# vim的基本命令

## 重点：vim最重要的地方在于

1. **理解命令格式为 [n]pw ，其中n表示重复次数，p是命令，w是命令的参数；n=0不是次数，而是移动到开头**
2. **掌握vim的移动，支持正则表达式移动，/ ?；随机移动；gg/G移动**
3. **命令的大小写表示方向的不同f/F,t/T**

* i → *Insert* 模式，按 ESC 回到 *Normal* 模式.
* x → 删当前光标所在的一个字符。
* :wq → 存盘 + 退出 (:w 存盘, :q 退出)   （陈皓注：:w 后可以跟文件名）
* dd → 删除当前行，并把删除的行存到剪贴板里
* p → 粘贴剪贴板

**推荐**:

* hjkl (强例推荐使用其移动光标，但不必需) →你也可以使用光标键 (←↓↑→). 注: j 就像下箭头。
* :help <command> → 显示相关命令的帮助。你也可以就输入 :help 而不跟命令。（陈皓注：退出帮助

 **各种插入模式**

* a → 在光标后插入
* o → 在当前行后插入一个新行
* O → 在当前行前插入一个新行
* cw → 替换从光标所在位置后到一个单词结尾的字符

 **简单的移动光标**

* 0 → 数字零，到行头
* ^ → 到本行第一个不是blank字符的位置（所谓blank字符就是空格，tab，换行，回车等）
* $ → 到本行行尾
* g\_ → 到本行最后一个不是blank字符的位置。
* /pattern → 搜索 pattern 的字符串（陈皓注：如果搜索出多个匹配，可按n键到下一个）

 **拷贝/粘贴** （陈皓注：p/P都可以，p是表示在当前位置之后，P表示在当前位置之前）

* P → 粘贴
* yy → 拷贝当前行当行于 ddP

 **Undo/Redo**

* u → undo
* <C-r> → redo

 **打开/保存/退出/改变文件**(Buffer)

* :e <path/to/file> → 打开一个文件
* :w → 存盘
* :saveas <path/to/file> → 另存为 <path/to/file>
* :x， ZZ 或 :wq → 保存并退出 (:x 表示仅在需要时保存，ZZ不需要输入冒号并回车)
* :q! → 退出不保存 :qa! 强行退出所有的正在编辑的文件，就算别的文件有更改。
* :bn 和 :bp → 你可以同时打开很多文件，使用这两个命令来切换下一个或上一个文件。（陈皓注：我喜欢使用:n到下一个文件）

下面，让我们看一下vim是怎么重复自己的：

1. . → (小数点) 可以重复上一次的命令
2. N<command> → 重复某个命令N次

面是一个示例，找开一个文件你可以试试下面的命令：

* 2dd → 删除2行
* 3p → 粘贴文本3次
* 100idesu [ESC] → 会写下 “desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu desu “
* . → 重复上一个命令—— 100 “desu “.
* 3. → 重复 3 次 “desu” (注意：不是 300，你看，VIM多聪明啊).

 NG → 到第 N 行 （陈皓注：注意命令中的G是大写的，另我一般使用 : N 到第N行，如 :137 到第137行）

 gg → 到第一行。（陈皓注：相当于1G，或 :1）

 G → 到最后一行。

 按单词移动：

1. w → 到下一个单词的开头。
2. e → 到下一个单词的结尾。

> 如果你认为单词是由默认方式，那么就用小写的e和w。默认上来说，一个单词由字母，数字和下划线组成（陈皓注：程序变量）

> 如果你认为单词是由blank字符分隔符，那么你需要使用大写的E和W。（陈皓注：程序语句）

* % : 匹配括号移动，包括 (, {, [. （陈皓注：你需要把光标先移到括号上）
* \* 和 #:  匹配光标当前所在的单词，移动光标到下一个（或上一个）匹配单词（\*是下一个，#是上一个）

还有很多时间并不一定你就一定要按y才会拷贝，下面的命令也会被拷贝：

* d (删除 )
* v (可视化的选择)
* gU (变大写)
* gu (变小写)
* 等等

 0 → 到行头

 ^ → 到本行的第一个非blank字符

 $ → 到行尾

 g\_ → 到本行最后一个不是blank字符的位置。

 fa → 到下一个为a的字符处，你也可以fs到下一个为s的字符。

 t, → 到逗号前的第一个字符。逗号可以变成其它字符。

 3fa → 在当前行查找第三个出现的a。

 F 和 T → 和 f 和 t 一样，只不过是相反方向

# vim替换操作s

标准的模式为 :n1,n2 s/regex/replace/how

n1,n2有以下几种模式

%每一行

1,$ 到最后一行

. 当前行

how有以下几种

空 替换一次

g 一行中的所有

c 带有交互确认的操作

# vim配置文件

在~/.vimrc中配置你的vim个性化文件

# vim设置跳转tag

# vim自动补全

ctrl-n p 下一条和上一条建议

# vim esc可用ctrl c代替

# vim changelist

有以下几种模式

:help changelist

g; 后

g, 前

这些并不是光标所处的位置,而是编辑过得位置.

# vim ctrl+o回到上次编辑的位置

ctrl+o

ctrl+i

see :help jumplist

‘ 是一个真正的jmp命令,它会改变jumplist

m 设置一个标记

‘’ 最后一个标记

查看 vim 单引号跳转这一节

注意jumplist与changelist的不同, jumplist可以在文件之间跳转.

这些命令需要在命令模式下使用.如果是在编辑模式下使用,可能是缩进相关的命令.

# vim 大小写翻转

~ 转换

# vim 区域选择

ctrl v 选择类似于cmd

# vim 单引号跳转

mx 用x标记一行

‘x 跳转到x标记处

:’a,’b co ‘c 在位置a，b处复制到c

# vim记录宏和保存加载

宏通过@m m是一个小写字母来引用

Each register is identified by a letter a to z.

To enter a macro, type:

q<letter><commands>q

To execute the macro <number> times (once by default), type:

<number>@<letter>

@@ 重复上一个命令

# ctags

Ctrl+] - go to definition  
Ctrl+T - Jump back from the definition.  
Ctrl+W Ctrl+] - Open the definition in a horizontal split

Add these lines in vimrc  
map <C-\> :tab split<CR>:exec("tag ".expand("<cword>"))<CR>  
map <A-]> :vsp <CR>:exec("tag ".expand("<cword>"))<CR>

Ctrl+\ - Open the definition in a new tab  
Alt+] - Open the definition in a vertical split

After the tags are generated. You can use the following keys to tag into and tag out of functions:

Ctrl+Left MouseClick - Go to definition  
Ctrl+Right MouseClick - Jump back from definition

参考: <http://tartley.com/?p=1277>

安装完毕之后才能使用

# vim的正则表达式

vim默认认为正则表达式的符号必须转义，这是为了方便快捷输入特殊符号。当然，这种特性由\v very magic 任何元字符都不加转义 \V very nomagic 任何元字符都转义

\vV选项用在一个正则表达式的开头，用于暗示是否开启

set magic 即不转义

set nomagic 即转义

# ex 命令集

命令格式 :[range] cmd args

ex命令集同样适用于sed，因为sed是远古时代ed编辑器的扩展

* **:=** -- 打印文件的总行数
* **:/pattern/=** -- 打印匹配pattern的行号
* **:wq** -- 保存并退出.
* **:wa** -- 把所有的buffer写入文件,也即保存所有的文件(如果你打开了多个文件的话).
* **:x** -- 保存并退出,程序员最应该使用的命令.它与:wq的区别在于,如果文件没有修改:x是不会修改文件的时间戳,因此就不需要重新编译.而:wq命令,即使文件未修改也会修改时间戳,也即会触发重新编译.
* **:xa** -- 保存并退出所有的文件,如果打开了多个文件话.
* **:w** *a-new-file* -- 另存为a-new-file
* **:[range] w** *a-new-file* -- 把[range]所指定的范围的内容另存为a-new-file
* **:[range] w >>** *another-file* -- 把[range]所指定的范围的内容追加到another-file中
* **:r** *file* or **:read** *file* -- 把file文件中的内容读出来放到当前行下面.细节是打开file,拷贝其中的内容然后粘贴到当前行下面.
* **:n r** *file* -- 把file中的拷贝到第n行下面
* **:/pattern/ r** *file* -- 把file中的内容拷贝到第一个匹配pattern的行下面
* **:e** *another-file* -- 把another-file读入内存并开始编辑
* **:e!** -- 舍弃所有的未保存的修改,回到上一次文件所保存的状态

range部分：

% 所有行 $最后一行 .当前行 +/-x 相对于当前-+x行

**/pattern/, /pattern2/** -- 第一个含有pattern的行到第一个含有pattern2的

：数字 直接跳转

；一起按顺序执行多个命令，如 :50;40 d

* **:[range] d** -- 删除.如:1, 20 d删除第1到第20行
* **:[range] m target** -- 后面需要加上移动的目的地.剪切[range]指定的部分粘贴到target所指定的位置.
* **:[range] co/t target** -- 把[range]指定的部分拷贝到target位置
* **:[range] y/ya** -- 复制[range]到剪切板中
* **:[range] y** *name* -- 复制[range]到剪切板中,并且命名为a,这样就可以同时操作多个拷贝
* **:n pu** -- 把剪切板中的内容粘贴到第n行下面
* **:n pu** *name* -- 把剪切板中名字为name的拷贝粘贴到第n行下面
* :3,6w another：将3-6行另存为文件another（编辑神器）  
  :9,13w >>another：将9-13行追加到文件another后面（编辑神器2）

# sh 进入终端

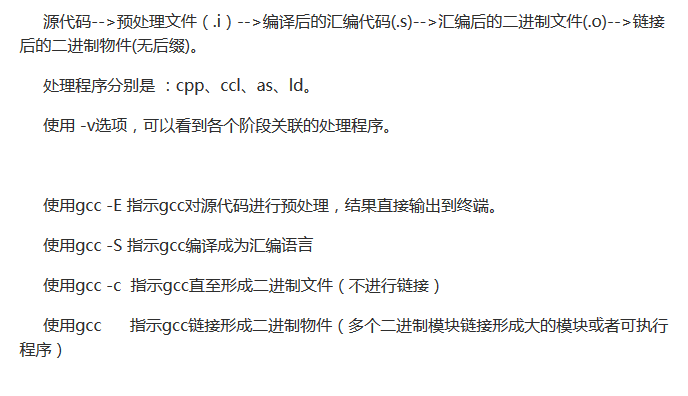
!cmd 执行命令

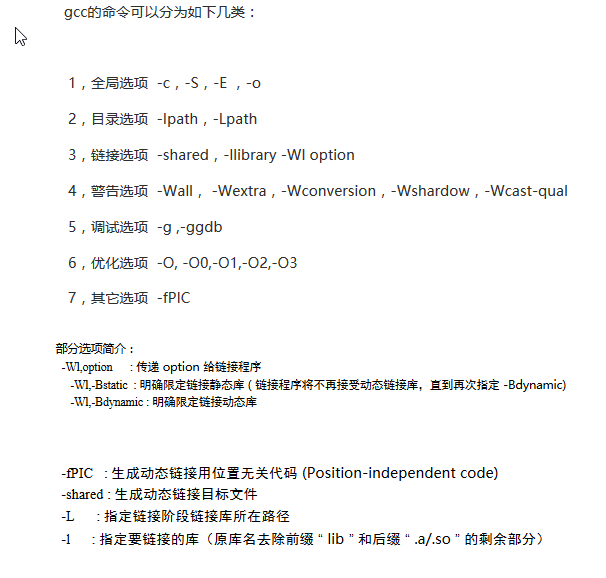
r!替换结果

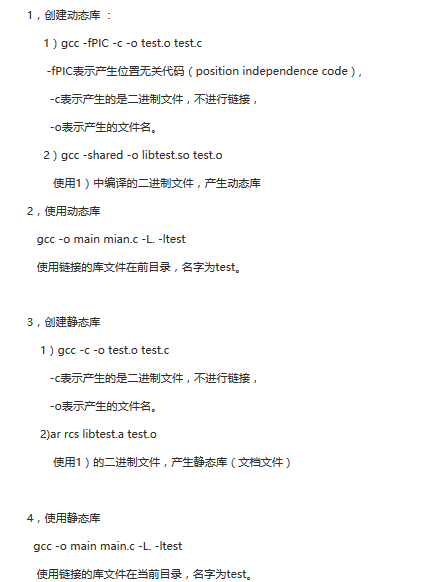
# GCC的使用：GCC最重要的就是理解文件的分步生成顺序和库，理解链接和编译的差别，知道如何生成动态库和静态库；使用ldd工具可以查看可执行文件的依赖选项

-Wl的参数：-Wl,-soname,libto.so.1

-v 可以看到各个阶段的详细信息

生成顺序：





# readline库 ： 同emacs的快捷键

c-l 清除屏幕

c-a/e 行首和行尾

c-b/f 回退/前进一个字符

c-c 终止运行

c-g 引起一个响铃

c-h/d 前进/回退删除

c-i tab键

c-j 回车，\a

c-k 清除直到行尾

c-m 回车，carriage

c-n 从history中选择一条命令

c-o 新的一行

c-p 上一个命令

c-r 回溯搜索命令

c-u 清除一行

c-v 输入转义的控制符

c-w 删除一个单词

c-y 粘贴之前的数据

c-/ 撤销

可以把整个命令行看成一个emacs编辑器，使用ctrl的命令按行移动，使用meta的命令按单词移动

# VIM 进阶教程

Reference: <http://www.yolinux.com/TUTORIALS/LinuxTutorialAdvanced_vi.html>

command line arguments

| Arguments | Description |
| --- | --- |
| +[num] | Open editor with cursor on line "num". If "num" is not specified, the cursor will be on the last line of the file. |
| +/{pat} | Open editor with cursor on the first occurrence of {pat}. |
| -c {command} --cmd {command} | A "ex" command in dowble quotes will be processed against the file specified. |
| -b | Binary file mode. |
| -C -v | VI compatibility mode. Loses the more advanced vim features. |
| -d | Diff file mode. Must list all files to perform a diff upon (list 2, 3 or 4 files). Same as vimdiff. |
| -g | GUI gvim mode (if compiled in and available). |
| -h --help | Print help messages. Also see vimtutor |
| -i *filename* | Specify viminfo file. Default is ~/.viminfo |
| -r -L | Recovery mode. Used after a crash. The ".swp" file is used. See ":help recovery". |
| -M -R | File modifications and write not allowed. |
| -n | Prohibit ".swp" file generation. Required for special devices of limited space. |
| -x | Use encryption when writing files. Will prompt for a crypt key. |
| --noplugin | Skip loading plugins. |
| --version | Print vim version. |

**Cursor Movement Commands:**

| Keystrokes | Action |
| --- | --- |
| h/j/k/l | Move cursor left/down/up/right |
| spacebar | Move cursor right one space |
| -/+ | Move cursor down/up in first column |
| ctrl-d n ctrl-d | Scroll **d**own one half of a screen. Set scroll to "n" lines. New default set for half screen. |
| ctrl-u n ctrl-u | Scroll **u**p one half of a screen Set scroll to "n" lines. New default set for half screen. |
| ctrl-f n ctrl-f | Scroll **f**orward one screen Scroll forward "n" screen |
| ctrl-b n ctrl-b | Scroll **b**ack one screen Scroll back "n" screen |
| ctrl-y n ctrl-y | Scroll forward one line Scroll forward "n" lines |
| ctrl-e n ctrl-e | Scroll back one line Scroll back "n" lines |
| M (shift-m) | Move cursor to middle of page |
| H (shift-h) | Move cursor to top of page |
| L (shift-l) | Move cursor to bottom of page |
| W w 5w | Move cursor a **word** at a time (white space delimited) Move cursor a word at a time (first non-alphanumeric) Move cursor ahead 5 words |
| B b 5b | Move cursor **back** a word at a time (white space delimited) Move cursor back a word at a time (first non-alphanumeric) Move cursor back 5 words |
| E e 5e | Move cursor to **end** of word (white space delimited) Move cursor to end of word (first non-alphanumeric) Move cursor ahead to the end of the 5th word |
| 0 (zero) | Move cursor to beginning of line |
| :30 | Move cursor to line thirty |
| $ | Move cursor to end of line |
| ) | Move cursor to beginning of next sentence (delimeted by ".", "?" or "!") |
| ( | Move cursor to beginning of current sentence |
| } | Move cursor to beginning of next paragraph (delimeted by blank line or nroff macros: .IP, .LP, .PP, .QP, .P, .LI and .bp) Also see "set paragraphs" to define a paragraph. |
| { | Move cursor to beginning of current paragraph |
| ]] | Move cursor to beginning of next section (delimeted by nroff macros: .NH, .SH, .H, .HU). Also see "set sections" to define a section. |
| [[ | Move cursor to beginning of current section |
| G | Move cursor to end of file |
| **%** | **Move cursor to the matching bracket. Place cursor on {}[]() and type "%".  Use the** [**matchit**](http://www.vim.org/scripts/script.php?script_id=39) **or** [**xmledit**](http://www.vim.org/scripts/script.php?script_id=301) **plug-in to extend this capability to XML/XHTML tags.** |
| '. | Move cursor to previously modified line. |
| m ma | Mark the line on which the cursor resides. Marking requires an identifier. Mark the line as identified by the letter "a" by marking with keystroke "ma" |
| 'a | Move cursor to line mark "a" generated by marking with keystroke "ma" |
| 'A | Move cursor to line mark "A" (global between buffers) generated by marking with keystroke "mA" |
| ]' | Move cursor to next lower case mark. |
| [' | Move cursor to previous lower case mark. |

|  |  |
| --- | --- |
| . | repeat last command action. |

**File Information:** ctrl+g or :f

**Show Special Character: :**set list/nolist

**Show Command Line Args:** :args

**Edit Newfile:** :e newfile #this will close current editing file.

**:set all** - display all mode settings of your editing session.   
**:set termcap** - display terminal settings of your editing session.

**:set ic** - Change default to ignore case for text searches  
Default is changed from noignorecase to ignorecase. (ic is a short form otherwise type **set ignorecase**)

Common options to set:

| Full "set" Command | Short form | Description |
| --- | --- | --- |
| :set autoindent :set noautoindent | :set ai :set noai | {CR} returns to indent of previous line.  Turn on autoindent: :set ai  Turn off autoindent: :set noai  Set indent width: set shiftwidth=4  Intelligent auto-indent: set smartindent  Toggle autoindent on/off when pasting text (press F2 key to toggle mode after one is in "insert" mode): set pastetoggle=<F2> |
| :set autowrite :set noautowrite | :set aw :set noaw | This tells vim to automatically write the file when switching to edit another file. See tags, editing multiple files (next, rewind) |
| :set backspace=indent,eol,start :set backspace | :set bs=indent,eol,start :set bs | Allow backspacing over an indent, line break (end of line) or start of an insert |
| :set backup=on :set backup=off | :set bk=on :set bk=off | Create backup file of file changes while editing.  To automatically remove the backup file after the file being edited is written, use the option :set writebackup=on/off  File backup mode settings: :set backupcopy=yes/no/auto |
| :set cryptmethod=zip :set cryptmethod=blowfish |  | Set file encryption for file save of buffer contents.   * zip: pkzip * blowfish: strong encryption   This is set upon reading a file if encrypted. Vim 7.3+ |
| :set errorbells :set noerrorbells | :set eb :set noeb | Silence error beep |
| :set flash :set noflash | :set fl :set nofl | Screen flashes upon error (for deaf people or when noerrorbells is set) |
| :set tabstop=8 | :set ts | Tab key displays 8 spaces |
| :set ignorecase :set noignorecase | :set ic :set noic | Case sensitive searches |
| :set number :set nonumber | :set nu :set nonu | Display line numbers |
| :set showmatch :set noshowmatch | no abbreviations | Cursor shows matching ")" and "}" |
| :set showmode :set noshowmode | no abbreviations | Editor mode is displayed on bottom of screen |
| :set showmatch :set noshowmatch | no abbreviations | Cursor shows matching ")" and "}" |
| **:set syntax on :set syntax off** | **no abbreviations** | **Set syntax highlighting and color highlighting for a file type (eg XML, HTML, C++, Java, etc). Also cursor shows matching ")" and "}" Also can set syntax highlighting explicitly: :set syntax=html Syntax definition files: /usr/share/vim/vim73/syntax/** |
| :set taglength | :set tl | Default=0. Set significant characters |
| :set closepunct='".,;)]} |  | % key shows matching symbol. Also see showmatch |
| :set linelimit=1048560 |  | Maximum file size to edit |
| :set wrapscan :set nowrapscan | :set ws :set nows | Breaks line if too long |
| :set wrapmargin=0 :set wrapmargin=8 :set nowrapmargin | :set wm   :set nowm | Define right margin for line wrapping. Wrap when past 8 characters from the edge of column display (often default 80). |
| :set list :set nolist |  | Display all Tabs and Ends of lines (Dislays these hidden characters). |
| :set bg=dark :set bg=light |  | VIM: choose color scheme for "dark" or "light" console background. |

**Buffers:** use “[a-z] to reference to a named buffer. Buffer show be prefix of p,y, etc.

Storage buffers can be named with letters of the alphabet: a-z.   
To capture lines into the buffer:

* Single "yanked" line stored in buffer "a": **"ayy** - yanks a single line (defined by current cursor position) into the buffer named "a"
* Deleted line stored in buffer "b": **"bdd** - deletes a single line (defined by current cursor position) into the buffer named "a"
* Deletes 4 lines and stores in buffer "t": **"t4dd** - deletes four lines (defined by current cursor position) into the buffer named "a"

To paste lines from a named buffer:

* Single line stored in buffer "a": **"ap** - paste contents of the buffer named "a" after the current line (defined by current cursor position)

**Share**

**Multiple Frames:** the basic command is ‘CTRL+W’

vim -o x y #open two windows,split vertically

c-w n #open new window without file

c-w s #split current windows

c-w o #keep only one window

c-w c-w #quickly switching

:new file #start with new window

5 c-w +/- #resize a window

## Moving windows cheat sheet

|  |  |
| --- | --- |
| Key combination | Action |
| CTRL-W h | move to the window on the left |
| CTRL-W j | move to the window below |
| CTRL-W k | move to the window above |
| CTRL-W l | move to the window on the right |
| CTRL-W t | move to the TOP window |
| CTRL-W b | move to the BOTTOM windo |

**Block Indent:**

**vim: Shift a block of code left or right (method #2):**

* **:20,40>**   
  Shift text from row 20 to 30, to the right
* **:20,40<**   
  Shift text from row 20 to 30, to the left

**Cooperate With Shell:**

**Moving columns, manipulating fields and awk:**   
**:'t,. !awk '{print $3 " " $2 " " $1}'** - This will reverse the order of the columns in the block of text. The block of text is defined here as from the line marked with the keystroke **"bt"** and the current line (**"."**). This text block is referenced as "**'t,.**" (man page: [awk](http://man.yolinux.com/cgi-bin/man2html?cgi_command=awk))

aaa bbb ccc ccc bbb aaa

xxx yyy zzz becomes-> zzz yyy xxx

111 222 333 333 222 111

**Macro** use @ to record and reference a macro. definition between q..q, reference between @

Each register is identified by a letter a to z.

To enter a macro, type:

q<letter><commands>q

To execute the macro <number> times (once by default), type:

<number>@<letter>

So, the complete process looks like:

|  |  |
| --- | --- |
| qd | start recording to register d |
| ... | your complex series of commands |
| q | stop recording |
| @d | execute your macro |
| @@ | execute your macro again |

**Registers**

use the **:registers** command to view the current contents of any or all register values in Vim. For example, use :reg to view all registers, or :reg a to view only what you have recorded into register a. Typing :reg abx will show the contents of registers a, b, and x.

**Modes :**  normal

|  |  |  |
| --- | --- | --- |
| Name | Description | help page |
| normal | For navigation and manipulation of text. This is the mode that vim will usually start in, which you can usually get back to with **ESC**. | :help Normal-mode |
| insert | For inserting new text. The main difference from vi is that many important "normal" commands are also available in insert mode - provided you have a keyboard with enough meta keys (such as Ctrl, Alt, Windows-key, etc.). | :help Insert-mode |
| visual | For navigation and manipulation of text selections, this mode allows you to perform most normal commands, and a few extra commands, on selected text. | :help Visual-mode |
| select | Similar to visual, but with a more MS Windows-like behavior. | :help Select-mode |
| command-line | For entering editor commands - like the help commands in the 3rd column. | :help Command-line-mode |
| Ex-mode | Similar to the command-line mode but optimized for batch processing. | :help Ex-mode |

## GenKeyForTyping程序，基于Python3

流程：

随机生成15个可打印的字符

等待这15个字符的输入，如果输入不足15字符，先进行验证；如果输入超过15个字符，丢弃多余的字符

验证是否完全切合，如果有错，查找第一个出错的地方并从这里开始重新输入

重复这个过程

代码：

import random

import time

random.seed(time.time())

def genChars(num):

'''except one thing:if a character is l(little L) or 1(one), it should always causes no

confusion,so all l must be L.

and O wiht 0.all O(big o) must be o

'''

ls=[]

while num>0:

ch=chr(random.randrange(1,128))

if ch.isprintable():

if ch == 'l':

ch = 'L'

elif ch == 'O':

ch = 'o'

ls.append(ch)

num = num - 1

return ls

def verifyChars(right\_chars,to\_verify):

'''verify the input with the generated charset.'''

if len(to\_verify) > len(right\_chars):

to\_verify = to\_verify[:len(right\_chars)]

for i,ch in enumerate(to\_verify):

if ch != right\_chars[i]:

return i

return i+1

def main():

try:

while True:

start = time.time()

right = ''.join(genChars(15))

print(right)

previous\_len = 0

wrong\_times = 0

while True:

chars = input()

i = verifyChars( right, chars )

if i == len(right):

break

if i != len(chars):

wrong\_times = wrong\_times + 1

right = right[i:]

previous\_len = previous\_len + i

print( ' ' \* previous\_len,end="")

end = time.time()

print("\*\*\*\*\*\*")

print('Time:',int(end-start),'ms')

print( 'Wrong Count:',wrong\_times)

print("\*\*\*\*\*\*")

except KeyboardInterrupt as e:

print('Exit.',end='')

if \_\_name\_\_ == '\_\_main\_\_':

main()