ipython常用功能，iPython 安装 比shell好用

概述

iPython 是一个Python 的交互式Shell，比默认的Python Shell 好用得多，功能也更强大。

她支持语法高亮、自动完成、代码调试、对象自省，支持 Bash Shell 命令，内置了许多

很有用的功能和函式等，非常容易使用。

应用

Windows 下的iPython 安装

在Windows 下安装iPython 可分为以下几步：

1. 下载ipython-0.8.4.win32-setup.exe 和pyreadline-1.5-win32-setup.exe。

（1）下载ipython-0.8.4.win32：

http://ipython.scipy.org/dist/ipython-0.8.4.win32-setup.exe

精巧地址：http://bit.ly/YqbkJ

（2）下载pyreadline-1.5-win32：

http://ipython.scipy.org/dist/pyreadline-1.5-win32-setup.exe

精巧地址：http://bit.ly/2JFKDM

安装pyreadline。直接双击安装

ubuntu 下安装

```
sudo apt-get install  'ipython'
```

在交互环境和在Python 默认交互环境中一样，编写代码进行调试、测试等。但比默认

Python 环境好的几点如下所示。

1. Magic。iPython 有一些“magic”关键字：

%Exit, %Pprint, %Quit, %alias, %autocall, %autoindent,
%automagic,
%bookmark, %cd, %color_info, %colors, %config, %dhist, %dirs,
%ed,
%edit, %env, %hist, %logoff, %logon, %logstart, %logstate,
%lsmagic,
%macro, %magic, %p, %page, %pdb, %pdef, %pdoc, %pfile, %pinfo,
%popd,
%profile, %prun, %psource, %pushd, %pwd, %r, %rehash, %rehashx,
%reset,
%run, %runlog, %save, %sc, %sx, %system_verbose, %unalias, %who,
%who_ls, %whos, %xmode
iPython 会检查传给它的命令是否包含magic 关键字。如果命令是一个magic 关键字，
iPython 就自己来处理。如果不是magic 关键字，就交给 Python 去处理。如果
automagic 打开（默认），不需要在magic 关键字前加%符号。相反，如果 automagic
是关闭的，则%是必须的。在命令提示符下输入命令magic 就会显示所有magic 关键
字列表，以及它们简短的用法说明。良好的文档对于一个软件的任何一部分来说都
是重要的，从在线iPython 用户手册到内嵌文档（%magic），iPython 当然不会在这方
面有所缺失。下面介绍些常用的magic 函式，如：
%bg function
把function 放到后台执行，例如：%bg myfunc(x, y, z=1)，之后可以用jobs 将其结
果取回，myvar = jobs.result(5)或myvar = jobs[5].result。另外，
jobs.status()

可以查看现有任务的状态。

%ed 或 %edit

编辑一个文件并执行，如果只编辑不执行，用 ed -x filename 即可。

%env

显示环境变量

%hist 或 %history

显示历史记录

%macro name n1-n2 n3-n4 ... n5 .. n6 ...

创建一个名称为 name 的宏，执行 name 就是执行 n1-n2 n3-n4 ... n5 .. n6 ... 这

些代码。

%pwd

显示当前目录

%pycat filename

用语法高亮显示一个 Python 文件（不用加.py 后缀名）

%save filename n1-n2 n3-n4 ... n5 .. n6 ...

将执行过多代码保存为文件

%time statement

计算一段代码的执行时间

%timeit statement

自动选择重复和循环次数计算一段代码的执行时间，太方便了。

2. iPython 中用！表示执行shell 命令，用$将Python 的变量转化成 Shell 变量。通过

这两个符号，就可以做到和Shell 命令之间的交互，可以非常方便地做许多复杂的

工作。比如可以很方便地创建一组目录：

```
for i in range(10):
s = "dir%s" % i
!mkdir $s
```

不过写法上还是有一些限制，$ 后面只能跟变量名，不能直接写复杂表达式，

$"dir%s"%i 就是错误的写法了，所以要先完全产生 Python 的变量以后再用。
例如：

```
for i in !ls:
print i
```

这样的写法也是错的，可以这样：

```
a = !ls
for i in a:
print i
```

还有一点需要说明，就是执行普通的 Shell 命令中如果有 $ 的话需要用两个$。比
如原来的echo $PATH 现在得写成!echo $$PATH。

3. Tab 自动补全。iPython 一个非常强大的功能是tab 自动补全。标准Python 交互式解
释器也可以tab 自动补全：

```
~$ python
Python 2.5.1 (r251:54863, Mar 7 2008, 04:10:12)
[GCC 4.1.3 20070929 (prerelease) (Ubuntu 4.1.2-16ubuntu2)] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> import rlcompleter, readline
>>> readline.parse_and_bind('tab: complete')
>>> h
hasattr hash help hex
```

标准Python 交互式解释器和iPython 都支持“普通”自动补全和菜单补全。
使用自动补全，要先输入一个匹配模型，然后按Tab 键。如果是“普通”自动补全
模式（默认），按Tab 键后会：
匹配模型按最大匹配展开；
列出所有匹配的结果。
例如：

In [1]: import os

In [2]: os.po

os.popen os.popen2 os.popen3 os.popen4

In [2]: os.popen

输入os.po 然后按Tab 键，os.po 被展开成os.popen（就像在In [2]:提示符显示的那样），

并显示os 所有以po 开头的模块、类和函式，它们是popen、popen2、popen3 和popen4。

而菜单补全稍有不同。关闭默认Tab 补全，使用菜单补全，需修改配置文件$HOME/.ipython/ipythonrc。

注释掉：

readline_parse_and_bind tab: complete

取消注释:

readline_parse_and_bind tab: menu-complete

不同于“普通”自动补全的显示，当前命令所有匹配列表的菜单补全会随着每按一

次Tab 键而循环显示匹配列表中的项目。例如：

In [1]: import os

In [2]: os.po

结果是：

In [3]: os.popen

接下来每次按Tab 键就会循环显示匹配列表中的其他项目：popen2、popen3、popen4，

最后回到po。菜单补全模式下查看所有匹配列表的快捷键是Ctrl+L。

5. 历史。当在iPython shell 下交互地输入了大量命令、语句等，就像这样：

In [1]: a = 1

In [2]: b = 2

In [3]: c = 3

In [4]: d = {}

```
In [5]: e = []
In [6]: for i in range(20):
...: e.append(i)
...: d[i] = b
...:
```

可以输入命令“hist”快速查看那些已输入的历史记录：

```
In [7]: hist
1: a = 1
2: b =2
3: c = 3
4: d = {}
5: e = []
6:
for i in range(20):
e.append(i)
d[i] = b
7: _ip.magic("hist ")
```

要去掉历史记录中的序号（这里是1 至7），可使用命令“hist -n”：

```
In [8]: hist -n
a = 1
b =2
c = 3
d = {}
e = []
for i in range(20):
e.append(i)
d[i] = b
_ip.magic("hist ")
_ip.magic("hist -n")
```

这样就可方便地将代码复制到一个文本编辑器中。要在历史记录中搜索，可以先输
入一个匹配模型，然后按Ctrl+P 键。找到一个匹配后，继续按Ctrl+P 键就会向后搜
索再上一个匹配，按Ctrl+N 键则是向前搜索最近的匹配。
6. 编辑。如果想在Python 提示符下试验一个想法，经常要通过编辑器修改源代码（甚
至是反复修改）。**在iPython 下输入edit 就会根据环境变量$ EDITOR 调用相应的编**
辑器。如果$EDITOR 为空，则会调用vi（Unix）或记事本（Windows）。要回到iPython
提示符，直接退出编辑器即可。如果是保存并退出编辑器，输入编辑器的代码会在
当前名字空间下被自动执行。如果不想这样，可使用edit+X。如果要再次编辑上次
最后编辑的代码，使用edit+P。在上一个特性里，提到使用hist-n 可以很容易地将
代码拷贝到编辑器。一个更简单的方法是edit 加Python 列表的切片（slice）语法。
假定hist 输出如下：

```
In [29]: hist
1 : a = 1
2 : b = 2
3 : c = 3
4 : d = {}
5 : e = []
6 :
for i in range(20):
e.append(i)
d[i] = b
```

7 : %hist
现在要将第4、5、6 句代码导出到编辑器，只要输入：
edit 4:7
7. Debugger 接口。iPython 的另一特性是它与Python debugger 的接口。在iPython Shell
下输入magic 关键字pdb 就会在产生一个异常时开关自动debugging 功能。在pdb
自动呼叫启用的情况下，当Python 遇到一个未处理的异常时Python debugger 就会自
动启动。debugger 中的当前行就是异常发生的那一行。iPython 的作者说有时候当他
需要在某行代码处debug 时，他会在开始debug 的地方放一个表达式1/0。启用pdb，
在iPython 中运行代码。当解释器处理到1/0 那一行时， 就会产生一个ZeroDivisionError 异常，然后它就从指定的代码处被带到一个debugging session 中了。
8. 运行。有时候在一个交互式Shell 中，如果可以运行某个源文件中的内容将会很有用。
运行magic 关键字run 带一个源文件名就可以在iPython 解释器中运行一个文件了（例
如run <源文件> <运行源文件所需参数>）。参数主要有以下这些：
-n 阻止运行源文件代码时{{{__name__}}}变量被设为"{{{__main__}}}"。这会
防止if {{{__name__}}} == "{{{__main__}}}":块中的代码被执行。
-i 源文件在当前iPython 的名字空间下运行而不是在一个新的名字空间中。如果
你需要源代码可以使用在交互式session 中定义的变量，它会很有用。
-p 使用Python 的profiler 模块运行并分析源代码。使用该选项的代码不会运行
在当前名字空间。

9. 宏。宏允许用户为一段代码定义一个名字，这样可在以后使用这个名字来运行这段
代码。就像在magic 关键字edit 中提到的，列表切片法也适用于宏定义。假设有一
个历史记录如下：

```
In [3]: hist
1: l = []
2:
for i in l:
print i
```

可以这样来定义一个宏：

```
In [4]: macro print_l 2
Macro `print_l` created. To execute, type its name (without quotes).
Macro contents:
for i in l:
print i
```

运行宏：

```
In [5]: print_l
------> print_l()
```

在这里，列表l 是空的，所以没有东西被输出。但这其实是一个很强大的功能，赋予
列表l 某些实际值，再次运行宏就会看到不同的结果：

```
In [7]: l = range(5)
In [8]: print_l
------> print_l()
0
1
2
3
4
```

当运行一个宏时就好像你重新输入了一遍包含在宏print_l 中的代码。它还可以使用
新定义的变量l。由于Python 语法中没有宏结构（也许永远也不会有），在一个交互
式shell 中它更显得是一个有用的特性。
10. 环境（Profiles）。就像早先提到的那样，iPython 安装了多个配置文件用于不同的环
境。配置文件的命名规则是ipythonrc-。要使用特定的配置启动iPython，需要这样：
ipython -p
一个创建自己环境的方法是在$HOME/.ipython 目录下创建一个iPython 配置文件，
名字就叫做ipythonrc-，这里是你想要的环境名字。如果同时进行好几个项目，而这
些项目又用到互不相同的特殊的库，这时候每个项目都有自己的环境就很有用了。
也可以为每个项目建立一个配置文件，然后在每个配置文件中import 该项目中经常
用到的模块。
11. 使用操作系统的Shell。使用默认的iPython 配置文件，有几个Unix Shell 命令（当然，
是在Unix 系统上），cd、pwd 和ls 都能像在bash 下一样工作。运行其他的shell 命
令需要在命令前加!或!!。使用magic 关键字%sc 和%sx 可以捕捉shell 命令的输出。
pysh 环境可以被用来替换掉shell。使用-p pysh 参数启动的iPython，可以接受并执
行用户$PATH 中的所有命令，同时还可以使用所有的Python 模块、Python 关键字和
内置函式。

例如，想要创建500 个目录，命名规则是从d_0_d 到d_499_d，可以使用
-p pysh 启动iPython，然后就像这样：

```
[~/ttt]|1> for i in range(500):
|.> mkdir d_${i}_d
|.>
```

这就会创建500 个目录：

```
[~/ttt]|2> ls -d d* | wc -l
500
```

注意这里混合了Python 的range 函式和Unix 的mkdir 命令。虽然ipython -p pysh 提
供了一个强大的shell 替代品，但它缺少正确的job 控制。在运行某个很耗时的任务
时按下Ctrl+Z 键将会停止iPython session 而不是那个子进程。
最后，退出iPython。可输入Ctrl+D 键（会要求你确认），也可以输入Exit 或Quit（注
意大小写）退出而无须确认。
小结
经过本文对iPython 的特性及其基本使用方法的介绍，已经充分感受到iPython 的强大功
能了吧！那么，就把它作为一个有利的工具帮助我们开发吧！对于进一步的配置和更多
的用途，有兴趣的读者可以在以下网络上发掘更丰富的资料。
iPython 网站：http://ipython.scipy.org
iPython 英文文档：http://ipython.scipy.org/moin/Documentation
精巧地址：http://bit.ly/GdPZU
iPython 中一些magic 函式：http://guyingbo.javaeye.com/blog/111142
精巧地址：http://bit.ly/3GVxE8

Ipython很强大，特性也很多，但是我根本记不住，常用的特性记录下：

A　自动补全功能

需要单独安装pyreadline模块，下载地址

http://launchpad.net/pyreadline/1.6/1.6.1/+download/pyreadline-1.6.1.win32.exe

安装后进入Ipython交互命令行后执行下面代码后方可用，标准python命令行也是如此 ：

>>>import readline,rlcompleter

>>>readline.parse_and_bind(‘tab complete’)

B 方便的?

?moudle 可以查看 模块的说明，显示的比dir好看

C 历史

hist可以查看到输入命令的历史

hist -n 去掉历史前面的行数

按Ctrl+p 可以快捷输入上次输入的命令，如果再按之前输入一个匹配项，则会匹配历史命令，然后输入

D快速编辑

hist查出来的语句可以通过 edit [x:y]来快速插入到 默认的文本编辑器里，很是方便

E执行系统命令

!command可以执行 系统的命令，并返回例如：

!whoami

python中的变量需要在前面加上$，才能在shell命令中使用

F magic关键字

输入magic可以看到详细说明，ipython定义了一些常见的命令，执行用户输入的时候，会先判断 是否为magic关键字，如果是，他直接就处理了 。edit就是一个magic keyword

G 运行源码

run pyname.py [-arg]

* -n 阻止运行源文件代码时__name__变量被设为”__main__”。这会防止

if __name__ == “__main__”:

块中的代码被执行

* -i 源文件在就当前IPython的名字空间下运行而不是在一个新的名字空间中。如果你需要源代码可以使用在交互式session中定义的变量就会很有用。
* -p 使用Python的profiler模块运行并分析源代码。使用该选项代码不会运行在当前名字空间。