

# 잘생긴 상훈오빠랑 함께하는 빔 테마미나 #2

lucidash - 김상훈

말 하나면

Vim 잘 쓰기 위한 각종 '것' 들

- 유용한 Commands
- vimrc
- vim plugins

유용한 Command

:jump

커서 위치가 변할 때

그 위치를 `stack` 처럼 저장한다

한 파일 안, 혹은 여러 파일 사이를  
돌아다닐 수 있다.

그렇다고 hjkl 로 커서가 움직일  
때 마다 기록하는 건 아니고

## 8. Jumps

### jump-motions

A "jump" is one of the following commands: "'", "`", "G", "/", "?", "n", "N", "%", "(", ")", "[[", "]]", "{", "}", ":s", ":tag", "L", "M", "H" and the commands that start editing a new file. If you make the cursor "jump" with one of these commands, the position of the cursor before the jump is remembered. You can return to that position with the "'" and "`" command, unless the line containing that position was changed or deleted.

:help jump-motions

## Ctrl - O (중요)

- 이전 위치로 점프

Ctrl - i

- Ctrl - o 의 반대

마크 후 점프 ( 간단한 **Bookmark** 기능 )

- mx : x 라는 문자에 현재 위치를 마크
- 'x : x 라는 위치로 점프

:jumps

jump	line	col	file/text
76	95	0	ProblemSolving/B0J/1185.cpp
75	1	0	ProblemSolving/B0J/a.cpp
74	1	0	.dotfiles/.git/MERGE_MSG
73	6	0	.dotfiles/.git/MERGE_MSG
72	1	0	a.cpp
71	8	23	a.cpp
70	13	0	a.cpp
69	86	0	.vimrc
68	316	0	.vimrc
67	288	0	.vimrc
66	6	14	a.cpp
65	11	6	a.cpp
64	7	0	a.cpp
63	4	4	a.cpp
62	9	13	a.cpp
61	10	0	a.cpp
60	15	0	.vimrc
59	13	3	.vimrc
58	5	0	ProblemSolving/B0J/1185.cpp
57	97	11	ProblemSolving/B0J/1185.cpp
56	16	0	ProblemSolving/B0J/1185.cpp
55	1	0	ProblemSolving/B0J/1185.cpp
54	95	0	ProblemSolving/B0J/1185.cpp
53	1	0	ProblemSolving/B0J/a.cpp
52	16	12	.vimrc
51	18	0	.vimrc
50	304	0	.vimrc
49	299	0	.vimrc
48	27	11	.vimrc
47	1	0	.dotfiles/.git/MERGE_MSG
46	6	0	.dotfiles/.git/MERGE_MSG
45	2	11	.vimrc
44	19	1	.vimrc

## :marks

mark	line	col	file/text
'	24	0	""
A	89	4	~/workspace/zabo/zabo/apps/board/views.py
0	18	0	
1	307	32	map <F5> <ESC>:w<CR>:!g++ %<CR><CR>:!. /a.out<CR>
2	310	32	map <F3> <ESC>:!cat %\lpbcopy<CR>
3	26	15	1934.cpp
4	46	0	set enc=utf-8 tenc=utf-8
5	21	0	abc.cpp
6	18	3	abc.cpp
7	34	0	3142.cpp
8	4	0	1697.cpp
9	1	0	"" Personal vimrc
"	18	0	
[	18	0	
]	18	0	
.	18	0	

## Ctrl - O (중요)

- 이전 위치로 점프

Ctrl - i

- Ctrl - o 의 반대

마크 후 점프 ( 간단한 **Bookmark** 기능 )

- mx : x 라는 문자에 현재 위치를 마크
- 'x : x 라는 위치로 점프

간단하고 대표적인 사용 예시를 봅시다

$${}^n C_r = \left( \frac{n!}{(n-r)!r!} \right)$$

```
1 import sys
2
3 inp = str(sys.stdin.readline())
4 n = int(inp.split( ' ' )[0])
5 r = int(inp.split( ' ' )[1])
6
7
8 nCr = math.factorial( n ) / ( math.factorial( n - r ) * math.factorial( r ) )
9 █
```

import math를 해야하는데...

gg 를 입력해서 맨 위로 커서를 옮긴 후

import math 를 추가

```
1 import sys
2 import math
3
4 inp = str(sys.stdin.readline())
5 n = int(inp.split( ' ' )[0])
6 r = int(inp.split( ' ' )[1])
7
8
9 nCr = math.factorial( n ) / ( math.factorial( n - r ) * math.factorial( r ) )
10
```

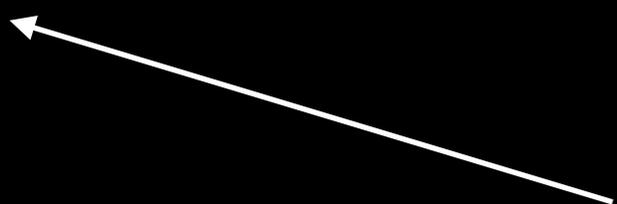
다시 아까 코딩 하던 곳으로 가서...

출력을 해야겠다!

ctrl + o

```
1 import sys
2 import math
3
4 inp = str(sys.stdin.readline())
5 n = int(inp.split( ' ' )[0])
6 r = int(inp.split( ' ' )[1])
7
8
9 nCr = math.factorial( n ) / ( math.factorial( n - r ) * math.factorial( r ) )
10 █
```

커서가 여기로 이동!



발표자는 주로 방금 살펴본 예와 같이

`#include` , `import from` 류나

전역변수를 추가하거나 할 때 많이 씀

또 mx 로 마크한 북마크로 'x 로 이동 후

다시 돌아 올 때도 유용하다!

% : 현재 커서 위치의 괄호와 짝이 맞는 괄호로 이동

{ : 이전 문단으로 커서 이동

} : 다음 문단으로 커서 이동

>> : 현재 라인을 1tab 만큼 indent

<< : 현재 라인을 1tab 만큼 unindent

Visual block 잡고서 > : 블록을 1tab 만큼 indent

Visual block 잡고서 < : 블록을 1tab 만큼 unindent

Visual block 잡고서 = : auto indent

Very Useful Command  
(in normal mode)

cW : dWi 와 같은 동작 ( (cc = ddi 와 같다! ) )

- 현재 커서 이후로 단어를 지우고  
insert mode 로 변경

C (capital c) : d\$a 와 같은 동작

- 현재 커서 이후로 개행문자까지 모두 삭제하고  
insert mode 로 변경

cc : ddO 또는 ^c\$ 와 같은 동작

- 현재 라인을 지우고 (인덴트에 맞게) 새로운 라인을 열어  
insert mode 로 진입

A : \$a 와 같은 동작

- 현재 라인의 마지막 글자 뒤로 커서 이동 후  
insert mode 로 변경

I : ( ) 와 같은 동작

- 현재 라인의 공백이 아닌 첫 글자 앞으로 커서 이동 후  
insert mode 로 변경

[t|T]x - 현재 커서가 위치한 라인에서 커서 위치 이후 처음 위치한  
x를 찾아 그 위치 바로 앞으로 커서를 이동

[f|F]x - 현재 커서가 위치한 라인에서 커서 위치 이후 처음으로 위치한  
x를 찾아 그 위치로 커서를 이동

( 대문자 T / F 는 각각 검색 방향이 커서 위치 앞으로 ... )

이전 까진 전부 ...

normal 모드에서 작동하는 녀석들이 있고  
insert mode 에서 제일 많이 쓰는 커맨드는

- `ctrl - u`

= 현재라인을 다 지워

- `ctrl - w`

= 현재 단어를 지워

하는 짓을 들어보니까 도대체

이걸 뭐할 때 쓰는거지.....

싫을 지도 모르지만

```
print 'https://www.google.com'
```

‘ ’ 내부를 지우고 새로 쓰고 싶다면 .. ?

h에 커서를 올려두고

dt' 를 하면 지워지고

dt'i( ct' ) -> 지우고 insert mode

dw는 word 단위로 지우는데  
특수문자를 기준으로 word를 취급

7dw 를 해야

<http://www.google.com>

을 다 지울 수 있음

배열 등의 [blahblah]

에서 blahblah를 지울 때도

dt] 를 하면 됨!

array[index]에서는 cw

`array[index1+index2-index3]`

에서는 `dt] ( ct] )`

[c|d]i[<|'|"|(…)] 도 가능함

- 'ci' 를 입력하면 “ 안의 내용을 지우고  
insert mode
- 'di' 를 입력하면 “안의 내용을 지우고  
normal mode

ctrl + a / ctrl + x( on Normal mode )

- 현재 커서에 위치한 integer 값을  
1 만큼 증가 / 감소

. (dot, 점) : 방금 했던 거 또 하기

( Normal mode action + insert mode action

기준으로 최근에 한 하나의 동작에 한정 )

이게 뭘 소리가 하면 .....

```
9 def fibonacci( n ):
10     if ( n == 0 ) return 1
11     if ( n == 1 ) return 1
12     if ( n == 2 ) return 2
13     if ( n == 3 ) return 3
14     if ( n == 4 ) return 5
15
16     return fibonacci(n-1) + fibonacci(n-2)
17
18
```

```
9 def fibonacci( n ):
10     if ( n == 0 ) return 1
11     if ( n == 1 ) return 1
12     if ( n == 2 ) return 2
13     if ( n == 3 ) return 3
14     if ( n == 4 ) return 5
15
16     return fibonacci(n-1) + fibonacci(n-2)
17
18
```

저 세 lines을 주석 처리 하고 싶다!?

Cursor at the 12 line

|

# <ESC>

j | # <ESC>

j | # <ESC>

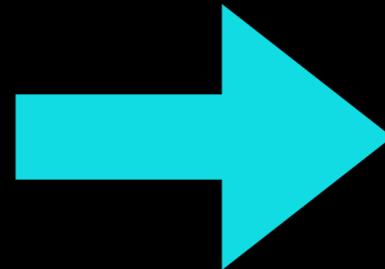
Cursor at the 12 line

|

# <ESC>

j.

j.



연속 된 구간을 주식 처리하려면

:`[range]s/^/#/g`

하나의 동작이 아니라 여러 동작을  
반복하고 싶다면 ?

매크로!

- $qa [action]^* q$ 
  - $[action]^*$  동안 했던 행동들을  $a$  라는 별명으로 저장
- $@a$ 
  - 별명이  $a$  인 매크로를 수행
- $\%d@a$ 
  - 별명이  $a$  인 매크로를  $\%d$ 회 수행

간단한 매크로 사용의 예를 볼까요?

한 줄에 하나씩 1 부터 100까지 입력



이 상태에서

qa yy p ctrl-a q

현재 줄을 복사해서 붙여 넣고 숫자를 1  
증가시키는 매크로!



이제 98@a 를 해보자

62 62  
63 63  
64 64  
65 65  
66 66  
67 67  
68 68  
69 69  
70 70  
71 71  
72 72  
73 73  
74 74  
75 75  
76 76  
77 77  
78 78  
79 79  
80 80  
81 81  
82 82  
83 83  
84 84  
85 85  
86 86  
87 87  
88 88  
89 89  
90 90  
91 91  
92 92  
93 93  
94 94  
95 95  
96 96  
97 97  
98 98  
99 99  
100 100

매크로 안에서 매크로를 호출 가능!

qa ..... @b ..... q

간단한 반복잡업을 하기 쉬움

익숙해지면 script를 코딩할 필요 없음

간단한 반복잡업을 하기 쉬움

익숙해지면 script를 코딩할 필요 없음

window split

- `sp[lit]` - 화면을 2행으로 분할
- `vs[split]` - 화면을 2열로 분할

1

```
1 #include <cassert>
2 #include <cmath>
3 #include <cstdio>
4 #include <cstring>
5 #include <string>
6 #include <vector>
7 #include <queue>
8 #include <map>
9 #include <set>
10 #include <algorithm>
11 #include <iostream>
12 #include <sstream>
13 using namespace std;
14
15 #define DEBUG(x) cerr << #x << " = " <
    < x << endl
16 #define DEBUGALL(x) { cerr << #x << "
    = "; for(const auto &e: x) cerr << e <
    < " "; cerr << endl; }
17
18
19 int n,m;
20 vector<int> a;
21
22 int main() {
23     cin.sync_with_stdio(false);
24     //freopen("input.txt", "rt", stdin
    );
25     //freopen("output.txt", "wt", stdo
    ut);
26
27     cin >> n;
28     cin >> m;
29
30     a.resize( n );
31     for(int i = 0; i < n; ++i)
32         cin >> a[i];
33
34     sort( a.begin(), a.end() );
```

0,0-1 All

1

- `sp[lit] filename`
- `vs[split] filename`

- ctrl - w(window) + @로 다양한 command 가 매핑되어 있음

- $\text{ctrl} - w v = :vs$
- $\text{ctrl} - w n|s = :sp$
- $\text{ctrl} - w < = :vertical\ res - 1$
- $\text{ctrl} - w > = :vertical\ res + 1$
- $\text{ctrl} - w - = :res - 1$
- $\text{ctrl} - w + = :res + 1$
- **ctrl - w =** = 같은 비율로 자동조정

- `ctrl - w [ h | j | k | l ]`  
window focus 이동
- `ctrl - w w`  
window focus 를 차례대로 이동  
(`alt + tab`의 그 것)

tab

```
~/.vimrc [No Name] [No Name] [No Name]
```

```
221
222
223 " ----- Key mappings
224 "
225 "
226 "the leader key
227 let mapleader="," " comma is the <Leader> key now.
228
229 map <C-_) <leader>ci
230
231 " Switch between window splits using big J or K and expand the split to its
232 " " full size.
233 " " Move vertically in the window through the horizontal splits...
234 nnoremap <C-J> <C-w>j
235 nnoremap <C-K> <C-w>k
236 " " Move horizontally in the window through the vertical splits...
237 nnoremap <C-H> <C-w>h
238 nnoremap <C-L> <C-w>l
239
240 " Tab navigations
241 nnoremap <C-t> :tabnew<CR>
242 "nnoremap <C-h> :tabprevious<CR>
243 "nnoremap <C-l> :tabnext<CR>
244 " " Tab alternative navigations
245 " " (terminals might capture <Ctrl-t> or something else)
246 nnoremap <leader>tn :tabnext<CR>
```

- `:tabnew` = 새로운 탭을 연다
- `:tabprev` = 이전 탭
- `:tabnext` = 다음 탭

Vimrc

vimrc 가 뭐하는 녀석인가?

vim + rc

rc = run command

- 간단하게 말하면 Configuration File
- 설정 파일이다
- 기본적으로 user의 홈디렉토리에 숨김파일  
형태로 존재( ~/.vimrc )

vim을 실행하면 자동으로 .vimrc파일의 내용을  
command mode 에 나 대신 입력해줌

한 번 눈으로 보자

```
$vim ~/.vimrc
```

```
1 "" Personal vimrc
2 "" Author :
3 "" Kim SangHoon <lucidash@gmail.com>
4 "" =====
5
6 set nocompatible
7
8 "" {{{{{{{{{{{{{{{{{
9 "" Arrow No map
10 map <leader>src :source ~/.vimrc<CR>
11 map <Up> <silent>
12 map <Down> <silent>
13 map <Left> <silent>
14 map <Right> <silent>
15 "" } } } } } } } } } }
16
17 ""
18 set rtp+=~/vim/bundle/vim-pathogen
19 execute pathogen#infect()
20
21
22 ""
23 set wildignore+=*/tmp/*,*.so,*.swp,*.zip
24
25 scripte utf-8
26 "set nocp all&
27
```

물론 여러분의 ... 흥미롭게는...

Empty File  
or

다른 누군가(보통 강 모 성배님)의

.vimrc File

이제 진짜 가장 먼저 할 일은



방향키를 “Do nothing” 키로

Mapping!

vim을 쓰다가

방향키에 손을 올리면

맞습니다

그럼 이제 뭘 설정하냐?

세미나 #1

command mode

:set ~~~~~

우리는 코딩을 할거니까...

- set nu[mber]
- set ru[uler]
- set ai(noai)
- set tabstop=%d
- set et(or noet)
- set hls[earch]
- set ls=2
- syntax enable

- `set paste ( nopaste )`  
vim 외부에서 복사한 거 붙여넣을 때 예상치 못한 부작용을 방지하기 위해 존재함
- `set mouse ( v | i | v | c | a ... )`  
마우스 사용을 감하 허락해주마

사실 이런 부분에서는  
몇 몇 필수요소를 제외하면  
[개인의 취향]

Key mapping

A 키를 B 하는 녀석으로 매핑!

- `[n|v|i|..]map A B`
- `[n|v|i|..]noremap A B`

vim 은 ( 꽤 ..? ) 자유롭게  
원하는 기능의 키를 원하는 키에 매핑시켜서  
(어느정도)내 마음대로 쓸 수 있다

아까 전에 봤던 window split



살다보면

이 따위 짓을 하는 날도 온다

포커스 이동을 위해 ...

ctrl w h ctrl w h ...

ctrl w w ctrl w w ctrl w w

ctrl ㅈㅈ

nnoremap <C-J> <C-w>j

nnoremap <C-K> <C-w>k

nnoremap <C-H> <C-w>h

nnoremap <C-L> <C-w>l

set mouse = [ni | a] 를 설정하면 ...

그냥 윈도우 클릭하면 포커스 이동 됨 ㅋ

또 다른 꿀 키 매핑

:noh

= :nohlsearch

```
1 "" Personal vimrc
2 "" Author :
3 "" Kim SangHoon <lucidash@gmail.com>
4 "" =====
5
6 set nocompatible
7
8 "" {{{{{{{{{{{{{
9 "" Arrow No map
10 map <leader>src :source ~/.vimrc<CR>
11 nnoremap <Up> <silent>
12 nnoremap <Down> <silent>
13 nnoremap <Left> <silent>
14 nnoremap <Right> <silent>
15
16 set mouse=a
17
18 map <CR> :noh<CR>
19 "" } } } } } } } } }
20
21 ""
22 set rtp+=~/vim/bundle/vim-pathogen
23 execute pathogen#infect()
24
25 ""
26 set wildignore+=*/tmp/*,*.so,*.swp,*.zip
27
28 scripte utf-8
29 "set nocp all&
30
31 " TERMINAL {{{ -----
32 colorscheme desert256
```

검색 결과 하이라이팅을 지워주는데

우리는 생각보다 검색을 자주한다

그래서 저 하이라이팅이 검색 할 땐 좋은데

끝나고나면 현자타임 옴

nnoremap <CR> :noh<CR>

nnoremap <CR> :noh<CR>

```
au FileType ruby map <F5> <ESC>:w<CR>:!ruby %<CR>
au FileType python map <F5> <ESC>:w<CR>:!python %<CR>
au FileType cpp map <F4> <ESC>:w<CR>:!g++ -O2 -std=gnu++11 % && ./a.out
au FileType cpp map <F5> <ESC>:w<CR>:!g++ -O2 -std=gnu++11 % && ./a.out < in <CR>
au FileType c map <F5> <ESC>:w<CR>:!gcc %<CR><CR>:!. /a.out < input.txt<CR>
imap <F2> <ESC>:w<CR>
map <F3> <ESC>:!cat %\|pbcopy<CR>
```

아쉽게도 쉘이나 ssh 클라이언트의  
privileged commad 는 못건드림 π

ex ) ctrl - z

그래도 leader key 를 주셨습니다

브멘

<leader> 로 쓰는 leader key

기본적으로 ‘\’ 키를 말함

사실 왜 있는지 정확하게는 모르겠는데  
아마 예약커맨드들 때문에 부족한 키조합을  
늘려주려는 의도는 아닐까?

덕분에

```
-          ~~~~~~  
9 "" Arrow No map  
0 map <leader>src :source ~/.vimrc<CR>  
1 nnoremap <Up> <silent>  
2 nnoremap <Down> <silent>
```

Plugins

- 원래는 ... text를 편집하려던 빔인데
- 근데 하다 보니 괜찮은데 뭐 더 있으면 좋을 것 같은 기능들이 있네?

그래서 만들었습니다(!?)

- 누가 ? 많은 사람들이
- 왜 ? 편해지려고
- 무엇으로 ? Python + Vim script +  
(c++, etc , sometimes )
- 어떻게 ? open source

- 설치하기

- Pathogen vs Vundle

- 둘 중 아무거나 골라 쓰면 됨!

- <https://github.com/tpope/vim-pathogen>

- <https://github.com/gmarik/Vundle.vim>

- 설치하기

- plugin들은 보통 깃허브저장소로 관리되고 있는 오픈소스 프로젝트들이 대다수.

- 원하는 기능을 검색만 잘 하면 괜찮은 플러그인들이 많고, 게 중에 맘에 안드는 기능이 있으면 내가 고쳐서 써도 됨ㅋ

Fuzzy file, buffer, mru, tag, etc finder. <http://kien.github.com/ctrlp.vim>

448 commits

3 branches

21 releases

13 contributors



branch: master ▾

ctrlp.vim / +



Don't check for CtrlPLinePre ...



kien authored on 29 Jul 2013

latest commit b5d3fe66a5

<a href="#">autoload</a>	Don't check for CtrlPLinePre	a year ago
<a href="#">doc</a>	Add a max results setting	a year ago
<a href="#">plugin</a>	Allow :CtrlPLine to take a bufname argument	a year ago
<a href="#">.gitignore</a>	Update .gitignore	2 years ago
<a href="#">readme.md</a>	Update readme.md	2 years ago

[readme.md](#)

# ctrlp.vim

Full path fuzzy **file, buffer, mru, tag, ...** finder for Vim.

- Written in pure Vimscript for MacVim, gVim and Vim 7.0+.
- Full support for Vim's regexp as search patterns.
- Built-in Most Recently Used (MRU) files monitoring.
- Built-in project's root finder.
- Open multiple files at once.
- Create new files and directories.
- [Extensible](#).

- Ctrl-P

```
~
requirements.txt
zabo/apps/registration/models.py
zabo/apps/account/models.py
zabo/apps/board/models.py
zabo/apps/club/models.py
zabo/apps/main/models.py
prt path <mru>={ files }=<buf> <->
>>> models_
```

- NERDTREE

```
.. (up a dir)
/Users/Lucidash/workspace/zabo/zabo/
├── apps/
│   ├── account/
│   ├── board/
│   └── club/
│       ├── __init__.py
│       ├── admin.py
│       ├── models.py
│       ├── tests.py
│       └── views.py
│   └── main/
│       ├── __init__.py
│       ├── admin.py
│       ├── models.py
│       ├── tests.py
│       └── views.py
│       └── registration/
│           └── __init__.py
├── media/
├── static/
│   ├── css/
│   └── image/
│       ├── calendar.png
│       ├── down.png
│       ├── edit.png
│       └── plus.png
│       └── js/
├── templates/
├── utils/
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
~
```



- nerdcommenter

주석처리를 쉽고 간편하게 도와주는 친구

당연히 filetype based comment 가능하고

구간 주석, 주석 토글 다 됨

- gundo

```

Gundo - (-/src/gundo.vim) - VIM
- Gundo for doc/gundo.txt [6]
- j/k - move between undo states
- <cr> - revert to that state

[13] 63 seconds ago
o [12] 68 seconds ago
o [11] 69 seconds ago
o [10] 72 seconds ago
o [9] 19 hours ago
o [8] 19 hours ago
o [7] 19 hours ago
o [6] 21 hours ago
o [5] 21 hours ago
o [4] 21 hours ago
o [3] 21 hours ago
o [2] 21 hours ago
o [1] 21 hours ago
o [0] Original

3 +gundo.txt+ Graph your undo tree so you can actually USE it.-
2 -
1 -
0 CURRENT STATUS: BETA~
1 -
2 IT MIGHT EAT YOUR DATA~
3 -
4 SERIOUSLY: IF YOU USE THIS PLUGIN, LOSE DATA AND COMPLAIN ABOUT IT~
5 I AM GOING TO MAKE FUN OF YOU ON TWITTER~
6 -
7 -
8 Making's Vim's undo tree usable by humans.-
9 -
10 =====
11 1. Intro +Gundo-plugin+ +Gundo+
12 -
13 You know that Vim lets you undo changes like any text editor. What you might-
14 not know is that it doesn't just keep a list of your changes -- it keeps-
15 a goddamned |:undo-tree| of them.-
16 -
17 Say you make a change (call it X), undo that change, and then make another-
18 change (call it Y). With most editors, change X is now gone forever. With Vim-
19 you can get it back.-
20 -
21 The problem is that trying to do this in the real world is painful. Vim gives-
22 you an |:undolist| command that shows you the leaves of the tree. Good luck-
23 finding the change you want in that list.-
24 -
25 Gundo is a plugin to make browsing this ridiculously powerful undo tree less-
26 painful.-
27 -
28 =====
29 2. Usage +GundoUsage+
30 -
31 We'll get to the technical details later, but if you're a human the first-
32 thing you need to do is add a mapping to your |:vimrc| to toggle the undo-
33 graph: >-
34 -
35 noremap <F5> :GundoToggle<CR>-
36 -
37 Change the mapped key to suit your taste. We'll stick with F5 because that's-
38 what the author uses.-
39 -
40 Now you can press F5 to toggle the undo graph and preview pane, which will-
41 look something like this: >-
42 -
43 Undo graph File-
44 -----
45 " Gundo for something.txt [1] one
46 " j/k - move between undo states two
47 " <cr> - revert to that state three
48 | five
49 |
50 | [5] 3 hours ago
51 | |
52 | | [4] 4 hours ago
53 | | |
54 | | | [3] 4 hours ago
55 | | | |
56 | | | | [2] 4 hours ago
57 | | | | |
58 | | | | | [1] 4 hours ago
59 | | | | | |
60 | | | | | | [0] Original
61 -----
62 --- 3 2010-10-12 06:27:35 PM
63 +++ 5 2010-10-12 07:38:37 PM
64 @ -1,3 +1,4
65 one
66 two

Gundo Preview 1,1 All doc/gundo.txt 4,48 Top

```





- 사실 이런 것 만으로도 충분히 [개발|코딩]을 하는데에 있어서 충분히 편리함을 제공가능
- 그래도 IDE의 생산성을 따라갈 순 없지 ㅎㅎ;
- 그래서 대단하신 분들께서 만들어 주신게 있으니 편하게 쓰자

- ultisnips| snipmates

<https://github.com/SirVer/ultisnips>

감상합시다

- jedi-vim

jedi-vim is a is a VIM binding to the autocompletion library [Jedi](#).

Here are some pictures:

```
1 from django.core import management
2 b = [management]
3 utility = b[0].ManagementUtility()
4 utility.main_help_text().l
~ ljust function: __builtin__.str.ljust
~ lower function: __builtin__.str.lower
~ lstrip function: __builtin__.str.lstrip
```

Completion for almost anything (Ctrl+Space).

```
1 strftime(format[, tuple]) -> string
2
3 Convert a time tuple to a string according to a format specification.
4 See the library reference manual for formatting codes. When the time tuple
5 is not present, current time as returned by localtime() is used.
[Scratch][none,unix][-][Preview] 1/6, 1
4 import time
5
6 class Foo(object):
7     global time
8     asdf = time
9
10 def asdfy():
11     return Foo
12
13 xorz = getattr(asdfy(), 'asdf')
14             (format, tuple=None)
15 xorz.strftime(
```

Display of function/class bodies, docstrings.

```
1 Docstring for parsing: def _remove_executions_from_set_vars@621
2 =====
3 Important mainly for associative arrays:
4
5 >>> a = 3
6 >>> b = {}
7 >>> b[a] = 3
8
9 `a` is in this case not a set_var, it is used to index the dict.
'__doc__'[none,unix][-][rst]
608         self.used_funcs = used_funcs
609         self.used_vars = used_vars
```

- YCM( You Complete Me )

<https://github.com/Valloric/YouCompleteMe>

Jedi 기반 python auto-complete  
C-family 언어 autu-complete  
C++ , C , C# ...

감상합시다

- 그 외에 javascript 를 위한 autocomplete plugin 은 물론
- ruby.vim 과 같은 ruby autocomplete plugin도 있고
- Java 를 위한 것도 있다!  
( 근데 자바는 웬만하면 이클립스Intellij 쓰자 )

( 사실 아주 솔직한 심정으로 )

이렇게 관찬은 플러그인이 많아도, 돈 받고 파는  
IDE 들에 비하면 많이 부족한 것은 사실이지만...

매우 저렴하고(ㄱ ㄴㅇㅈㅊ)  
가볍고(YCM같은 덩치큰 거 빼면...)  
빠른

Vim 많이 많이 쓰자!

모르는게 있으면 언제든지

:help

<http://vimdoc.sourceforge.net/html/doc/>