IPython and Shell Commands

IPython 和 Shell命令

> When working interactively with the standard Python interpreter, one of the frustrations is the need to switch between multiple windows to access Python tools and system command-line tools. IPython bridges this gap, and gives you a syntax for executing shell commands directly from within the IPython terminal. The magic happens with the exclamation point: anything appended after `!` on a line will be executed not by the Python kernel, but by the system command-line.

当使用标准的Python解释器时，有一个让人感到沮丧的地方就是你需要在不同的窗口之间进行切换，有时你需要使用Python，有时你又需要使用系统命令行工具。IPython将两者联系起来，它允许你直接在IPython终端中直接运行shell命令。这个魔术使用的是感叹号：任何出现在 `!` 之后的内容将被系统shell执行，而不是Python解释器。

> The following assumes you're on a Unix-like system, such as Linux or Mac OSX. Some of the examples that follow will fail on Windows, which uses a different type of shell by default (though with the 2016 announcement of native Bash shells on Windows, soon this may no longer be an issue!). If you're unfamiliar with shell commands, I'd suggest reviewing the Shell Tutorial put together by the always excellent Software Carpentry Foundation.

本节内容假定你在使用一个类Unix的系统，如Linux或者Mac OS X。下面的一些例子会在Windows下面失效，因为它使用的是一种完全不同的shell（2016年Windows宣布将直接支持原生的Bash，很快这将不成为问题。译者注：目前在windows下使用bash还是会有很多问题，微软的原生实现并不理想）。如果你对于shell命令不熟悉，作者推荐你去Shell教程这本学习一下基础的shell命令。

Quick Introduction to the Shell

Shell快速介绍

> A full intro to using the shell/terminal/command-line is well beyond the scope of this chapter, but for the uninitiated we will offer a quick introduction here. The shell is a way to interact textually with your computer. Ever since the mid 1980s, when Microsoft and Apple introduced the first versions of their now ubiquitous graphical operating systems, most computer users have interacted with their operating system through familiar clicking of menus and drag-and-drop movements. But operating systems existed long before these graphical user interfaces, and were primarily controlled through sequences of text input: at the prompt, the user would type a command, and the computer would do what the user told it to. Those early prompt systems are the precursors of the shells and terminals that most active data scientists still use today.

如何使用shell环境/命令行远远超出了本章的范围，但是对于初学者，作者还是准备了一个简单快速的介绍。从80年代中开始，微软和苹果想用户推出了它们的图形界面，时至今日，图形化操作系统已经是无处不在了。大部分的计算机用户都是使用他们熟悉的菜单点击和拖放操作来使用操作系统。但是实际上操作系统比这些图形用户界面出现早得多，当时都是由用户输入一系列的文本内容对操作系统进行控制：在提示符下，用户输入一个命令，然后计算机会按照用户的指示进行工作。这种早期的提示符界面就是shell和终端的前身，也是直到今天很多数据科学家仍在使用的工具。

> Someone unfamiliar with the shell might ask why you would bother with this, when many results can be accomplished by simply clicking on icons and menus. A shell user might reply with another question: why hunt icons and click menus when you can accomplish things much more easily by typing? While it might sound like a typical tech preference impasse, when learning about basic tasks it quickly becomes clear that the shell offers much more control of advanced tasks, though admittedly the learning curve can intimidate the average computer user.

不熟悉shell的人可能会问，为什么你们要这么麻烦，为什么简单的通过点击图表和菜单就能实现的功能你们要敲命令。熟练使用shell的用户可能会这样回应：为什么通过简单的键盘命令就能完成的工作你们要点击鼠标呢。虽然看起来这是一个典型的技术偏好问题，但是当你需要完成的任务变得复杂的时候，shell确实能够提供更多的控制，哪怕shell的学习曲线会吓跑很多普通的计算机用户。

> As an example, here is a sample of a Unix/OSX shell session where a user explores, creates, and modifies directories and files on their system (`osx:~ $` is the prompt, and everything after the `$` sign is the typed command; text that is preceded by a `#` is meant just as description, rather than something you would actually type in):

作为一个例子，这里有一个用户在Linux/OSX系统上浏览、创建和修改目录以及文件的shell会话（`osx:~ $` 是提示符，所有出现在 `$` 后面的文本都是一条命令；以 `#` 开始的文本是注释作为命令的解释，而不是你需要真正输入的内容）：

```
osx:~ $ echo "hello world"                  # 使用echo打印输出，类似Python中的print
hello world

osx:~ $ pwd                                 # pwd = 打印当前工作目录
/home/jake                                  # 这是我们当前的工作目录

osx:~ $ ls                                  # ls = 列示目录内容
notebooks  projects

osx:~ $ cd projects/                        # cd = 改变目录位置

osx:projects $ pwd
/home/jake/projects

osx:projects $ ls
datasci_book   mpld3    myproject.txt

osx:projects $ mkdir myproject              # mkdir = 创建新目录

osx:projects $ cd myproject/

osx:myproject $ mv ../myproject.txt ./  # mv = 移动文件，这里我们将父目录中的myproject.txt
                                        # 移动到当前工作目录下
osx:myproject $ ls
myproject.txt
```

> Notice that all of this is just a compact way to do familiar operations (navigating a directory structure, creating a directory, moving a file, etc.) by typing commands rather than clicking icons and menus. Note that with just a few commands (`pwd`, `ls`, `cd`, `mkdir`, and `cp`) you can do many of the most common file operations. It's when you go beyond these basics that the shell approach becomes really powerful.

请注意，上面的命令都是使用命令输入完成我们平常使用鼠标点击操作完成的任务（浏览目录结构、创建目录、移动文件等）。只需要少量的命令输入（`pwd`、`ls`、`cd`、`mkdir` 和 `cp`）我们就能完成很多通用的文件操作。当你要深入学习shell之后，你就会发现它们非常强大。

Shell Commands in IPython

IPython 中的 shell 命令

> Any command that works at the command-line can be used in IPython by prefixing it with the `!` character. For example, the `ls`, `pwd`, and `echo` commands can be run as follows:

任何在命令行中可以使用的命令，也都可以在IPython中使用，只需要在前面加上 `!` 号。例如，`ls`、`pwd` 和 `echo` 命令：

```
In [1]: !ls
myproject.txt

In [2]: !pwd
/home/jake/projects/myproject

In [3]: !echo "printing from the shell"
printing from the shell
```

Passing Values to and from the Shell

与 shell 之间传递值

> Shell commands can not only be called from IPython, but can also be made to interact with the IPython namespace. For example, you can save the output of any shell command to a Python list using the assignment operator:

shell命令不但能被IPython环境中调用，还能与IPython的命名空间产生交互。例如，你可以将shell命令的输出保存成一个Python的列表：

```
In [4]: contents = !ls

In [5]: print(contents)
['myproject.txt']

In [6]: directory = !pwd

In [7]: print(directory)
['/Users/jakevdp/notebooks/tmp/myproject']
```

> Note that these results are not returned as lists, but as a special shell return type defined in IPython:

值得注意的是，这些结果并不是返回成为普通的Python列表，而是一个IPython定义的特殊shell返回值类型：

```
In [8]: type(directory)
IPython.utils.text.SList
```

> This looks and acts a lot like a Python list, but has additional functionality, such as the `grep` and `fields` methods and the `s`, `n`, and `p` properties that allow you to search, filter, and display the results in convenient ways. For more information on these, you can use IPython's built-in help features.

它看起来很像一个Python列表，但是还包含额外的功能，比方说 `grep` 和 `fields` 方法，以及 `s`、`n` 和 `p` 属性，让你能够使用简单方式搜索，过滤和显示结果。如果你想获得更多信息，请使用IPython内建的帮助特性来查看。

> Communication in the other direction–passing Python variables into the shell–is possible using the `{varname}` syntax:

反过来，也可以传递Python的变量给shell，通过 `{变量名}` 语法就可以实现：

```
In [9]: message = "hello from Python"

In [10]: !echo {message}
hello from Python
```

> The curly braces contain the variable name, which is replaced by the variable's contents in the shell command.

花括号里面是变量的名称，在执行shell命令的时候将会被变量的值替代。

Shell-Related Magic Commands

Shell 相关魔术命令

> If you play with IPython's shell commands for a while, you might notice that you cannot use `!cd` to navigate the filesystem:

如果你已经在IPython中使用了shell命令一段时间了，你会发现你无法使用 `!cd` 来改变你的工作目录：

```
In [11]: !pwd
/home/jake/projects/myproject

In [12]: !cd ..

In [13]: !pwd
/home/jake/projects/myproject
```

> The reason is that shell commands in the notebook are executed in a temporary subshell. If you'd like to change the working directory in a more enduring way, you can use the `%cd` magic command:

这是因为在notebook里面shell是在一个子shell空间中执行的。如果你需要改变工作目录的话，你可以使用 `%cd` 魔术命令：

```
In [14]: %cd ..
/home/jake/projects
```

> In fact, by default you can even use this without the `%` sign:

事实上，你甚至可以不用 `%` 号：

```
In [15]: cd myproject
/home/jake/projects/myproject
```

> This is known as an `automagic` function, and this behavior can be toggled with the `%automagic` magic function.

这被称为 自动魔术，你可以使用 `%automagic` 来切换它的开关状态。

> Besides `%cd`, other available shell-like magic functions are `%cat`, `%cp`, `%env`, `%ls`, `%man`, `%mkdir`, `%more`, `%mv`, `%pwd`, `%rm`, and `%rmdir`, any of which can be used without the `%` sign if `automagic` is on. This makes it so that you can almost treat the IPython prompt as if it's a normal shell:

除了 `%cd` 之外，其他类似shell命令的魔术命令包括 `%cat`、`%cp`、`%env`、`%ls`、`%man`、`%mkdir`、`%more`、`%mv`、`%pwd`、`%rm` 和 `%rmdir`，这些命令在 `automagic` 开启时都可以不带 `%` 使用。这功能令你可以几乎将IPython shell当成系统的shell来使用了：

```
In [16]: mkdir tmp

In [17]: ls
myproject.txt  tmp/

In [18]: cp myproject.txt tmp/

In [19]: ls tmp
myproject.txt

In [20]: rm -r tmp
```

> This access to the shell from within the same terminal window as your Python session means that there is a lot less switching back and forth between interpreter and shell as you write your Python code.

能够在IPython环境中直接使用shell，意味着你可以不用来回在解释器和shell终端两个窗口之间进行切换，可以提高你写Python代码时候的效率。