Intro

> Other way to explain *Practical-Vim-Edit-Text-at-the-Speed-of-Thought.pdf*
>
> 学习之前你必需知道什么是VIM,和常见的模式.
>
> 按键约定

表示	意思
X	Press X once
dw	In sequence, press d , then w
dap	In sequence, press d , a , then p
< C-n>	Press < Ctrl> and n at the same time
g< C-]>	Press g , followed by < Ctrl> and] at the same time
< C-r>0	Press < Ctrl> and r at the same time, then 0
< C-w>< C-=>	Press < Ctrl> and w at the same time, then < Ctrl> and = at the same time
f{char}	Press f , followed by any other character
`{a-z}	Press ` , followed by any lowercase letter
m{a-zA-Z}	Press m , followed by any lowercase or uppercase letter
d{motion}	Press d , followed by any motion command
< C-r> {register}	Press < Ctrl> and r at the same time, followed by the address of a register
< Esc>	Press the Escape key
< CR>	Press the carriage return key (also known as)
< Ctrl>	Press the Control key
< Tab>	Press the Tab key
< Shift>	Press the Shift key
< S-Tab>	Press the and keys at the same time
< Up>	Press the up arrow key
< Down>	Press the down arrow key
␣	Press the space bar

> 提示如果你在表格最后一行看到方框说明你的编辑器不支持UTF16

> VI 家族
>
> 假如你和我一样对一个开源的事物非常感兴趣那么我们一起来分享一下,VI的大家族成员吧, *ex*是原始的unix 编辑器那时候还不兴盛显示器,几乎是和电报机的输入是是一个年代,后来才有了*ed*,有了可视化使编辑器后才有了*emen*ex ,当有了visual才真正的有了*vi*,*vim*就是improved.参见wiki

> 获取帮助 在所有普通模式下(normal)输入:h sth

Content

> 此部分讲述所有重要的Tips,以练习的方式,部分讲解不到位的可与我交流,也可以参原文档

Tip1 使用. 命令(Meet the Dot Command)

> . 命令几乎可以是vim最常用的命令之一(除了< Esc>之外),作用是重复最后一次更改,
> 类似于CAD中的空格键.

```
original
```

Line one
Line two
Line three
Line four

```
keystrokes
```

`x...`

```
result
```

```
one
Line two
Line three
Line four
```

> original:你可以用u来回退(undo)

```
original
```

Line one
Line two
Line three
Line four

```
keystrokes
```

`dd..`

```
result
```

```
Line three
Line four
```

```
original
```

Line one
Line two
Line three
Line four

```
keystrokes
```

`j>Gj.j.`

```
result
```

```
Line one
    Line two
        Line three
            Line four
```

Tip2 不要自己重复做(Don't Repeat Yourself)

```
original
```

var foo = 1
var bar = 'a'
var foobar = foo + bar

```
keystrokes
```

`A;< Esc>j.j.`

```
result
```

```
var foo = 1;
var bar = 'a';
var foobar = foo + bar;
```

> 提示:你可能得到结果是分号前有两个空格，那是因为markdown语法差异
> 如果你自行构建原始文本的话，就能得到相同的结果

一代二 功效表

一个键	两个键

C	c$
s	cl
S	^C
I	^i
A	$a
o	A< CR>
O	ko

Tip3 做一次,然后重复(Take One Step Back, Then Three Forward)

```
original
```

var foo = "method("+argument1+","+argument2+")";

```
keystrokes
```

`f+s□+□< Esc>;.;.;.`

```
result
```

```
var foo = "method(" + argument1 + "," + argument2 + ")";
```

> Explain ;表示重演上一次搜索.

Tip4 执行,重复,回退(Act, Repeat, Reverse)

> 写在前面的话,小写的u就是精髓表示undo.当然更多请看表。

更改的执行，重复来回退 对照表

Intent	Act	Repeat	Reverse
Make a change	{edit}	.	u
Scan line for next character	f{char} / t{char}	;	,
Scan line for previous character	F{char} / T{char}	;	,
Scan document for next match	/pattern	n	N
Scan document for previous match ? pattern	n	N	
Perform substitution	:s/target/replacement	&	u
Execute a sequence of changes	qx{changes}q	@x	u

Tip5 手动查找替换 (Find and Replace by Hand)

```
original
```

...We're waiting for content before the site can go live...
...If you are content with this, let's go ahead with it...
...We'll launch as soon as we have the content...

```
keystrokes(注意以:开头是VIM的命令行模式)
```

`:182,185s/content/copy/g`

or

`:182.185s@content@copy@g`

```
result
```

```
...We're waiting for copy before the site can go live...
...If you are copy with this, let's go ahead with it...
...We'll launch as soon as we have the copy...
```

> 你可以要调整:182.185s 为original文本的的行号

```
original
```

...We're waiting for content before the site can go live...
...If you are content with this, let's go ahead with it...
...We'll launch as soon as we have the content...

```
keystrokes
{start}光标在content的c字母上（本教程所以未指明{start}的都在original的最开头）
```

`*cwcopy< Esc>n.`

```
result
```

```
...We're waiting for content before the site can go live...
...If you are copy with this, let's go ahead with it...
...We'll launch as soon as we have the copy...
```

Tip6 尽量使用.命令 (Meet the Dot Formula)

---

reference Tip2

Tip7

---

reference原文档

Tip 打点你的回退(Chunk Your Undos)

> 这个很简单就是用u进行undo,这里主要讲什么会生成你的Undo列表,用的多自行发现吧

但是这里有个非常重要的知识点，

> 当你在Insert mode里面用Up Down Left Right也会生成undo事件点。

Tip9 提交可重复的更改(Compost Repeatab Changes)

```
original
```

The end is nigh

```
keystrokes
```

光标{start}@The的T上面 `$dbx`

```
result
```

```
The end is
```

```
original
```

The end is nigh

```
keystrokes
```

光标{start}@The的T上面 `$bdw`

```
result
```

```
The end is
```

```
original
```

The end is nigh

```
keystrokes
```

光标{start}@The的T上面 `$daw`

```
result
```

```
The end is
```

> 以上方法最后一个更妙，因为可以使用. repeat

Tip10 简单计算(User Counts Do simple Arithmetic)

> 使用< C-a>和< C-x> .

```
original
```

.blog, .news { background-image: url(/sprite.png); }
.blog { background-position: 0px 0px }

```
keystrokes
```

`jyypcW.news< Esc>180< C-x>`

```
result
```

```
.blog, .news { background-image: url(/sprite.png); }
.blog { background-position: 0px 0px }
.news { background-position: -180px 0px }
```

> 关于数字格式化的趣事，*007* < c-a>后不是*008* 而是*010*

Tip11 不要去计算除非你要重复操作(Don't Count If You Can Repeat)

```
original
```

I have a couple of questions.

```
keystrokes
鼠标{start}@a couple的a上面
```

`c3wsome more< Esc>`

```
result
```

```
I have some more questions.
```

```
original
```

I have a couple of questions.

```
keystrokes
```

```
鼠标{start}@a couple的a上面
```

`daw..isomemore< Esc>`

```
result
```

```
I have some more questions.
```

> 使用第二种的原因是可以不用计数，可以用u逐个回退

Tip12 组合(Combine and Conquer)

黄金法则

Operator + Motion =Action

vim操作命令 表

按键	效果
c	Change
d	Delete
y	Yank into register
g~	Swap case
gu	Make lowercase
gU	Make uppercase
>	Shift right
<	Shift left
=	Autoindent
!	Filter {motion} lines through an external program

Tip13 Insert 模式下的迅速更正(Make Corrections Instantly from Insert Mode)

> 正如标题所说的一样是在插入模式入的
>
> *按键表*

< C-h>	Delete back one character (backspace)
< C-w>	Delete back one word
< C-u>	Delete back to start of line

Tip14 返回常规模式(Get Back to Normal Mode)

> 你懂的都很简单，看下表

按键	效果
< Esc>	Switch to Normal mode
< C-[>	Switch to Normal mode
< C-o>	Switch to Insert Normal mode

> 为什么会有个Insert Normal mode呢，请接着看

```
当我们在编辑文本的时候经常会写写就到了编辑器的底部，在常规模式下可以使用
zz，使编辑区在屏幕的中央,但不要鼠标，但是在插入模式下你用zz后肯定会输入
zz这两个字符，为了实现需求我们不需要退到Normal mode-->zz-->再进入
Insert mode.  而是可以进入一个过度mode叫做Insert Normal Mode,这样
就能在INsert mode下使用 zz. 自行演示，不理解请google,或者沟通.
```

Tip15从注册区中粘贴而不离开Insert Mode (Paste From a Register Without Leaving Insert mode)

> (see :h i_CTRL-R).
> (see :h i_CTRL-R_CTRL-P).

```
original
```

Practical Vim, by Drew Neil
Read Drew Neil's

```
keystrokes
```

`yt,jA< C-r>0.< Esc>`

```
result
```

```
Practical Vim, by Drew Neil
Read Drew Neil's Practical Vim.
```

Tip16 要自己看了，翻译不来(DoBack-of-the-EnvelopeCalculations in Place)

```
original
```

6 chairs, each costing $35, totals $

```
keystrokes
```

`A< C-r>=6*35< CR>`

```
result
```

```
6 chairs, each costing $35, totals $210
```

Tip17输入非常规字符(Insert Unusual Characters By Character Code)

> 你可以输入√½这样的更多字符当然,当然要有字符集,看 See: h i_CTRL-V_digit

非常规输入 表

Keystrokes	Effect
< C-v>{123}	Insert character by decimal code
< C-v>u{1234}	Insert character by hexadecimal code
< C-v>{nondigit}	Insert nondigit literally
< C-k>{char1}{char2}	Insert character represented by {char1}{char2} digraph

Tip18 通过二全字母插入非常规字符(Insert Unusual Characters by Digraph)

Ĩ√

> : h digraph-table

Tip19 在Replace Mode下替换存在的文本(Overwirte Existing Text with Replace Mode)

```
original
```

Typing in Insert mode extends the line. But in Replace mode
the line length doesn't change.

```
keystrokes
```

`f.R, b< Esc>`

```
result
```

```
Typing in Insert mode extends the line, but in Replace mode
the line length doesn't change.
```

Tip 20 Grok Visual Mode

无

Tip21 定义一个Visual区(Define a Visual Selection)

区域定义表

Command	Effect
v	Enable character-wise Visual mode

V	Enable line-wise Visual mode
< C-v>	Enable block-wise Visual mode
gv	Reselect the last visual selection

visual 切换法表

< Esc> /	Switchto Normal mode
< C-[>	Switch to Normal mode (when used from character-, line- or
v / V /	Switch to character-wise Visual mode
< C-v>	Switch to line-wise Visual mode
v	Switch to block-wise Visual mode
V	Go to other end of highlighted text
< C-v>	block-wise Visual mode, respectively)
o	Go to other end of highlighted text

```
original
```

Select from here to here.

```
keystrokes
鼠标{start}@第二个here的第一个h上.
```

`vbboe`

```
result
```

```
Select from here to here.
____________xxxxxxxxxxxx.
```

> 注:以上x表示选中，__表示未选中.

Tip 22 重复行选择命令(Repeat Line-Wise Visual Commands)

```
original
```

def fib(n): a, b = 0, 1 while a < n: print a, a, b = b, a+b fib(42)

```
keystrokes
鼠标{start}@第一个print的p上
```

`Vj>.`

```
result
```

```
def fib(n):
    a, b = 0, 1
    while a < n:
        print a,
        a, b = b, a+b
fib(42)
```

> 如果你得到的缩进不对,请设置 :set shiftwidth=4 softtabstop=4 expandtab 不清楚请自行google或者找我

Tip23尽可能得使用Visual命令(Prefer Operators to Visual Commands where Possible)

```
original

<a href="#">one</a>
<a href="#">two</a>
<a href="#">three</a>

keystrokes
```

`vitUj.j.`

```
result

<a href="#">ONE</a>
<a href="#">TWO</a>
<a href="#">THRee</a>
```

```
original

<a href="#">one</a>
<a href="#">two</a>
<a href="#">three</a>

keystrokes
```

`gUitj.j.`

```
result
```

```
<a href="#">ONE</a>
<a href="#">TWO</a>
<a href="#">THREE</a>
```

Tip24 使用Visual-Block快速表格化(Edit Tabular Data with Visual-Block MOde)

```
original
```

Chapter Page
Normal mode 15
Insert mode 31
Visual mode 44

```
keystrokes
鼠标{start}@Chapter和Page之间
```

`< C-v>3jx...gvr|yypVr-`

```
result
```

```
Chapter         |  Page
-------------------------
Normal mode     |  15
Insert mode     |  31
Visual mode     |  44
```

Tip 25改变字段内的文本(Change Columns of Text)

```
original
```

li.one a{ background-image: url('/images/sprite.png'); }
li.two a{ background-image: url('/images/sprite.png'); }
li.three a{ background-image: url('/images/sprite.png'); }

```
keystrokes
```

鼠标{start}@第一行/image的i字母上 `< C-v>jjeccomponents< Esc>`

```
result
```

li.one a{ background-image: url('/components/sprite.png'); }
li.two a{ background-image: url('/components/sprite.png'); }
li.three a{ background-image: url('/components/sprite.png'); }

Tip26 追加在最后(Append After a Ragged Visual Block)

> 正好前面.命令的例子

```
original
```

var foo = 1
var bar = 'a'
var foobar = foo + bar

```
keystrokes
鼠标{start}@第一行的1数字上
```

`< C-v>jj$A;`

```
result
```

```
var foo = 1  ;
var bar = 'a'  ;
var foobar = foo + bar  ;
```

趣味历史

```
In the beginning, there was ed. ed begat ex, and
ex begat vi, and vi begat Vim.
    ➤ The Old Testament of Unix
```

Tip27初见Vim的命令行(Meet Vim's Command Line)

Command	Effect
:[range]delete [x]	Delete specified lines [into register x]
:[range]yank [x]	Yank specified lines [into register x]
:[line]put [x]	Put the text from register x after the specified line
:[range]copy {address}	Copy the specified lines to below the line specified by {address}
:[range]move {address}	Move the specified lines to below the line specified by {address}

:[range]join	Join the specified lines
:[range]normal {commands}	Execute Normal mode {commands} on each specified line
:[range]substitute/{pattern}/{string}/[flags]	Replace occurrences of {pattern} with {string} on each specified line
:[range]global/{pattern}/[cmd]	Execute the Ex command [cmd] on all specified lines where the {pattern} matches

Tip28 执行一条命令多行生效(Execute a Command on One or More Consecutive Lines)

```
original
Line1 <!DOCTYPE html>
    2 <html>
    3   <head><title>Practical Vim</title></head>
    4   <body><h1>Practical Vim</h1></body>
    5 </html>
```

```
keystrokes 1
```

`:1`

`:print`

```
commandline result 1
```

```
    Line1 <!DOCTYPE html>
```

keystrokes 2

`:663`
`:p`

```
commandline result 2
    5 </html>
```

keystrokes 3

`:661p`

```
commandline result 3
    3   <head><title>Practical Vim</title></head>
```

keystrokes 4

`:661,663p`

```
commandline result 4
    3   <head><title>Practical Vim</title></head>
    4   <body><h1>Practical Vim</h1></body>
    5 </html>
```

> 注:本章节中有 $ 和 % 之类的定位符，因为内嵌教程,不能演示请自行单独文本操作

```
original

<!DOCTYPE html>
<html>
    <head><title>Practical Vim</title></head>
    <body><h1>Practical Vim</h1></body>
</html>
```

```
keystrokes
行虚拟块选中原始文档中的第二到最后一行
```

`:'<,'>p`

```
commandline result
```

```
<html>
   <head><title>Practical Vim</title></head>
   <body><h1>Practical Vim</h1></body>
</html>
```

```
original

<!DOCTYPE html>
<html>
    <head><title>Practical Vim</title></head>
    <body><h1>Practical Vim</h1></body>
</html>
```

```
keystrokes
```

`:/<html>/,/<\/html>/p`

```
commandline result

<html>
   <head><title>Practical Vim</title></head>
   <body><h1>Practical Vim</h1></body>
</html>
```

```
original

<!DOCTYPE html>
<html>
    <head><title>Practical Vim</title></head>
    <body><h1>Practical Vim</h1></body>
</html>
```

```
keystrokes
```

`:/<html>/+1,/<\/html>/-1p`

```
commandline result

    <head><title>Practical Vim</title></head>
    <body><h1>Practical Vim</h1></body>
```

Tip29 使用':t'and ':m'命令重复或者移动和(DuplicateorMoveLinesUsing‘:t’and ‘:m’Commands)

> 重复类似于复制但不是copy而是duplicater

```
original

Line 1 Shopping list
     2     Hardware Store
     3         Buy new hammer
     4 Beauty Parlor
     5      Buy nail polish remover
     6      Buy nails
```

```
keystrokes
鼠标{start}@Hardware的H上
```

`:6copy.`

```
result

Line 1 Shopping list
     2     Hardware Store
     6         Buy nails
     3         Buy new hammer
     4 Beauty Parlor
     5      Buy nail polish remover
     6      Buy nails
```

> 提示这里面的6是相对行数，在文档中不是6
> 格式:[range]copy {address}
> *命令表*

Command	Effect
:6t.	Copy line 6 to just below the current line
:t6	Copy the current line to just below line 6
:t.	Duplicate the current line (similar to Normal mode yyp)
:t$	Copy the current line to the end of the file
:'<,'>t0	Copy the visually selected lines to the start of the file

> 格式:[range]move {address}

```
original
本文本参见上部分(copy部分)
```

```
keystrokes
鼠标{start}@Hardware的H上
```

`Vjj:m$`

```
result

Shopping list
    Beauty Parlor
        Buy nail polish remover
        Buy nails
    Hardware Store
        Buy nails
        Buy new hammer
```

Tip30常用技巧:使用常规模式命令在选项范围内(Run Normal Mode Commands Across a Range)

```
original
```

var foo = 1 var bar = 'a' var baz = 'z' var foobar = foo + bar var foobarbaz = foo + bar + baz

```
keystrokes
```

`A;< Esc>jVjjjj:normal.`

```
result
```

var foo = 1;
var bar = 'a';
var baz = 'z';
var foobar = foo + bar;
var foobarbaz = foo + bar + baz;

> 注:自行匹配行数，和相对行数

Tip31 重复Ex命令(Repeat the Last Ex Command)

看:h @:

Tip32用Tab补全你的Ex命令(Tab-Complete Your Ex Commands)

---用IDE的同志们，这是常用技能 看 :h :command-complete

> 主动掷出列表 `:col< C-d>`
> `:colorscheme < C-d>`

Tip33通过命令提示输入当前字母(Insert the Current Word at the Command Prompt)

```
original
```

var tally;
for (tally=1; tally <= 10; tally++) {
// do something with tally

};

```
keystrokes
鼠标{start}@第一行的tally t上面
```

`*cwcounter< Esc>`
`%s//< C-r>< C-w>/g`

var counter;
for (counter=1; counter <= 10; counter++) {
// do something with counter
};

回忆命令历史(Recall Commands from history)

> :打头，能够展示历史命令
> /打头，能够展示搜索历史
> 同样为了不离开主编辑区，可以使用< C-p>/< C-n>替代
>
> 组合命令使用|而非 bash里面的&& 比如 `:write |! ruby %
>
> 当然你也可以使用 `q:`来像Insert mode一样编辑历史命令.

进入cli-win的方式表

q/	Open the command-line window with history of searches
q:	Open the command-line window with history of Ex commands
ctrl-f	Switch from Command-Line mode to the command-line window

Tip35运行Shell 脚本(Run Commands in the Shell)

```
original
none
```

```
keystrokes
```

`:!ls`

```
result

[No write since last change]
Practical-Vim-Edit-Text-at-the-Speed-of-Thought.pdf  praticaldir  Tips.md
```

> 提示:可能显示不一样，因为目录下内容不一样
>
> 当然你也可以把vim置于后台来终端跑命令,请查看bash 的< C-z> ，jobs和fg命令.

```
original
```

first name,last name,email
john,smith,john@example.com
drew,neil,drew@vimcasts.org
jane,doe,jane@example.com

```
keystrokes
```

`:'<,'>!sort -t',' -k2`

```
result

jane,doe,jane@example.com
first name,last name,email
drew,neil,drew@vimcasts.org
john,smith,john@example.com
```

> 提示:请自行调整范围

一些命令表

Command	Effect
:shell	Start a shell (return to Vim by typing exit)
:!{cmd}	Execute {cmd} with the shell
:read !{cmd}	Execute {cmd} in the shell and insert its standard output below the cursor
:[range]write !{cmd}	Execute {cmd} in the shell with [range] lines as standard input
:[range]!{filter}	Filter the specified [range] through external program {filter}

Tip36 通过缓冲列表追踪打开的多个文件(TrackOpenFileswiththeBufferList)

> 一定要理解什么是file ,什么是buffer

> vim 能同时打开多个文件，比如vim *.html ,这种情况下，可以运行`:ls`

```
result
你可能看到类似于下列的列表:

:ls
1 %a   "[No Name]"                   line 1
Press ENTER or type command to continue
```

Tip37 通过参数列表分组Buffers(Group Buffers into a Collection with the Argument List)

> 具体可自行运行体验，或者 google 或者交流与我，当然你能直接从下表中看出端倪

Glob	Files Matching the Expansion
:args .	index.html
	app.js
:args */.js	app.js
	lib/framework.js
	app/controllers/Mailer.js
	...etc
:args */.*	app.js
	index.js
	lib/framework.js
	lib/theme.css
	app/controllers/Mailer.js
	...etc

Tip38管理隐藏文件(Manage Hidden Files)

> 实在不好讲解到此文档中，细交流于我

Tip39分窗口(Divide Your Workspace into Split Windows)

> *多窗口生成命令表*

Command	Effect
< C-w>s	Split the current window horizontally, reusing the current buffer in the new window
< C-w>v	Split the current window vertically, reusing the current buffer in the new window

:sp[lit] {file}	Split the current window horizontally, loading {file} into the new window
:vsp[lit] {file}	Split the current window vertically, loading {file} into the new window

> 编辑要焦点，*焦点切换表*

Command	Effect
w	Cycle between open windows
h	Focus the window to the left
j	Focus the window below
k	Focus the window above
l	Focus the window to the right

> *关闭窗口命令表*

Ex Command	Normal Command	Effect

:cl[ose]	c	Close the active window
:on[ly]	o	Keep only the active window, closing all others

> *调整窗口大小命令表*

Keystrokes	Buffer Contents
=	Equalize width and height of all windows
_	Maximize height of the active window
|	Maximize width of the active window
[N]_	Set active window height to [N] rows
[N]|	Set active window width to [N] columns

Tip40用Tab页来管理布局(Organize Your Window Layouts with Tab Pages)

> 你需要查看 (`:h CTRL-W_T`).
>
> *标签页操作命令表*

Command	Effect
:tabe[dit] {filename}	Open {filename} in a new tab
T	Move the current window into its own tab
:tabc[lose]	Close the current tab page and all of its windows
:tabo[nly]	Keep the active tab page, closing all others

> *标签页切换命令表*

Ex Command	Normal Command	Effect
:tabn[ext] {N}	{N}gt	Switch to tab page number {N}
:tabn[ext]	gt	Switch to the next tab page
:tabp[revious]	gT	Switch to the previous tab page

Tip41通过:edit文件路径打开文件(OpenaFileby ItsFilepathUsing‘:edit’)

> 使用:edit /path/to/file.t 打开文件。要配合使用:pwd 准确定位path
>
> 小技巧 设置
> `cnoremap <expr> %% getcmdtype() == ':' ? expand('%:h').'/' : '%%'`
> 到你的.vimrc 这样就能%%快速输入./(当前目录)了。

Tip42 通过 :find打开一个文件 (OpenaFileby ItsFilenameUsing‘:find’)

> 在工程项目中,find命令能够快速打开文件以供编辑
> 使用之前你可能 需要劫 path 比如 `:set path+=app/**`

Tip43

> 作者认识netrw已经不如NERDTree好用了，所以跳过，要了解请查看文档.
> 如果你要查看NERDTree的帮助的话github

Tip44在非存在的目录下保存文件(SaveFilestoNonexistentDirectories)

> 假如你看到这样的报错

```
result
E212: Can't open file for writing
```

> 原因之一是你碰到了非法目录。解决的思路是调用shell命令然后:write

Tip45作为超级用户保存文件(Save a File as the Super User)

> 假如你看到这样的报错

```
result
E212: Can't open file for writing
```

> 原因之二是你碰到了权限错误

```
keystrokes
```

`:w !sudo tee % > /dev/null`

Tip46确保你的手在键盘主编辑区(Keep Your Fingers on the Home Row)

Command	Move cursor
h	One column left
l	One column right
j	One line down
k	One line up

> 为了大家能够更好的完成vim学习，大家可以把< up>< down>< lef>< right>重新绑定 使用下列配置

```
noremap <Up> <Nop>
noremap <Down> <Nop>
noremap <Left> <Nop>
noremap <Right> <Nop>
```

Tip47区分显示行和真实行(DistinguishBetweenRealLinesandDisplayLines)

> 类似于windows的记事本一样，会发生一行显示不下的情况，nodepad提供了
> 自动换行的功能，vim同样也会有这问题的，默认情况下j,k会以直接行来移动光标
> 但有时候的文档却偏偏1000+个字符在一行里面，自动换行 后可通过gj/gk来
> 移动显示行.
> *操作命令表*

Command	Move Cursor
j	Down one real line
gj	Down one display line
k	Up one real line
gk	Up one display line
0	To first character of real line
g0	To first character of display line
^	To first nonblank character of real line
g^	To first nonblank character of display line
$	To end of real line
g$	To end of display line

Tip48漂移于字符间(Move Word-Wise)

> 查看 `:h word-motions`
>
> *命令表*

Command	Move Cursor
w	Forward to start of next word
b	Backward to start of current/previous word
e	Forward to end of current/next word
ge	Backward to end of previous word

```
original
```

Go fast.

```
keystrokes
```

`weaer< Esc>`

```
result
```

```
Go faster.
```

Tip49字符 (Find by Character)

> 你需要查看 `:h f`
> 看文档看上去很复杂 直接看例子

```
original
```

Find the first occurrence of {char} and move to it.

```
keystrokes 1
```

`fx`

```
result 1
Find the first occurrence of {char} and move to it.
x_________________________________________________
```

> 注:以上x表示光标位置,

```
keystrokes 2
```

`fo`

```
result 2
Find the first occurrence of {char} and move to it.
_______________x__________________________________
```

> 注:以上x表示光标位置,

```
keystrokes 3
```

`fc;;;`

```
result 3
Find the first occurrence of {char} and move to it.
________________xx_____x______x__________________
                12     3      4
```

> 注:以上x表示光标依次跳动位置，数字为顺序 以上例子有问题可以联系我，

```
keystrokes 4
```

`fo;;,`

```
result 4
Find the first occurrence of {char} and move to it.
_______________x__________x______________x_______
               1          2              3
                          4
```

> 注:以上x表示光标依次跳动位置，数字为顺序 以上例子有问题可以联系我，特别是按了`,`没反应原因很可能是`,`被重置.

命令表

Command	Effect
f{char}	Forward to the next occurrence of {char}
F{char}	Backward to the previous occurrence of {char}
t{char}	Forward to the character before the next occurrence of {char}
T{char}	Backward to the character after the previous occurrence of {char}
;	Repeat the last character-search command
,	Reverse the last character-search command

Tip50通过搜索来导航(Search to Navigate)

> 搜索命令是`/`或者`?`{char}< CR>

```
original
```

search for your target
it only takes a moment
to get where you want

```
keystrokes 1
```

`/ta< CR>`

```
result 1
```

```
search for your target
________________1_____
it only takes a moment
to get where you want
```

> 光标显示在1位置

```
keystrokes 2
```

`/tak< CR>`

```
result 2
```

```
search for your target
it only takes a moment
________1_____________
to get where you want
```

> 光标显示在1位置

Tip51

```
original

var tpl = [
    '<a href="{url}">{title}</a>'
]
```

```
keystrokes 1
鼠标{start}@第二行url的r上面
```

`vi}`

```
result 1
var tpl = [
    '<a href="{url}">{title}</a>'
_______________xxx______________
]
```

```
keystrokes 2
鼠标{start}@接上面
```

`a"`

```
result 2
var tpl = [
    '<a href="{url}">{title}</a>'
_____________xxxxxxx____________
]
```

```
keystrokes 3
鼠标{start}@接上面
```

`i>`

```
result 3

var tpl = [
    '<a href="{url}">{title}</a>'
______xxxxxxxxxxxxxx____________
]
```

```
keystrokes 4
鼠标{start}@接上面
```

`it`

```
result 4
var tpl = [
    '<a href="{url}">{title}</a>'
_____________________xxxxxxx____
]
```

```
keystrokes 5
鼠标{start}@接上面
```

`at`

```
result 5
var tpl = [
    '<a href="{url}">{title}</a>'
_____xxxxxxxxxxxxxxxxxxxxxxxxxxx
]
```

```
keystrokes 6
鼠标{start}@接上面
```

`a]`

```
result 6
var tpl = [
__________x
    '<a href="{url}">{title}</a>'
____xxxxxxxxxxxxxxxxxxxxxxxxxxxxx
]
x
```

> 注:以上x表示选中，__表示未选中.
>
> 以下*动作请每天都练习表*

a) or ab	A pair of (parentheses)
i) or ib	Inside of (parentheses)
a} or aB	A pair of {braces}
i} or iB	Inside of {braces}
a]	A pair of [brackets]
i]	Inside of [brackets]
a>	A pair of < angle brackets>
i>	Inside of < angle brackets>
a’	A pair of 'single quotes'
i’	Inside of 'single quotes'
a"	A pair of "double quotes"
i"	Inside of "double quotes"

a`	A pair of `backticks`
i`	Inside of `backticks`
at	A pair of < xml>tags< /xml>
it	Inside of < xml>tags< /xml>

> 当然你还要用操作来配合上面的表

`d{motion},c{motion},y{motion}`分别表示删除，替换，和yank所执行的选项

Tip52删除周遭,或者更新内部(Delete Around ,or Change Inside)

> *Text Objects*表

Keystrokes	Buffer Contents
iw	Current word
aw	Current word plus one space
iW	Current WORD
aW	Current WORD plus one space
is	Current sentence
as	Current sentence plus one space
ip	Current paragraph
ap	Current paragraph plus one blank line

```
original
```

Improve your writing by deleting excellent adjectives.

```
keystrokes
鼠标{start}@excellent 的x字母上
```

`daw`

```
result

Improve your writing by deleting adjectives.
```

```
original
```

Improve your writing by deleting excellent adjectives.

```
keystrokes
鼠标{start}@excellent 的x字母上
```

`ciwmost< Esc>`

```
result

Improve your writing by deleting most adjectives.
```

> 同样可以把iw,换成Tip52里面的所以{motion}

Tip53虽然不能翻译但是这节讲标记以供快速Jump(MarkYourPlaceandSnapBackto)It

> 请查看`:h m`
>
> `m{a-zA-Z}`能够定义多个 marker，以供`'{a-zA-Z}`使用.
> 当然vim还有部分*automatic Marks*的表

Keystrokes	Buffer Contents
``	Position before the last jump within current file
`.	Location of last change
`^	Location of last insertion
`[	Start of last change or yank
`]	End of last change or yank
`<	Start of last visual selection
`>	End of last visual selection

Tip 54在匹配的对应子集跳转(Jump Between Matching Parentheses)

> 本节主角是`%`

> 一些操作实效

Keystrokes	Buffer Contents
{start}	console.log([{'a':1},{'b':2}])
	____________x__________________
%	console.log([{'a':1},{'b':2}])
	______________________________x
h	console.log([{'a':1},{'b':2}])
	_____________________________x_
%	console.log([{'a':1},{'b':2}])
	_____________x_________________
l	console.log([{'a':1},{'b':2}])
	______________x________________
%	console.log([{'a':1},{'b':2}])
	____________________x__________

> 注:x为光标位置

> 一些操作实效

Keystrokes	Buffer Contents
{start}	cities = %w{London Berlin New\ York}
	________x___________________________
dt{	cities = {London Berlin New\ York}
	________x___________________________
%	cities = {London Berlin New\ York}
	________________________________x____
r]	cities = {London Berlin New\ York]
	________________________________x____
``	cities = {London Berlin New\ York]
	_______x____________________________
r[	cities = [London Berlin New\ York]
	____________________________________

> 提示`` ` ` ``命令要配合< C-o>
> 可查看*Tip13*

游戈于Jump List(Traverse the Jump List)

> `:jumps`大概你能看到一个列表表示你到过哪里，
> 可以通过< C-o>与< C-i>来切换游走
> 当然有个*快捷键*表

Command	Effect
[count]G	Jump to line number
//pattern /?pattern / n / N	Jump to next/previous occurrence of pattern
%	Jump to matching parenthesis
(/)	Jump to start of previous/next sentence
{ / }	Jump to start of previous/next paragraph
H / M / L	Jump to top/middle/bottom of screen
gf	Jump to file name under the cursor
	Jump to definition of keyword under the cursor
’{mark} / `{mark}	Jump to a mark

Tip56在Change List上跳转(Traverse The Change List)

> `:changes`大概你还能看到一个列表表示你的更新过哪里，

> 同样这次你可以用`g;或g,`

Tip57可能通过文件名来跳转(Jump to the Filename Under the Curser)

> 这个理解比较简单
> 首先(使用之前的技能`!touch /tmp/test.t`然后在当前文本中输入
> /tmp/test.t
> 操作之前请保存，不然你会迷失的

Tip58使用全局Marks(Snap Between Files Using Global Marks)

> 基本上就是`m{letter}`里面的大小写的关系，用大写字母就像使用书签一样
> 可以在文件中跳转
> 请自行演示，问题请google或call me

Tip59删除,抽取,和推送通过vim's没有命令的注册列表(Delete, Yank, and Put with Vim’s Unnamed Register)

Keystrokes	Buffer Contents
{start}	Practica lvim
	___________x
F□	Practica lvim
	_______x____
x	Practicalvim
	_______x____
p	Practical vim
	________x___

> 提示:x为水标位置，而且假如你看到F后面一个方框说明你不能显示
> 某些字符，解决方法很简单看穿尘世，

Keystrokes	Buffer Contents
{start}	2) line two
	x__________
	1) line one
	3) line three
dd	1) line one
	3) line three
pp	1) line one
	2) line two
	3) line three

> 能看懂吗？要尝试的话也直接通过以上文本实现，

> 也可以自行意念理解下

Tip60Grok Vim’s Registers

> 讲的好复杂，大意是说了些能够register的操作
> 比如`x,s,d{motion},c{motion},y{motion}`
> 可以`:reg "0`

Tip61用注册器替换选项区(Replace a Visual Selection with a Register)

```
original
```

collection = getCollection();
process(somethingInTheWay, target);

```
keystrokes
```

`yiwjwwvep`

```
result
collection = getCollection();
process(collection, target);
```

```
original
```

I like chips and fish.

```
keystrokes
```

fcdemmwwvep`mP

> 提示这儿没有代码块是因为我还不知道怎么在代码块中显示`,我也没Google在边上

```
result

I like fish and chips.
```

Tip62从Register中粘贴(Paste from a Register)

```
original
```

collection = getCollection();
process(somethingInTheWay, target);

```
keystrokes
```

`yiwjwwciw< C-r>0< Esc>`

```
result
```

collection = getCollection();
process(collection, target);

```
original

<table>
    <tr>
        <td>Keystrokes</td>
        <td>Buffer Contents</td>
    </tr>
</table>
```

```
keystrokes
```

`/<tr>yapgP`

```
result

<table>
    <tr>
        <td>Keystrokes</td>
        <td>Buffer Contents</td>
    </tr>
</table>

<table>
    <tr>
        <td>Keystrokes</td>
        <td>Buffer Contents</td>
    </tr>
</table>
```

> 如果你得到了一个table，请不要在意这些细节要看破红尘。因为你的vim
> 识别的段落不一致。

Tip63联系系统剪切板(Interact with the System Clipboard)

> 大概的意思是让你明白`:set pastetoggle=<f5>` 请自行Google pastetoggle。上面讲的比这清楚或者例子
> 不能很好展现

Tip64录制和执行一个宏(Record and Execute a Macro)

> 这节开始我们就要练习宏了，和office的宏一样
> 就是一组操作的集合

```
original
```

foo = 1 bar = 'a' foobar = foo + bar

```
keystrokes
```

`qaA;< Esc>Ivar < Esc>q`

```
result

var foo = 1;
bar = 'a'
foobar = foo + bar
```

> 在这里不能只看到最后效果，
> 请运行`:reg a` 你应该能看到如下

```
:reg a
--- Registers ---
"a   A;^[Ivar ^[
Press ENTER or type command to continue
```

> 没错我们录制了宏，开始执行吧，

```
original
```

var foo = 1;
bar = 'a'
foobar = foo + bar

```
keystrokes
鼠标{start}@var后面的空格上
```

`j@aj@@`

```
result

var foo = 1;
var bar = 'a'  ;
var foobar = foo + bar  ;
```

> 以上说明执行可以用@a 和@@来，@a可以区分不同的宏
> 而@@则可以快速执行上个录制的宏

Tip65Normalize, Strike, Abort

> TODO 无

Tip66 执行次数 (Play Back with a Count)

```
original
```

x = "("+a+","+b+","+c+","+d+","+e+")";

```
keystrokes
这次我们录制一个q的macro并执行了22次
```

`f+s + < Esc>qq;.q22@q`

```
result
```

x = "(" + a + "," + b + "," + c + "," + d + "," + e + ")";

> 是不是很神奇。

Tip67Repeat a Change on Contiguous Lines

```
original
```

1. one
2. two

```
keystrokes
光标{start}@one的o上面
```

`qa0f.r)w~jq`

1) One
2. two

> 可以看`:reg a`

```
:reg a
--- Registers ---
"a   0f.r)w~j
Press ENTER or type command to continue
```

> 这次我们要使用以上录制的宏

```
original
```

1) One
2. two
3. three

4. four

```
keystrokes
光标{start}@two的w上面
```

`3@a`

```
result

1) One
2) Two
3) Three
4) Four
```

> bingo没让你失望吧，下面请看碰到异常怎么继续

```
original
```

1. one
2. two // break up the monotony
3. three
4. four

```
keystrokes
```

`使用以上录制的宏`
`5@a`

```
result

1) One
2) Two
// break up the monotony
3. three
4. four
```

> 结果卡住了 没关系 我们要并行的执行宏来消除这个

```
original
```

1. one
2. two // break up the monotony
3. three
4. four

```
keystrokes
`使用以上录制的宏`
```

`VG:normal @a`

```
result

1) One
2) Two
// break up the monotony
3) Three
4) Four
```

> `VG`请自行调整选中区域到第1到第5

Tip68追加宏脚本(Append Commands to a Macro)

> 这节意思相当简单就是`q{letter}`会overwrite一个宏
> 但你用了`q大写字母时`就会在后面追加命令

Tip69(Act Upon a Collection of Files)

> TODO 无 请自行演示

Tip70(EvaluateanIteratortoNumberItems in a List)

```
original
```

partridge in a
French hens
calling birds
golden rings
tree

```
keystrokes
```

`let i=1` #初设置i
`qaI< C-r>=i<CR>)< Esc>`
`let i +=1` #加1
`q` #结束录制
`jVG` #同理请自行调整`VG`或者独立文件运行演示
`:normal @a`

```
result
```

1)partridge in a
2)French hens
3)calling birds

4)golden rings
5)tree

> 提示演示不通过请call me

Tip71编辑宏

> 此章节和编辑替换一样，自行演示或交流，文档不好展示

Tip72调整大小写搜索模式(Tune the Case Sensitivity of Search Patterns)

> 此节意思明了，用\c来忽略大小匹配
> 当然你也能用`:set ignorecase`来全局忽略大小写匹配

Tip73,74

> 使用\v来匹配正则式
> 关于正则是门大学问，我不想糟蹋了这个精华，请专门阅读正则大作

Tip75用模式匹配(UseParenthesestoCaptureSubmatches)

```
original
```

I love Paris in the
the springtime.

```
keystrokes
```

`/\v<(\w+)\_s+\1>`

```
result
是不是很神奇的匹配了the the
算了我又在亵渎神圣的正则请自行google
```

Tips76,77,78这几单都是讲搜索正则的

> Pass 真不想译，请GOOGLE

Tip79初见搜索模式(Meet the Search Command)

Command	Effect
n	Jump to next match, preserving direction and offset
N	Jump to previous match, preserving direction and offset
/	Jump forward to next match of same pattern
?	Jump backward to previous match of same pattern

> 一切在不言中，搜索历史的话使用< up>< down>

Tip80,81讲更友好的搜索选项

> 其实涉及一些 .vimrc的配置 如下

```
set hlsearch spell
set incsearch
set is
set smartcase
```

Tip82计算匹配的模式(Count the Matches for the Current Pattern)

> 两种方法
> 方法1
> `:%s///gn`
> 方法2
> `:%s//&/g`
>
> 大概第一种比较好，因为不会触发编辑

Tip83定位到模式末位 (Offset the Cursor to the End of a Search Match)

```
original
```

Aim to learn a new programming lang each year.
Which lang did you pick up last year?
Which langs would you like to learn?

```
keystrokes
```

`/lang/e< CR>auage< Esc>n.n.`

```
result


Aim to learn a new programming language each year.
Which language did you pick up last year?
Which languages would you like to learn?
```

> 匹配完模式后加`/e`

Tip84一个完整的匹配搜索(Operate on a Complete Search Match)

```
original
```

class XhtmlDocument < XmlDocument; end class XhtmlTag < XmlTag; end

```
keystrokes
```

`/\vX(ht)?ml\C< CR>gU//e< CR>//< CR>.//< CR> .`

```
result


class XHTMLDocument < XMLDocument; end
class XHTMLTag < XmlTag; end
```

Tip85创建一个复杂的模式(CreateComplexPatternsbyIterating uponSearch History)

> 看例子

```
original
```

This string contains a 'quoted' word.
This string contains 'two' quoted 'words.'
This 'string doesn't make things easy.'

```
keystrokes
```

1. `/\v'[^']+'`
2. `/\v'([^']|'\w)+'`
3. `/\v'([^']|'\w)+'`

```
result

请自行查看三种搜索的结果
```

Tip86Visual Selection也可以使用搜索(SearchfortheCurrentVisualSelection)

Keystrokes	Buffer Contents
{start}	She sells sea shells by the sea shore.
	xxxxxxxxxxxx__________________________
*	She sells sea shells by the sea shore.
	xxxxxxxxxxxxxxxxxxxxxxxxxxx____________

> 提示x表示选中区域