记得在之前的课中，我们曾经修改过一个 `.bashrc` 的文件吗？

这个 `.bashrc` 其实就是 Bash 这个 Shell 的配置文件。每个 Linux 用户都可以自定义自己的 `.bashrc` 来配置 Bash。它可以指定 Bash 的命令提示符样式，定义别名等等。

在使用 Linux 的过程中，我们经常会碰到一些以 rc 结尾的文件，比如 `.bashrc`，`.zshrc`，`.init.rc` 等等。

之前的课里我们已经介绍过 rc 这个名字的起源。一般以 rc 结尾的多为配置文件，里面包含了软件运行前会去读取并运行的那些初始化命令。

上一课我们提到 Vim 时，它的配置文件叫 `.vimrc`，也是类似的。

安装一个新的 Shell

目前，你的 Linux 系统中大概只安装了 Sh 和 Bash 这两个 Shell，是安装 Ubuntu 时预安装的程序。

如果你想要安装另一个 Shell，比如 Ksh，你可以这样安装：

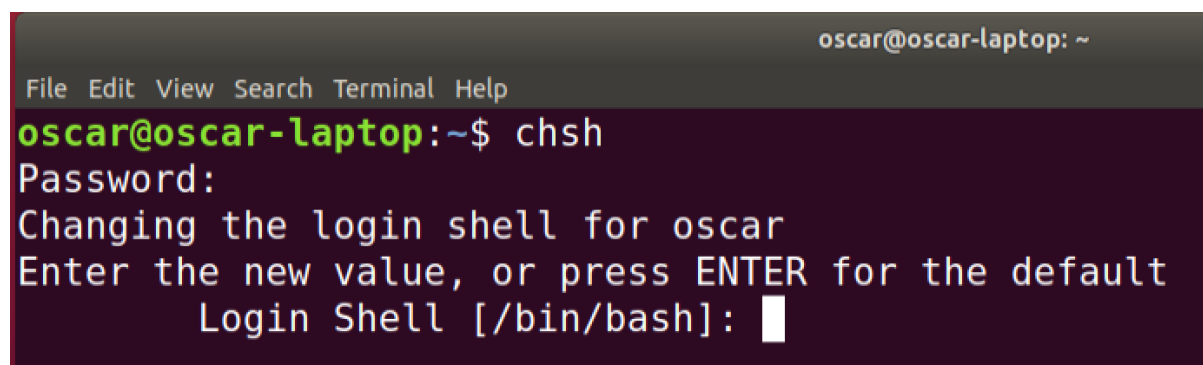
```
sudo apt install ksh
```

一旦你安装好了 Ksh，你还需要将你当前的 Shell（一般来说是 Bash）切换到 Ksh，才能生效。

为了切换 Shell，需要用到以下命令：

```
chsh
```

chsh 是 Change Shell 的缩写，change 是英语“替换，改变”的意思，所以 chsh 用于“切换 Shell”。

A terminal window with a dark background. The title bar shows 'oscar@oscar-laptop: ~'. The menu bar includes 'File Edit View Search Terminal Help'. The prompt is 'oscar@oscar-laptop:~\$'. The user has entered 'chsh'. The prompt changes to 'Password:'. Then, the text 'Changing the login shell for oscar' is displayed, followed by 'Enter the new value, or press ENTER for the default'. The final prompt is 'Login Shell [/bin/bash]:' with a cursor at the end.

运行 chsh 命令后，需要你输入当前用户的密码。

然后它会提示你在冒号后面输入你要切换成的 Shell 的程序路径。你可以输入 /bin/ksh（如果要切换到 Ksh），或 /bin/sh（如果要切换到 Sh）或 /bin/bash（如果要切换到 Bash）等等。

可以看到我们的 Ubuntu 中目前的 Shell 是 Bash。你如果不想修改，那么可以用 Ctrl + c 来退出操作。或者按回车键来保留默认的 Shell。

为什么切换 Shell 对于我们写 Shell 脚本至关重要呢？

因为你的 Shell 脚本需要依赖于某一个 Shell。简单来说，你在使用不同的 Shell（Sh, Bash, Zsh, Ksh等等）时，语法其实是不一样的。

比如，我们可以写 Sh 的脚本。Sh 的脚本基本能运行在大多数系统上，但是 Sh 的语法却并不那么“亲民”。

那么，我们将基于哪种 Shell 来编写我们的 Shell 脚本呢？

在本套课程中，我们学习 **Bash** 这种 Shell，因为：

在大部分 Linux 系统和苹果的 macOS 系统中，Bash 是默认的 Shell，不需要安装。

Bash 的脚本比 Sh 更容易编写。

比起 Ksh 和 Zsh，Bash 更常用。虽然这两个比 Bash 要高级一些。

不过，我推荐你学了 Bash 之后，就转到 Zsh 吧，非常好用，功能极为强大。

3. 我们的第一个 Shell 脚本

介绍了这么多 Shell 的相关知识点，我们就来写一个 Shell 脚本程序咯。这个脚本程序会很简单，但是已足够使我们了解创建 Shell 脚本和运行脚本的基本常识。这对于我们之后的课程是一个基础。

创建脚本文件

我们用 Vim 这个文本编辑器来创建一个 Shell 脚本文件：

```
vim test.sh
```

如果 `test.sh` 这个文件不存在，那么会被创建。

这里，我们给这个 Shell 脚本文件的后缀是 `.sh`。这已经成为一种约定俗成的命名惯例了（`sh` 就是 `shell` 的缩写），其实 Shell 脚本文件和普通的文本文件并没有什么区别。我们给它加上 `.sh` 以强调这是一个 Shell 脚本文件。我们大可以给这个文件起名叫 `test`（不带 `.sh` 后缀）。

但这个课程中，我们还是沿用惯例，把它起名为 `test.sh` 吧。

指定脚本要使用的 Shell

按下回车，上述命令被执行。Vim 就被打开了，在我们眼皮底下的是一个空文件。

在写一个 Shell 脚本时，第一要做的事就是指定要使用哪种 Shell 来“解析/运行”它。因为 `Sh`，`Ksh`，`Bash` 等等 Shell 的语法不尽相同。

因为我们的课程中要使用 `Bash`，因此我们在这个 Shell 脚本的第一行写上：

```
#!/bin/bash
```

上面这句代码中，`/bin/bash` 是 `Bash` 程序在大多数 Linux 系统中的存放路径，而最前面的 `#!` 被称作 `Sha-bang`，或者 `Shebang`。

在计算机科学中，`Shebang`（也称为 `Hashbang`）是一个由井号和叹号构成的字符串 `#!`，其出现在文本文档的第一行的前两个字符。

在文档中存在 `Shebang` 的情况下，类 Unix 操作系统的进程载入器会分析 `Shebang` 后的内容，将这些内容作为解释器指令，并调用该指令，并将载有 `Shebang` 的文档路径作为该解释器的参数。

这一行（`#!/bin/bash`）其实并不是必不可少的，但是它可以保证此脚本会被我们指定的 Shell 执行。

如果你没有写这一行，那么此脚本文件会被用户当前的 Shell 所执行。这就可能产生问题：假如你的脚本是用 `Bash` 的语法来写的，而运行这个脚本的用户的 Shell 是 `Ksh`，那么这个脚本就应该不能正常运行了。

所以记得在写 Shell 脚本文件时，先把 `Shebang` 开头的第一行写好。

运行命令

在以 `Shebang` 开头的第一行之后，我们就可以正式编码了。

原则很简单：只需要写入你想要执行的命令。暂时和之前我们在命令行提示符里写的命令没什么差别。

例如：

- ls: 用于列出目录中的文件。
- cd: 用于切换目录。
- mkdir: 用于创建目录。
- grep: 用于查找字符串。
- sort: 用于排序字符串。

等等。

简而言之，你在之前的课程中所学的命令，现在都可以用了。

好了，我们就以一个非常简单的命令开始吧。

我们输入 ls 这个命令。是的，这个脚本文件的目的暂时就是列出目录中文件。

```
#!/bin/bash  
  
ls
```

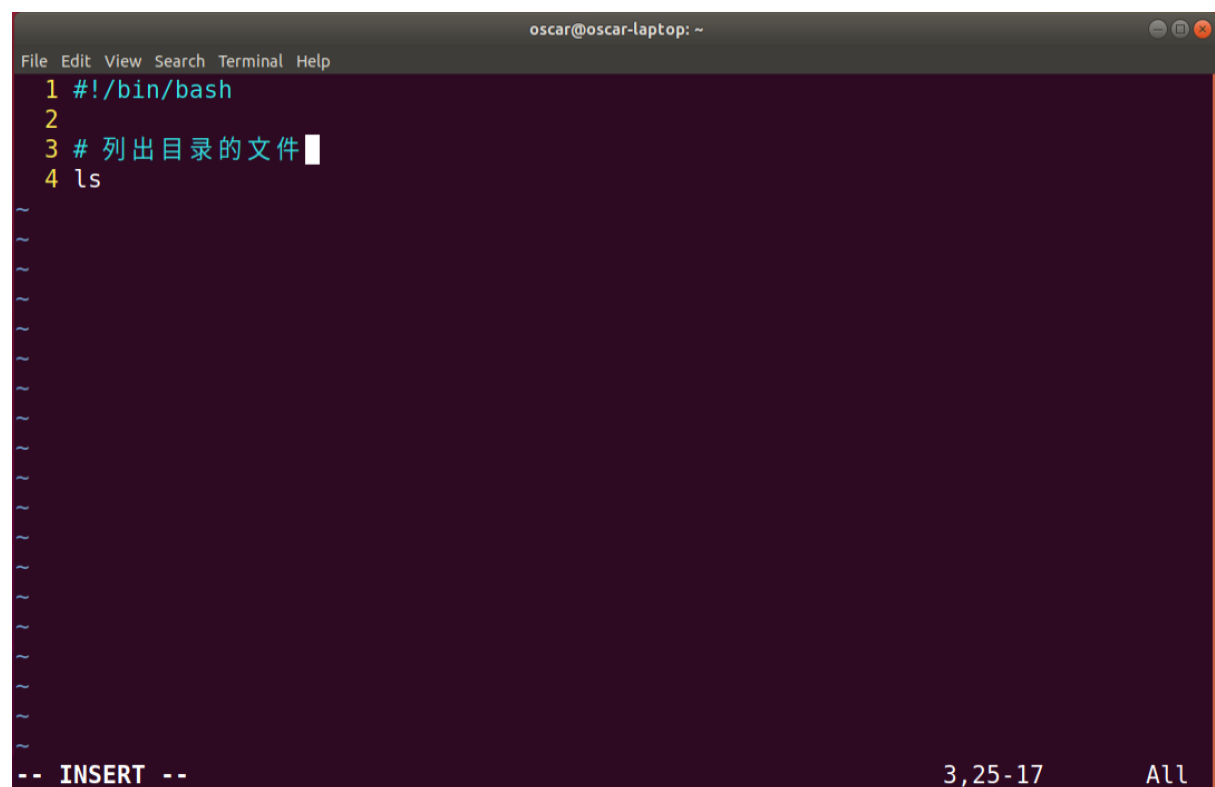
这就是全部了，很简单吧？

注释

就跟其它编程语言的程序一样，我们也可以在 Shell 脚本文件中加入注释。注释是不会被执行的行，但是可以用于解释我们的脚本做了什么。

Shell 脚本的注释以 #（井号）开头。例如：

```
#!/bin/bash  
  
# 列出目录的文件  
ls
```



The screenshot shows a terminal window titled "oscar@oscar-laptop: ~". The terminal has a menu bar with "File", "Edit", "View", "Search", "Terminal", and "Help". The script content is displayed with line numbers: 1 #!/bin/bash, 2, 3 # 列出目录的文件, and 4 ls. The prompt character is "~". At the bottom, it shows "-- INSERT --" and "3,25-17 All".

4. 运行 Shell 脚本

我们刚才写了一个非常简单的 Shell 脚本文件，就几行代码。那么接下来，我们只需要运行它即可。

首先，我们保存刚才的文件并退出 Vim。

是的，只需要用 Esc 键从插入模式进入交互模式，然后按下冒号键进入命令模式，输入 `wq` 或者 `x`，回车即可：

```
:wq
```

或：

```
:x
```

我们就重新找回了我们的命令提示符了。

给脚本文件添加可执行的权限

如果我们在当前目录运行 `ls -l` 命令：

```
ls -l
```

就可以看到刚刚创建的脚本文件了：

```
-rw-r--r--  1 oscar oscar          41 May 31 19:17 test.sh
```

可以看到此文件的权限是：

```
-rw-r--r--
```

如果你还记得我们之前学习的权限的知识，那么应该知道这个文件它没有可执行的权限（因为没有 x）。

因此，我们给它加上可执行权限：

```
chmod +x test.sh
```

再用 `ls -l` 命令来查看，发现已经有了可执行权限了：

```
-rwxr-xr-x  1 oscar oscar          41 May 31 19:17 test.sh
```

运行脚本

我们输入以下命令来运行这个脚本文件：

```
./test.sh
```

回车之后，脚本文件被我们系统中安装的 Bash 运行，显示结果如下：

```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ ./test.sh  
chinese.txt      htop-2.2.0.tar.gz      Public  
compression      Music      README  
date      myFile      redirect  
debian-9.9.0-amd64-netinst.iso  name.txt      renamed_file  
Desktop      new_file      repeat.txt  
Documents      newly_created_file      results.txt  
Downloads      node-v10.15.3-copy.tar.gz  share  
emacs-26.2-copy.tar.gz  node-v10.15.3.tar.gz  snap  
emacs-26.2.tar.gz      nohup.out      sorted_name.txt  
errors.log      number.txt      Templates  
find_log      one      test  
grep_log      one_copy      test.sh  
helloworld.cpp      output_find      unique.txt  
htop-2.2.0      Pictures      Videos  
oscar@oscar-laptop:~$
```

这个脚本做了什么呢？

它仅仅是执行了 `ls` 这个基本的 Linux 命令而已。因为这个脚本文件位于我的家目录（`/home/oscar`），因此，它列出了我的家目录中的所有文件。

看上去，这个脚本还真是“鸡肋”。为什么我们不直接在命令行运行 `ls` 命令，而要大费周章写一个 Shell 脚本呢？

相信我，Shell 脚本肯定是有用的，因为它可以一次性自动化执行很多很多命令，而不需要你一个个输入了。目前我们的脚本中没多少东西，但之后的课程你将见识到 Shell 的强大。

我们可以在刚才的 `test.sh` 这个 Shell 脚本中再加入一些命令，使之能做更多事情。比如，我们添加 `pwd` 命令，让它打印出当前目录的路径：

```
#!/bin/bash  
  
# 打印当前目录  
pwd  
  
# 列出目录的文件  
ls
```

保存文件，并再次运行，可以看到结果：

```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ ./test.sh  
/home/oscar  
chinese.txt      htop-2.2.0.tar.gz      Public  
compression     Music                  README  
date             myFile                 redirect  
debian-9.9.0-amd64-netinst.iso name.txt               renamed_file  
Desktop          new_file               repeat.txt  
Documents        newly_created_file     results.txt  
Downloads        node-v10.15.3-copy.tar.gz share  
emacs-26.2-copy.tar.gz node-v10.15.3.tar.gz  snap  
emacs-26.2.tar.gz  nohup.out              sorted_name.txt  
errors.log        number.txt             Templates  
find_log          one                    test  
grep_log          one_copy               test.sh  
helloworld.cpp    output_find            unique.txt  
htop-2.2.0        Pictures               Videos  
oscar@oscar-laptop:~$
```

这次先是调用 `pwd` 命令打印出了当前目录的路径（`/home/oscar`，我的家目录），接着才运行 `ls` 命令，打印出当前目录中的文件。

以调试模式运行

随着我们渐渐深入 Shell 编程，你也许会写出很长的 Shell 脚本，代码一多很可能就会有 Bug。

`bug` 是英语单词，本意是“臭虫、缺陷、损坏、犯贫、窃听器、小虫”等意思。现在人们将在电脑系统或程序中，隐藏着的一些未被发现的缺陷或问题统称为 `bug`（漏洞）。

因此，我们需要学习如何调试一个脚本程序。用法如下：

```
bash -x test.sh
```

我们直接调用 `Bash` 这个 Shell 程序，并且给它一个参数 `-x`（表示以调试模式运行），后面再跟上要调试运行的脚本文件。

如此一来，Shell 就会把我们的脚本文件运行时的细节打印出来了，在出现错误时可以帮助我们排查问题所在。

```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ bash -x test.sh  
+ pwd  
/home/oscar  
+ ls  
chinese.txt          htop-2.2.0.tar.gz      Public  
compression          Music                   README  
date                 myFile                  redirect  
debian-9.9.0-amd64-netinst.iso  name.txt               renamed_file  
Desktop              new_file               repeat.txt  
Documents            newly_created_file     results.txt  
Downloads            node-v10.15.3-copy.tar.gz  share  
emacs-26.2-copy.tar.gz  node-v10.15.3.tar.gz  snap  
emacs-26.2.tar.gz     nohup.out              sorted_name.txt  
errors.log           number.txt             Templates  
find_log             one                    test  
grep_log             one_copy               test.sh  
helloworld.cpp       output_find            unique.txt  
htop-2.2.0           Pictures               Videos  
oscar@oscar-laptop:~$
```

创建属于自己的命令

目前，我们的 Shell 脚本文件必须要这样运行：

```
./test.sh
```

而且我们必须位于正确的目录中。

那么其它的一些程序（命令其实都是程序），比如 `date`，`pwd`，`ls`，`cat`，`cp` 等等，为什么可以直接从不论哪个目录执行（不需要在前面加上 `./` 这样的路径）呢？

秘密就在于这些程序存放的目录是在 `PATH` 这个环境变量中的。

`PATH` 是英语“小路，路；路线，路程；途径”的意思。`PATH` 是 Linux 的一个系统变量。这个变量包含了你系统里所有可以被直接执行的程序的路径。

如果我们在终端输入：

```
echo $PATH
```

我们就可以看到目前自己系统里的那些“特殊”的目录了。

例如我的情况：

```
oscar@oscar-laptop: ~  
File Edit View Search Terminal Help  
oscar@oscar-laptop:~$ echo $PATH  
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin  
oscar@oscar-laptop:~$
```

每一个路径之间是用冒号（`:`）来分隔的。

因此，只要你把 `test.sh` 这个文件拷贝到上述路径列表的任意一个目录（例如 `/usr/local/bin`，`/usr/bin` 等等）中，你就可以在随便什么目录中运行你的 Shell 脚本了。

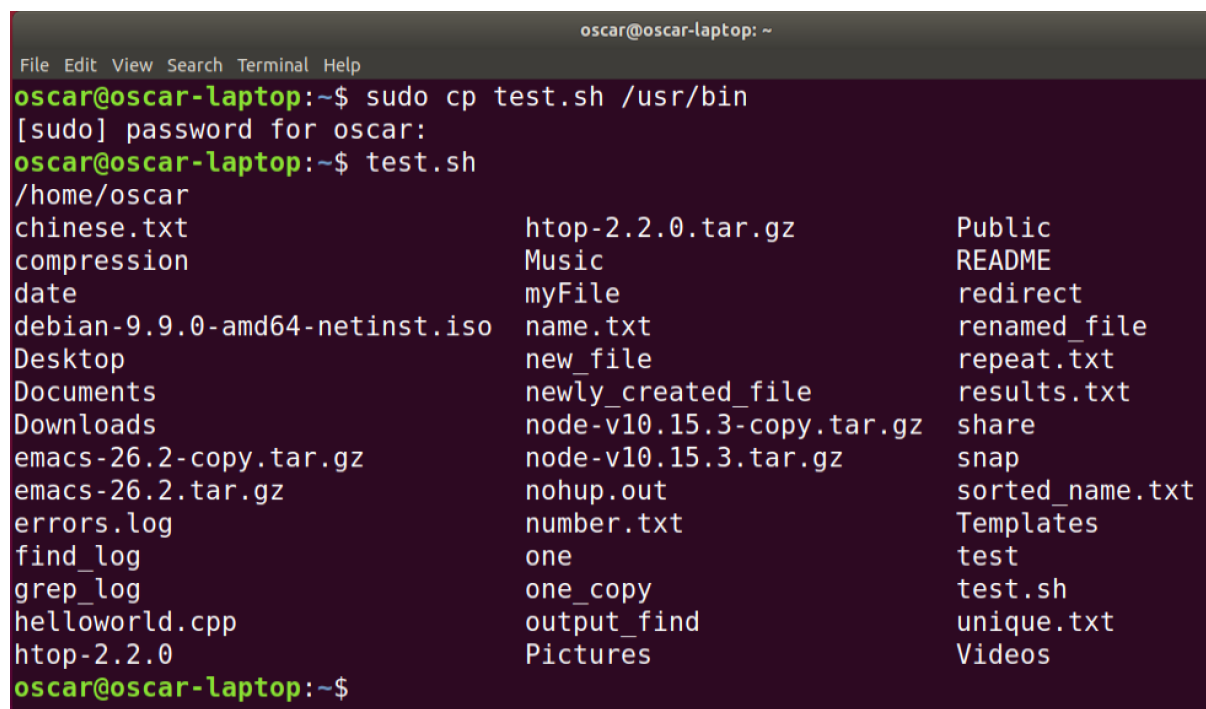
```
test.sh
```

当然了，如果是拷贝到系统目录，那么需要用 `root` 身份哦。

例如，我们可以用：

```
sudo cp test.sh /usr/bin
```

将 `test.sh` 拷贝到 `/usr/bin` 这个目录里，因为这个目录是在 `PATH` 系统变量中的，现在我们可以直接运行 `test.sh` 而不需要用 `test.sh` 啦：

A terminal window titled 'oscar@oscar-laptop: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The user runs 'sudo cp test.sh /usr/bin', enters a password, and then runs 'test.sh'. The output is a three-column list of files and directories: /home/oscar, chinese.txt, compression, date, debian-9.9.0-amd64-netinst.iso, Desktop, Documents, Downloads, emacs-26.2-copy.tar.gz, emacs-26.2.tar.gz, errors.log, find_log, grep_log, helloworld.cpp, htop-2.2.0, htop-2.2.0.tar.gz, Music, myFile, name.txt, new_file, newly_created_file, node-v10.15.3-copy.tar.gz, node-v10.15.3.tar.gz, nohup.out, number.txt, one, one_copy, output_find, Pictures, Public, README, redirect, renamed_file, repeat.txt, results.txt, share, snap, sorted_name.txt, Templates, test, test.sh, unique.txt, and Videos. The prompt returns to 'oscar@oscar-laptop:~\$'.

当然了，最好不要乱拷贝文件到系统路径里，所以我们可以把 `test.sh` 从 `/usr/bin` 这个目录删除：

```
sudo rm /usr/bin/test.sh
```

5. 总结

原先我们以为终端命令行只有一种形式，但其实有不少类型的终端：这就对应了不同的 Shell（外壳程序）。Shell 可以管理命令提示符，还有多种功能，例如命令的历史记录（用 `Ctrl + R` 来查找），命令的自动补全等等；

在 Ubuntu 中，默认的 Shell 是 Bash。但我们也可以安装其它的 Shell，例如 Ksh，Zsh 等等；

我们可以用 Shell 来自动化一系列命令。首先需要创建一个文件，包含要运行的命令的列表，称之为 Shell 脚本。这也称之为 **Shell 编程**；

根据使用的 Shell 种类不同，我们有不同的工具来处理 Shell 脚本。我们在本课中使用 Bash 来演示，因此在 Shell 脚本文件的开头需要写上：`#!/bin/bash`；

在 Shell 脚本文件里，一般来说只需要把我们要执行的命令一行一行地写入文件：

为了运行 Shell 脚本（也就是运行其中包含的那些命令），需要先给脚本文件添加可执行属性（`chmod +x script.sh`），然后这样运行：`./script.sh`。

今天的课就到这里，一起加油吧！

← 40 Vim的标准和高级操作，配置 Vim

42 变量在手，Shell不愁 →