Vi,Vim활용

성시창

nu11 (at) kldp.org

도운이

송호영

hoyoungkuri (at) yahoo.co.kr

vi는 텍스트 문서를 편집하는데 사용되는 텍스트 편집기입니다. 많은 종류의 OS상에서 동작하는 vi가 개발되어 있으며, 특히 unix계열의 OS에서는 거의 모든 시스 템에서 기본 텍스트 에디터로 사용되고 있죠. vim은 Vi IMproved로써 말 그대로 vi에 unlimited undo, syntax coloring, split windows, visual selection, gui support 등의 기능이 보강된 vi clone입니다. vim에 관련된 자료나 각 OS별 바이너리, 소스는 www.vim.org에서 찾아보시기 바랍니다.

고친 과정

고침 0.2 고친이 jaurang

DocBook 변환

고침 0.1 1999-07-10 고친이 성시창

최초작성

1. 기본적인 편집,명령

vi는 크게 세가지 모드로 나뉘어집니다. 우선 명령모드로 키 입력이 바로 바로 명령이 되는 모드, 다음은 입력모드로 실제 문서를 편집하는 모드, 마지막으로 ex 모 드로 ex명령을 실행시키는 모드입니다. vi를 실행시키면 처음에명령모드 상태에 있게 됩니다. 명령모드에서 : 키를 누르면 ex모드로, a, i 등의 키를 누르면 입력모 드로 전환되어 문서를 편집할 수 있는 상태로 됩니다. 본격적인 내용으로 들어가기 전에 기본적인 편집에 대해서 설명을 하겠습니 다. 특정 파일을 편집하기 위해 열려면 vi [filename]을 shell prompt에서 실행시킵 니다. 이렇게 연 파일을 편집모드로 들어가 편집하려면 i를 입력, 원하는 내용을 입 력한 후 종료하려면 [ESC]ZZ 나 [ESC]:wq를 누르면 저장 후 종료하게 됩니다. 이 정 도는 기본적으로 알아두셔야 지금부터 하는 내용을 따라해 보실 수 있겠죠? 또 기본 적으로 알아두셔야 할 명령이 명령모드에서 u 즉 undo명령입니다. 그래야 잘못 실행 된 명령을 바로 undo할 수 있겠죠...^^

```
i 현재위치에서 삽입
I 현재줄의 처음 위치에서 삽입
```

```
a 현재 위치에서 추가
A 현재줄의 끝 에서 추가
o 새로운 한 줄을 커서 아래줄에 연다.
O 새로운 한 줄을 커서 위줄에 연다.
S 줄을 지우고 삽입모드로
R 현재 위치에서 Relpace 모드로
J 다음줄과 현재줄을 합친다.
~ 대문자<->소문자
. 마지막 명령을 반복한다
u (<-> CTRL-R) 마지막 수정한 것을 취소한다.
U 줄을 처음 상태로 복구한다.
```

1.1. 편집

1.1.1. 복사, 붙이기, 삭제

텍스트 편집기를 사용하면서 가장 유용한 기능 중에 하나가 반복되는 문자열을 복사해서 다른 곳에 붙이는 기능이라고 생각합니다. vi 역시 이러한 기능을 가지고 있고 vi의 명령어의 특성상 매우 많은 방법으로 이러한 명령을 수행할 수 있습니 다. 명령모드에서 yy라고 입력하면 현재 커서가 위치한 한 줄을 카피(Yanking)하게 됩니다. 또 dd는 현재 행을 삭제하게 되는데 이 역시 삭제가 이루어지면서 버퍼에 복 사가 되어있습니다. yy나 dd로 버퍼에 들어간 내용을 붙이려면 붙이고자 하는 곳 한 줄 위에서 p를 입력하면 됩니다. 짐작하시듯이 y키가 복사를 하는데 사용되는 키이고 d가 삭제에, p가 붙이기에 사용되는 키입니다. 명령모드에서 사용되는 명령들은 명령 앞에 숫자를 입력한 후 명령을 내리면 그 횟수만큼 명령을 반복하게 됩니다. 쉽게 2yy 라고 입력하면 2줄을 복사하리라는 것을 생각할 수 있습니다. 뒤에서 나오게 되는 이동에 관련된 명령과 조합하면 복사나 삭제를 원하는 영역에서 해 낼 수 있습니다. 우선 여기서는 복사에 사용되는 키가 y키, 삭제에 사용되는 키가 d, 붙이기에 사 용되는 키가 p키라는 것만 숙지하고 넘어갑시다. 참 한가지 빠뜨리고 넘어갈 뻔했군요. X 나 x키가 명령모드에서 삭제에 사용됩니다. 문자 위에서 x를 누르면 위치한 문 자가 삭제되고 X를 누르면 [bs]키처럼 동작을 합니다.

```
-copy-
cw 한 단어
2cw or c2w 두 단어
cc 한 행
c$ or C 커서 위치에서 행의 끝까지
cO 커서 위치에서 행의 처음까지
r 한 문자 변경
-delete-
dw 한 단어
2dw or d2w 두 단어
dd 한 행
d$ or D 커서 위치에서 행의 끝까지
dO 커서 위치에서 행의 처음까지
x or X 한 문자 변경
```

```
-yanking-
yw 한 단어
2yw or y2w 두 단어
yy 한 행
y$ or Y 커서 위치에서 행의 끝까지
y0 커서 위치에서 행의 처음까지
y1 or yh 한 문자 변경
```

1.1.2. Named 버퍼 사용

보통의 에디터의 경우에는 어떤 특정 문자열을 새로 복사하면 이전에 복사했 던 문자열이 지워지게 됩니다. vi는 a-z까지의 이름을 갖는 버퍼에 각각을 유지하여 복사 해 둘 수 있는 매우 편리한 기능을 제공합니다. 이러한 기능을 named buffer라 하고 사용방법은 모든 명령모드에서의 명령 앞에 a~z까지의 버퍼명 a ~ z을 붙여주 는 것입니다. 위에서 복사를 했던 것을 다시 해보면 ayy를 누르면 현재 줄이 a 버 퍼에 복사가 됩니다. 이것을 다시 붙이기를 하려면 ap를 입력하면 되겠지요. 이미 있는 버퍼에 내용을 추가하거나 처음부터 버퍼에 써 넣는 방법을 설명 하겠습니다. 이미 어떤 버퍼에 내용이 들어가 있고 그 뒤에 내용을 추가하고 싶은 경우가 생기죠. 이런 경우 추가는 버퍼 이름의 대문자를 사용합니다. Ayy와 같이 현재 줄을 버퍼에 추가하거나, ex모드에서 특정 문자열이 있는 줄을 모두 복사해두고 싶은 경우 :g/string/y A 이런 식으로 사용하는 것이죠. 복사등을 하지 않고 버퍼에 직접 써넣으려면 q명령을 사용합니다. q 다음에 버퍼명을 누르면 화면하단에 recording이라는 표시가 나오고 지금부터 하는 키입력 은 선택한 버퍼명에 들어가게 됩니다. 키입력이 끝나면 다시 q를 누르면 됩니다. 뒤에서 ex모드에서 사용하는 방법이나 @-function에 사용할 수 있는 방법을 설명하겠습니다. 여기서는 이 정도만 알고 넘어가도록 하죠. (주: vim에서는 register라고 하는군요.

1.1.3. 여러 문서의 편집

vi에서 여러 문서를 동시에 열기 위해서는 vi [filename1] [filename2]...의 형태로 열고자 하는 파일을 뒤에 이어서 써 줍니다. 그럼 동시에 여러 문서를 편집하 게 되죠. 그리고 각 문서 사이의 전환은 :n키와 :N을 이용해서 합니다. 즉 vi test1 test2로 두 파일을 열어서 작업을 하고 있다면 :n을 입력하면 test2로 편집이 넘어가 게 됩니다. 다시 :N 또는 :prev을 입력하면 test1을 편집할 수 있습니다. 또 현재 어 떤 파일을 편집중인지 확인하려면 :args를 입력하면 현재 열린 모든 파일 중에 현재 편집중인 파일이 대괄호로 둘러싸여 표시됩니다. 열린 파일이 많아서 여러 문서를 건 너뛰어 편집해야 하는 경우에는 n이나 N앞에 skip할 파일 개수를 써주면 됩니다.
:4n 이런 식이 되겠죠.

1.2. 이동

문서내용을 수정하려면 편집하고자 하는 위치로 먼저 이동하여야 편집이가 능하겠지요.
vi는 다른 편집기와는 비교도 되지 않을 만큼 많은 이동 방법을 제공합 니다. 이동명령은 다른

명령(복사, 삭제, 붙이기)등과 조합해서 사용하기 때문에 확 실히 알아두시는 것이 좋습니다. 제 나름대로 분류를 가까운 거리의 이동에서 원거리 이동까지 분류를 해 보았습니다.

1.2.1. 짧은 이동

가장 기본적인 이동이 명령모드 상에서 h, j, k, l키를 이용한 이동입니다. 차례로 좌, 하, 상, 우로의 이동을 나타냅니다. 역시 다른 명령모드 명령과 마찬가지 로 회수를 명령 앞에 넣어서 반복할 수 있습니다. 요즘은 대부분 커서 키를 지원하고 있지만 손가락이 오가는 거리 상 사용하지 않으시는 것이 좋습니다. 단어간의 이동은 w, b, e, E키로 합니다. w키는 다음 단어의 첫 번째 문자로 이동을 합니다. b는 반대로 앞 단어의 첫 번째 문자로의 이동을 나타냅니다. e, E는 첫 번째 문자로의 이동이 아닌 단어의 마지막 문자로의 이동입니다. 역시 회수를 앞 에 넣어서 몇 단어를 한 번이 이동할 수 있습니다. 이 밖에도 현재행 상에서 이동을 하는 명령들이 많이 있습니다. 0키를 누르 면 현재 행의 맨 앞으로 이동합니다. ^ 이나 | 역시 행의 맨 처음으로 이동합니다. 0 키는 행의 처음에 공백이 나오다 문자열이 나오는 경우에는 공백을 무시하고 맨 처음 나타나는 문자로 이동합니다. 하지만 ^ 이나 |는 0번째 열로 이동하게 됩니다. 특정 열로 이동하려면 [숫자]| 즉 10번째 열로 이동하려면 10|을 입력하면 10번째 열로 이 동을 하게 됩니다. 현재 행의 끝으로 이동하려면 $기호를 사용합니다. 이제 행간의 이동입니다. 한 행 밑으로 이동은 + 또는 [enter], 위로의 이동 은 - 키를 사용합니다. 10[enter]라고 입력해 보십시오. 엔터를 한 번입력하면 한 행밑으로 이동하지만 한 번에 10 행씩 팍팍 떨어지지 않습니까?
10+도 같은 결과를 보여줍니다.

```
h, j, k, l 좌, 하, 상, 우
+ or [enter] 다음줄의 첫번째 문자로
- 이전줄의 첫번째 문자로
e, E 단어의 끝으로
w, W 다음 단어로
b, B 이전 단어로
$ 행의 끝으로
0 행의 처음으로 ( ’A’ 인 경우 A앞으로 커서이동)
^ 행의 처음으로 ( ’A’ 인 경우에도 맨앞으로 커서이동)
), ( 다음, 이전 문장의 처음으로
}, { 다음, 이전문단의 처음으로
]], [[ 다음, 이전 구절의 처음으로
```

1.2.2. 원거리 이동

이제 조금 원거리 이동으로 넘어갑니다. 문서가 20000행 정도 된다고 합시 다. 처음에 문서를 열게 되면 커서는 1행에 위치합니다. 에러가 10000행 정도에 발생 한 소스코드를 수정한다고 치면, 어떻게 이동하면 좋을까요? 물론 뒤에서 파일 열기 에서 배우겠지만 현재까지 저희가 언급한 바로는 10000+를 하든지 해야겠지요. 물론 이렇게 해도 되겠지만 특정 행으로 직접 이동하는 명령이 있습니다. 바로 G명령입니 다. 10G라고 입력하면 10번째 행으로 이동하게 되는 것이죠. 다른 방법으로는 ex명령 에서 :10라고 입력하고 엔터를 입력하면 그 행으로 이동하게 됩니다. 맨 처음 줄로의 이동은 gg 나 1G 나 :1 등을 사용하고 마지막 줄로의 이동은 G

나 :$ 등이 가능합니 다. 여기서 $는 뒤에서 언급을 하겠지만 ex모드에서는 마지막 줄을 나타냅니다. 명령 모드에서는 행의 끝으로 이동에 사용되었죠.

```
^F 한 화면 앞으로 스크롤 (means CTRL-F)
^B 한 화면 뒤로 스크롤
^D 반 화면 앞으로 스크롤
^U 반 화면 뒤로 스크롤
^E 한 줄 앞으로 스크롤
^Y 한 줄 뒤로 스크롤
H 화면의 맨 위줄로 ( nH인 경우 맨 위에서 n행 밑으로 )
M 화면의 중간 줄로
L 화면의 맨 아래줄로 ( nL인 경우 맨 밑에서 n행 위로 )
```

1.2.3. 찾기로 이동

자세한 내용의 찾기는 3장에서 다를 것이고 여기서는 이동에 사용되는 찾기 명령에 대해서 설명하도록 하겠습니다. 현재 행에서 순방향으로의 특정 문자를 찾을 때는 f 키를 사용합니다. 현재 행에서 커서위치에서 첫 번째 나타나는 o 문자까지 이 동하고 싶다면 fo를 입력하면 됩니다. 반대 방향이라면 Fo를 입력하면 되겠죠. 이 명 령의 반복은 어떻게 할까요? 다른 명령처럼 앞에 반복회수를 명시해도 되지만 찾은 후에 다시 찾기를 같은 방향으로 하려면 키를 반대방향으로 하려면 , 키를 사용해 서 명령을 반복합니다. 한 행을 떠나서 순방향으로 어떤 패턴이 나타나는 곳으로 이동하려면 /pattern을 역시 반대 방향이라면 ?pattern을 입력하면 됩니다. 이 역시 같은 방향으 로 반복을 하려면 n, 반대 방향은 N키를 사용합니다.

```
/pattern 문자열의 처음에서 앞으로 검색
?pattern 문자열의 처음에서 뒤로 검색
n 검색을 다시 반복 (같은 방향)
N 검색을 다시 반복 (반대 방향)
fx 현재 줄에서 x가 있는 곳으로 이동 (앞으로)
Fx 현재 줄에서 x가 있는 곳으로 이동 (역방향)
tx n행 밑에서 x가 있는 곳으로 이동
Tx n행 위에서 x가 있는 곳으로 이동
; 줄에서 찾기를 같은 방향으로 반복
, 줄에서 찾기를 반대 방향으로 반복
nG, :n 행 이동 n번째 줄로 이동 (n이 생략되면 마지막줄로) --변환자 주: 이
기능은 :n 으로 하는것이 편합니다 :)
nl 열 이동 (n이 생략되면 처음 열로)
*  현재 커서가 위치한 단어 찾기 (앞방향)
# 현재 커서가 위치한 단어 찾기 (역방향)
```

1.2.4. 마크를 이용한 이동

책을 읽다 북마크를 하는것처럼 문서를 편집하다가 특정 부분을 마크할수 수 있으면 편리하겠지요. vi에서는 역시 알파벳 개수만큼의 마크를 하고 마크한 곳이나 그 줄로 바로 이동할 수

있습니다. 우선 마크를 하려면 ma ~ mz를 마크 할 곳에서 입 력합니다. 마크한 곳으로 이동은 두 가지가 가능합니다. 즉 마크한 줄과 열이 일치하 는 이동은 ‘a ~ ‘z 으로 마크한 줄의 처음으로 이동은 ’a ~ ’z를 이용합니다.

```
mx 현재 위치를 x 이름의 마크로 저장
`x 마크한 위치(행, 열)로 이동
'x 마크한 줄로 이동
`` 이전에 마크한 위치로 이동
'' 이전에 마크한 줄로 이동
```

1.3. 파일 저장, 열기, 종료

파일을 열 때 여러 가지 옵션을 주어서 열 수 있습니다. 즉 특정 행에 커서 가 위치하도록 열 수 있는데, 마지막 줄에 위치하려면 vi + [filename] 과 같이 열 고, n번째 행에 위치시키려면 vi +n [filename] 과 같은 방식으로 열면 됩니다. 특정 문자열이 있는 줄에 커서를 위치시키려면 vi +/pattern [filename]로 엽니다. vi +R [filename] 은 파일을 읽기 전용으로 엽니다. 파일의 저장을 살펴볼까요. 파일을 저장하면서 종료하는 방법은 맨 처음에 두 가지를 설명했었습니다. :wq와 ZZ입니다. 다른 저장방법이라면 :x 이 명령은 파 일이 변경되었을 때에는 저장하고 종료를 하지만 아니면 그냥 종료를 합니다. ZZ와 같은 기능이 같죠. 저장과 열 때 ex모드에서 사용하는 몇 가지 기호를 알아두시면 편 리합니다. 즉 %는 작성중인 파일명을 나타냅니다. 따라서 현재 파일을 편집 중에 백 업파일을 저장하고 싶다면 :w %.bak 이런 식으로 하시면 쉽게 백업을 만들 수 있습니 다. 그리고 #는 alternative 파일을 나타낸다고 하는데 이전에 열어서 작성한 파일을 나타냅니다. vi file1 으로 편집을 하다가 :e file2로 편집 파일을 새로 연 경우 다 시 먼저의 file1을 열려면 :e # 라고 하시면 됩니다.

```
vi [filename] file 열기
vi [filename1] [filename2 여러 파일 열기
view [filename] 또는 vi -R [filename]읽기 전용으로 열기
vi + [filename] 열고 마지막 행에 위치
vi +n [filename] 파일을 열고 n 번째 행에 위치
vi +/pattern [filename] 패턴이 나타나는 곳에 행 위치
ZZ 또는 :x 변경된 파일이면 저장하고 종료
:wq 저장하고 종료
w 파일의 저장
:<범위>w [filename] 특정 범위만 저장
:<범위>w >> [filename] 특정 범위를 다른파일에 덧붙임
```

1.4. 정리 및 연습문제

명령 중에서 대부분이 case가 다르면 방향만 반대로 작용하는 경우가 많습니 다. Ff, Oo, Pp, Tt, Nn, Xx등은 case에 따라서 방향이 반대인 경우입니다. 다른 문 자들도 case에 따라서 약간씩 의미가 달라졌음을 볼 수 있습니다. 이점에 유의해서 명령을 기억하면 쉽게 기억할 수 있

습니다. 자 위에서 배운 내용들을 조합해서 키 입력을 만들어 봅시다. 예문을 가지고 설명을 하도록 하겠습니다. 아래와 같은 문서를 편집하고 있고 현재 커서가 4번째 줄 의 sutff의 s에 위치해 있다고 합시다. 천천히 따라오세요. stuff 단어 하나만 지우 려면 dw를 입력하면 됩니다. 다음 단어인 goes까지 같이 지우려면 dw를 두 번 반복하 면 되겠지요? 2dw, 혹은 d2w를 입력해도 됩니다. 현재 커서 위치에서 2w 즉 두 단어 이동한 곳까지 지우는 것이죠. d다음이 이동명령이 오면 이동한 곳까지 지우는군 요... 그럼 우리가 배운 이동명령을 총 동원해 보도록 하죠. d$ 이면 $가 행의 끝까 지이니까 stuff부터 행 끝까지 지워지겠군요. 이 명령은 D를 입력해도 같은 결과를 나타냅니다. 이번엔 찾기로 이동과 조합해 봅시다. dfl을 입력하면 첫 번째 l이 나타나는 곳까지 커서가 이동하니까 l까지 지워지겠죠? d/bash를 입력해 봅시다. 6행에 bash앞까지 삭제가 되었나요? 자 이번엔 mark로도 이동이 가능하다고 했으니 mark를 이용해서 지워봅시다. 7행의 some의 s위에서 ma라고 합시다. 다시 stuff의 s로 돌아가서 d'a라고 입력해 보 세요. 생각했던 것처럼 나왔다면 다른 조합들을 생각해 보시길 바랍니다. 복사는 d대 신 y를 넣어주면 됩니다. 그럼 연습문제를 풀어보세요.

```
# /etc/bashrc
# System wide functions and aliases
# Environment stuff goes in /etc/profile

# For some unknown reason bash refuses to inherit
# PS1 in some circumstances that I can't figure out.
# Putting PS1 here ensures that it gets loaded every time.
PS1=[\u@\h \W]\\$
alias which=type -path
```

연습문제 2

문제 1) 현재 커서 위치가 unknown의 u에 있다고 하자. 단 각각 문제에 결과는 다음 문제에 영향을 미치지 않는다고 가정...

현재위치에서 그줄 끝까지 삭제, 복사
4줄 복사 후 마지막 줄 아래에서 삽입, 복사
현재 위치에서 figure out의 out앞 까지 삭제, 복사
현재 줄에서 마지막 줄까지 삭제, 복사
현재 위치에서 줄의 처음까지 삭제, 복사
현재 위치에서 25열까지만 삭제, 복사

문제 2)
1.위에서 3번째 행과 4번째 행을 합쳐라.

삭제 : d$, D 복사 : y$
4yyGp
삭제 : d/out 복사 : y/out
삭제 : dG, 6dd 복사 : yG, 6yy
삭제 : d0, d|, d1|, 3db, d3b, d?#, dF# 복사 : y0, y|, y1|, 3yb, y3b, y?#, dF#
삭제 : d25| 복사 : y25|

2. 두 줄을 합치는 명령을 표에서 찾아서 해보도록.

2. 찾기, 치환

vi에서 찾기 명령과 치환 명령은 거의 예술에 가깝습니다. 거의 모든 문자열 을 찾고 다른 문자열로 대체할 수 있는 듯 합니다. 그 만큼 명령에 기능도 많고 사용 하기도 조금은 복잡합니다. 찾기와 치환은 ex 모드에서 행해집니다. 우선은 정규 표 현식을 알아두시면 찾고자 하는 패턴을 쉽게 찾는데 많은 도움이 됩니다.

2.1. 정규표현식

정규표현식은 vim의 경우 :help라고 입력하면 나오는 정규표현식 기호가 설 명이 잘 되어있습니다. 정규표현식은 각자 공부하시길 바랍니다. 여기서는 매우 간단 하게 기호 설명과 간단한 예만 들고 넘어가겠습니다. 자세히 하면 이것만 가지고도 한참을 설명하여야 하니까요... 아래의 표에 복잡한 기호들이 많이 나와있는데요. 우선 ., ^, \+, *, [], \< \>의 기호는 반드시 아셔야 합니다. 정말 기본적인 기호들이고 몇 개는 shell상이나 다른 곳에서도 사용하실 수 있는 것들이니까 알아두시면 도움이 되실 겁니다. 이 기 호들을 간략히 설명하면 .는 하나의 모든 문자를 나타냅니다. .라고 하면 이 기호에 어떠한 한 문자가 대치될 수 있는 것이지요. ^은 대괄호 안과 밖에서 의미가 달라지 는 데 대괄호 안에서는 not의 의미를 가지고, 밖에서는 위에서 명령모드에서 ^기호의 용도와 비슷하게 행의 처음을 나타냅니다. \+는 1회 이상 나타난다는 것을 의미합니 다. *도 비슷한데 이것은 0번 이상 즉 나타날 수도 있고 그렇지 않을 수도 있습니다. 마지막으로 []은 나타날 한 문자의 범위를 지정할 때 사용됩니다. 즉 [abc]는 a 또는 b 또는 c입니다. [a-z]는 a에서 z중에서 한 문자를 나타내겠지요. 그럼 이 기호들을 가지고 찾고자 하는 문자열의 패턴을 만들어 봅시다. The 나 the를 기호로 나타내면 [Tt]he 가 됩니다. 맞죠? []가 정해진 범위에서 한 문자를 나타낸다고 했으니까요. 그럼 행의 맨 처음에 나타나는 The나 the를 찾으려면 어떻게 해야 할까요? ^[Tt]he가 되겠죠. 이번엔 th로 시작하는 아무 문자를 만들어 봅시다. th로 시작하니까 th를 앞에 써주고 뒤에 아무 문자나 나오든지 안나오든지 상관이 없죠? th.* 이렇게 검색을 하면 the, this, that 등등이 검색이 되지만 although등도 검색이 됩니다. th로 시작하는 문자만 검색을 하려면 단어의 시작과 끝을 명시해주어 야겠죠. 그래서 필요한 것이 \< \>입니다. 이 사이에 위의 표현식을 넣어서 검색을 하면 the, this, that등과 같이 th로 시작하는 문자열을 검색하게 됩니다. *대신에 \+를 넣으면 어떻게 다를까요? th 의 경우 *라면 해당되겠지만 \+라면 해당되지 않습 니다. 이상이 간단한 정규 표현식에 대한 기호이고 이제 본격적으로 검색과 치환을 해 보도록 하죠.

```
(이 안은 원래 문자를 나타내는 식)
., (\.) 아무 문자나 한 문자
^, (^) 행의 처음에 시작
$, ($) 행의 끝을 의미
\<, (\<) 단어의 시작
\>, (\>) 단어의 끝
[a-z], (\[a-z]) 주어진 범위 내에서의 한 문자
[^a-z], (\[^a-z]) 주어진 번위 밖에서의 한 문자
\e, (\e) Esc
\t, (\t) Tab
\r, (\r) CR
\b, (\b) BS
*, (\*) 0 번 혹은 1번 이상 나타남
```

```
\+, (\+) 적어도 한 번 이상 나타남
\l, (\l) 또는
```

POSIX character classes라는 것도 있는데 이것도 정규표현식 내에서 쓸 수 있습니다. 이러한 새로운 기호들은 모두 위에서 제가 설명한 것만으로도 만들 수 있 지만 식을 짧게 하기 위해서 미리 만들어 놓은 것이라고 생각하시면 됩니다. 예를 들 어 뒤에 확장 정규표현식에 \p를 정규표현식으로 하려면 [0-9a-zA-Z \t][0-9a-zA-Z \t]* 뭐 이런 식으로 표시해야겠죠(틀릴 수도 있습니다). 정규표현식에 자신 있는 분 은 모르겠지만 대부분은 많이 생각하여야 하고 저 처럼 혹시 빼먹은 문자가 생길지도 모르기 때문에 이런 식으로 또 다른 기호를 만들어 표현하는 것이죠.

```
[:alnum:] 알파벳, 숫자
[:alpha:] 알파벳
[:blank:] 공백문자 탭문자 포함
[:cntrl:] control 문자
[:digit:] 숫자
[:graph:] 공백문자를 제외한
[:lower:] 소문자
[:upper:] 대문자
[:print:] 프린트가 가능한 문자 공백 포함
[:punct:] 구두점문자 ) ( , . " ' : ; - 등
[:space:] 공백문자
[:xdigit:] 16진수 숫자
```

2.2. 찾기

단순히 패턴만 찾는 거라면 명령모드에서 /pattern 또는 ?pattern 과 n, N키 를 이용해서 찾는 것이 편리합니다. ex모드에서의 패턴을 찾기 위해서는 :/pattern/ 형식을 사용합니다. ex모드에 이 찾기 명령이 있음으로 해서 다른 ex명령들과 조합이 가능토록 하는 것 같습니다. ex모드가 행단위로 동작한다는 것은 알고 계시리라 생각 합니다. :/pattern/형식으로 입력을 하면 패턴이 있는 행으로 이동을 하지요. 이것이 어떻게 쓰일 수 있을지 생각해 보시기 바랍니다. ^^ 위에서처럼 찾게 되면 :/pattern/ 찾은 곳으로 이동만 하게 됩니다. 여기에 몇 가지 명령과 옵션을 덧붙여서 조금 복잡한 찾기를 할 수 있습니다. 간단한 것부터 예를 들죠. :=를 입력해 봅시다. 현재 커서가 위치한 행의 번호가 출력이 됩니다. :g/pattern/을 해 봅시다. 패턴이 나타난 줄의 내용을 모두 표시하면서 마지막에 나 타난 행으로 이동할 것입니다. 다시 :/pattern/= 이나 :g/pattern/=을 입력해 봅시 다. 앞에 것은 처음 pattern이 나타난 행 번호만 표시를 합니다. 두 번째 것은 pattern이 나타나는 행 번호를 모두 표시를 합니다.

2.3. 치환

제가 vi를 쓰면서 감동(TT)을 받는 것이 바로 이 치환 명령의 강력함입니다. 일반적인 치환문의 형식은 다른 ex명령들과 마찬가지로 [범위] /old/new/ [옵션] 의 형식을 사용합니다. 여기서 한가지 보충 설명을 하고 넘어가야 하는 것이 있습니다. hold buffer 라는 것인데 검색을 하면

서 검색된 문자열을 보관하는 일종의 기억장소입니다. 사용 법은 검색할 문자열을 \(\)로 둘러싸면 됩니다. 그럼 앞에서 순서대로 \1 ~ \9까지 의 버퍼에 찾은 문자열이 복사가 되지요. 비슷한 기호로 &가 있는데 이는 찾아낸 문 자열을 나타냅니다. 또 한가지 ~기호인데 이것은 바로 전에 수행한 검색이나 치환의 정규표현식을 나타냅니다. 예를 들어서 설명해 보도록 하죠. 우선 문서가 하나 필요하군요... 이전에 사용한 문서를 다시 사용하도록 하죠.

```
# /etc/bashrc

# System wide functions and aliases
# Environment stuff goes in /etc/profile

# For some unknown reason bash refuses to inherit
# PS1 in some circumstances that I can't figure out.
# Putting PS1 here ensures that it gets loaded every time.
PS1=[\u@\h \W]\\$

alias which=type -path
```

:%s/#/_&_/g라고 하면 #가 모두 _#_바뀌어 있을 것입니다. 찾는 문자열이 길 어지는 경우에 &를 사용하면 간단하겠지요? 이번엔 위에서 언급한 hold buffer를 사 용해 봅시다. :%s/\(goes\)\(in\)/\2 \1/g 이렇게 입력하면 어떻게 바뀔까요? goes in을 찾아서 각각 hold 버퍼 1, 2 에 저장을 해 둡니다. 뒤에서 치환할 때 버퍼 이름 이 순서가 바뀌어 있으므로 그 내용인 in goes를 출력하게 됩니다.

```
# Environment stuff goes in /etc/profile 명령 전
# Environment stuff in goes /etc/profile 명령 후
```

2.4. 정리 및 연습문제

몇 가지 설명을 하지 않은 것들이 있습니다. 위의 예제에서처럼 설정 파일들 은 문서 내에 /(역슬래쉬)기호들이 많이 있어서 패턴을 찾는데 쓰는 형식의 /과 혼 동될 수 있습니다. 물론 원래의 문자를 의미할 때는 \/로 모두 대치하면 되지만 이런 경우에는 헛갈리죠. /대신 기호나 :을 사용해서 각각을 구분 짓습니다. :%s:/etc/:/etc/local/:g를 하면 /etc/를 /etc/local/로 치환을 하게 되죠. 위에서 배운 것들의 조합으로도 많은 패턴을 찾고 수정을 할 수 있습니다. 예를 들어 보도록 하지요. 연습문제로 다음이 어떤 일을 할지 생각해 보시고 풀이를 보시기 바랍니다. 아래 (1)과 (2) 번은 위에서 언급을 하지 않았지만 해보시면 어떤 의미를 갖는지 아실 수 있으리라 생각합니다.

```
  연습문제 3. 다음 치환문의 의미를 설명하시죠.... ^^

:%s/.*/\U&/g
:%s/.*/\L&/g
:%s/\(.*\) *$/\1/
:%s/^ \+\(.*\)/\1/
:%s/^ \+\|^\t\+\(.*\)/\1/
```

```
:%s/^M//g (여기서 ^M은 CTRL-V후에 [ENTER]를 누른 값) 다음
명령의 의미를 설명하세요.
:n out.txt | g/pattern/, .w >> out.txt (out.txt화일이 존재해야
합니다.) 참고) :g/pattern/y A
:g/pattern/d
```

3. The power of ex

제가 생각하기에 vi가 강력한 에디터가 될 수 있었던 가장 큰 이유가 이 ex 모드 때문인 듯합니다. ex모드는 vi에 여러 가지 확장된 기능과 외부 프로그램을 사 용할 수 있는 통로를 제공합니다. 셀 명령은 기본이고, 유닉스의 기본적이고 유용한 유틸리티를 사용할 수 있습니다.

```
    ex 명령의 형식
:<범위> ex명령어 [| 명령 ...]
```

우선 ex명령은 행 단위로 명령이 실행된다고 알아두시기 바랍니다.(원래 ex 에디터가 라인 단위 에디터죠) 이렇게 이해하는 것이 가장 쉽습니다. 범위 설정도 물 론 행 단위로 설정이 되겠지요? 범위를 설정할 때는 ,로 구분해서 시작과 끝을 나타 냅니다. 즉 1, 10 이면 1~10행에 대해서 명령을 적용한다는 것입니다. 만약에 생략하 면 현재 위치한 행이 해당됩니다. 제가 위에서 찾기 명령이 어떻게 사용될지 생각해 보라고 했던 얘기가 기억이 나십니까? 범위 내에서 명령을 수행할 수도 있지만 어떤 찾기를 한 후에 그 줄에, 혹은 어떤 처음 줄을 주어주고 찾기가 성공한 줄까지 ex명 령을 수행할 수 있다는 얘기가 되는 거죠... 자세한 것은 역시 뒤에서 나오는 예에서 찾아보시기 바랍니다.

```
:d 삭제
:co, t 복사
:m 이동
:pu 붙여 넣기
:y 버퍼에 복사
:= 현재 행번호 표시(버전에 따라 틀림)
:p 줄의 내용을 표시
```

예를 들어가면서 설명을 하겠습니다. :1, $d (문서 전체의 삭제) $기호는 문 서의 끝을 나타냅니다. 정규표현식의 마지막에서는 줄의 끝을 나타내고 명령모드에서 도 줄의 끝을 나타내죠. $기호를 알아두시라 예를 들었습니다. :%d (문서 전체의 삭 제) %기호는 문서 전체를 나타냅니다. 위에서 쓴 식과 같은 일을 하지만 훨씬 짧아졌 죠. :.,+10y (현재 위치에서 10행 아래까지 버퍼에 복사) 버퍼에 복사만 합니다. :.,+10y|$|pu (현재 위치에서 10행 아래까지 버퍼에 복사, 마지막줄로 이동, 붙여넣 기) | 기호를 사용하면 한 행에 여러 개의 ex명령을 써줄 수 있습니다. :.m$ (현재 줄을 잘라낸 후 줄의 맨 마지막에 추가) :-, +y (현재 줄과 위, 아래 줄을 복사) +- 기호를 사용하는 방법을 설명하고 있습니다. 현재 행에서 -는 위쪽을 +는 아래쪽 행 을 의미하죠. :.,’xd (현재 줄에서 x에 마크한 줄까지 삭제) 마크를 통한 이동 중에 서도 ’X 로 이동하면 행단위로 이동하는 것을 위에서 설명을 했었죠. ex모드에서도 바로 마크를 이용할 수 있습니다. ”등의 기호도 당연히 사용이 가능합니다. 단 행단 위니까 ‘는 무효합니다. :., /pattern/d (현

재 줄에서 pattern이 처음 나타나는 줄까 지 삭제) 검색을 이용하여 범위를 지정하는 방법입니다. 단 이렇게 지우면 패턴이 나 타난 줄까지 삭제된다는 점에 유의하시기 바랍니다. :.y a (현재 줄을 named buffer a에 저장) named buffer도 역시 사용이 가능합니다. 복사 할 버퍼를 지정하여 yanking 을 하였습니다. 이를 불러낼 때에는 :pu a (버퍼 a에 복사된 것을 붙이기) 또는 ap로 하면 되겠죠... :.co$ (현재 줄을 복사해서 줄의 마지막에 붙이기) :/pattern/= (패턴이 나타난 줄 번호를 표시), :10+20d (10번째 줄에서 +20까지 삭제) 이는 현재 위치가 10 행 이외일 때 지정한 10줄에서 +20까지 지운다는 의미입니다. :10,+20d로 하면 뭐가 달라질까요? 이렇게 하면 10행에서 현재 행에서 20을 더한 줄까지 지운다 는 의미가 됩니다. 더 깊이 들어가시고 싶으신 분은 문서를 편집할 때 ex모드로 편집 을 해보시기 바랍니다. ex [filename]하시면 됩니다. 이렇게 해보시면 좀 갑갑하겠지 만 ex명령어를 확실히 익히실 수 있으리라 생각합니다.

3.1. abbreviations

이 기능도 잘 쓰면 매우 유용한 기능입니다. 긴 문자열을 약어 입력으로 얻 어내는 것이죠. 다른 방법으로는 자주 타이핑되는 오타와 정타를 묶어서 자동 수정 기능을 하게 하는 것입니다. 역시 예를 들어 설명을 하죠. :ab univ. university 라 고 해 둡시다. 편집모드에서 univ.라고 입력하고 white character를 입력하는 순간 univ.가 university로 바뀔 것입니다. 내가 the를 teh라고 자주 오타를 낸다면 :ab teh the를 등록해 두면 되겠죠.

3.2. map, map!

이 기능도 매우 유용합니다. 매크로를 만들어 쓸 수 있도록 도와줍니다. 우 선 특수키의 입력에 대해서 알고 넘어갑시다. ex에서 [ctrl]-v를 누르고 [enter]를 누르면 ^M이라고 표시될 것입니다. 이 문자가 [enter]를 나타냅니다. 비슷하게 [ctrl]-v를 누르고 특수키를 누르면 해당하는 문자가 표시됩니다. map으로 정의해 놓 은 키 입력은 명령모드에서 동작을 하고 map!로 만들어 놓으면 편집모드에서 사용할 수 있게 됩니다. [esc]키가 너무 멀어서 누르기 힘드신 분은 :map! ^H ^[로 해 보시 기 바랍니다. [bs]키를 누르게 되면 편집모드에서 명령모드로 바뀌게 됩니다. 자주 사용하지 않는 키를 매핑해서 사용할 수 있겠지요. 처음 시작만 명령어 모드에서 사 용하고 편집모드에서 사용한다는 의미가 있을 뿐 키 입력을 그대로 대신해 주므로 실제로는 두 모드간을 자유롭게 넘나들 수 있습니다. 즉 :map & I/*^[A*/^[이렇게 해 놓으면 &키를 누르면 현재 줄을 C문의 주석으로 만들어 버립니다. 이 명령에서는 삽 입과 명령 모드를 두 번이나 넘나들었죠. Function키에 대해서도 이러한 매핑하는 것 이 가능한데 차례로 #1 ~ #9 번이 function키를 나타냅니다. :map #1 commands 이런 형식입니다. 아니면 위에서 했던 것처럼 [ctrl]-v 다음에 function키를 눌러서 입력 을 받아도 됩니다. 아니면 [F1] ~ [F9]를 대신 써주어도 되고 (이

에 값은 /etc/termcap 에 정의되어야 합니다.)

3.3. @-function

Named buffer를 이용한 macro입니다. 개념은 매우 간단한 것이죠. insert모 드에서 매크로로 사용할 키 입력을 그대로 친 후에 add ayy등의 형식으로 named buffer a에 저장을 합니다. 이 매

크로를 불러서 쓸 때 @를 앞에 쓰고 뒤에 버퍼명을 씁니다. 즉 @a라고 치게 되면 아까 만들어서 a버퍼에 넣었던 키 입력이 그대로 반복 되는 것이죠. @@라면 마지막에 버퍼에 들어간 것을 실행하게 됩니다. 허 이건 ex명령 과 별 관계가 없는 글이군요. 쩝~ 수정하려면 차례도 수정하여야 하는 ... --

3.4. shell command

우선 다른 에디터에서처럼 shell모드로 빠져나가려면 :sh 라고 입력하면 됩 니다. 다시 복귀하려면 [ctrl]-d를 누르면 다시 vi로 되돌아 오게 됩니다. 하지만 vi 는 shell로 나가지 않고 ex에서 바로 shell상의 명령을 실행 시킬 수 있습니다. 바로 ! 기호를 이용합니다. !gcc -c % 이런 식도 가능하겠죠. :r !ls라고 쳐 봅시다. ls가 실행되고 그 결과가 지금 위치에 덧붙여집니다.

```
total 18
drwxrwxr-x 2 nu11 nu11 1024 May 25 22:01 .
drwx------ 11 nu11 nu11 2048 May 25 22:01 ..
-rw------- 1 nu11 nu11 12288 May 25 22:01 .vi-beginner.txt.swp
-rw-rw-r-- 1 nu11 nu11 14 May 25 21:59 test
-rw-rw-r-- 1 nu11 nu11 1583 May 25 22:01 vi-beginner.txt
```

같은 방법으로 자체적인 정렬을 하는 명령은 없지만 sort라는 강력한 소트 프로그램을 이용할 수 있습니다. sort에 대한 자세한 내용은 다른 책을 참고 하시길 바라고, 다음과 같이 간단한 예만 들어보이도록 하겠습니다. 위의 내용을 크기에 따 라서 정렬을 해 볼까요?

```
:2,$ !sort +4

total 18
-rw-rw-r-- 1 nu11 nu11 14 May 25 21:59 test
drwxrwxr-x 2 nu11 nu11 1024 May 25 22:01 .
-rw-rw-r-- 1 nu11 nu11 1583 May 25 22:01 vi-beginner.txt
drwx------ 11 nu11 nu11 2048 May 25 22:01 ..
-rw------- 1 nu11 nu11 12288 May 25 22:01 .vi-beginner.txt.swp
```

자 이제 많고 많은 유닉스 셀 명령어들을 모두 사용할 수 있습니다. 무엇을 더 할 수 있을지는 ... ^^ 4.5 set, 자신에 맞는 vi 만들기 자신의 작업환경에 맞게 vi를 세팅하는 법입니다. 우선 set할 수 있는 목록 은 set all하면 주루룩 스크롤이 되면서 올라갑니다. 하나씩 설명하면 지금까지 써온 것 만큼이나 써야 할 것 같네요. ^^ 시간이 나면 나중에 추가하도록 하죠... set nu 화면의 좌측에 행번호를 출력합니다. set ruler 커서의 위치가 우측 하단에 표시됩니다. set aw , set ai ...저는 위에 것들을 자주 사용하는군요. 한가지만 언급하고 넘어가죠. 저도 처음에 몰라서 고생 좀 했죠. set aw 등 으로 세팅을 했다면 이에 대한 해제는 set noaw 이런 식으로 앞에 no만 붙이면 된다 는 겁니다. -- (자우랑 : 저도 이 기능을 상당히 좋아합니다 :))

3.5. 연습문제

(1) 현재 커서가 위치한 라인을 주석으로 만드는 @-function을 만들어라.

4. vim에 추가된 기능들

4.1. multi-window

vim은 멀티 윈도우 기능을 지원합니다. 이 기능은 창을 분할해서 하나의 vi 에 여러 개의 파일을 보면서 편집할 수 있도록 해 주는 기능입니다. :sp 로 창을 분 할하거나 [ctrl]-wn 으로 창을 새로 만들 수 있습니다. 창간의 이동은 command 모드 에서 [ctrl]-ww나 [ctrl]-wk, [ctrl]-wj 등으로 할 수 있습니다. 직접 해보시는 편이 이해가 쉬우리라는 생각이 드네요. 창을 닫을 때는 각 창에서 파일을 닫아도 되고 [ctrl]-wc 로 닫을 수 있습니다. 저는 개/인/적/으로는 창 기능을 거의 쓰지 않습니 다. 가뜩이나 적은 모니터에 창까지 분할하면 볼 수 있는 내용이 줄어드니까요. ^^ 물론 :resize 등의 명령이나 [ctrl]-w+ [ctrl]-w- [ctrl]-w=등의 명령으로 창의 크기 의 조절이 가능합니다. 이러면 좀 쓸만합니다.

4.2. 확장 정규표현식

아래 내용은 vim 패키지에 포함된 정규표현식에 사용되는 기호들입니다. 이 것들이 다 필요한지는 의심스럽군요. 위에서도 잠깐 말했듯이 자주 사용되는 정규식 들을 기호화 해 놓았을 뿐입니다.

```
  Character classes {not in Vi}:
\i \i isident=@,48-57,-,192-255 에 해당하는 글자들입니다.
set isident 명령으로 확인 및 정의할 수 있습니다.
\I \I \i와 같지만 숫자는 제외
\k \k keyword로 사용하는 문자, set iskeyword 참고
\K \K \k와 같지만 숫자제외
\f \f 파일이름으로 사용하는 문자, set isfname참고
\F \F \f와 같지만 숫자 제외
\p \p 프린트 가능한 문자, set isprint참고
\P \P \p와 같지만, 숫자 제외
\s \s whitespace character: [Space] and [Tab]
\S \S non-whitespace character opposite of \s
\d \d digit: [0-9]
\D \D non-digit: [^0-9]
\x \x hex digit: [0-9A-Fa-f]
\X \X non-hex digit: [^0-9A-Fa-f]
\o \o octal digit: [0-7]
\O \O non-octal digit: [^0-7]
\w \w word character: [0-9A-Za-z-]
\W \W non-word character: [^0-9A-Za-z-]
\h \h head of word character: [A-Za-z-]
\H \H non-head of word character: [^A-Za-z-]
\a \a alphabetic character: [A-Za-z]
\A \A non-alphabetic character: [^A-Za-z]
\l \l lowercase character: [a-z]
\L \L non-lowercase character: [^a-z]
```

```
\u \u uppercase character: [A-Z]
\U \U non-uppercase character [^A-Z]
\e \e [ESC]
\t \t [tab]
\r \r [CR]
\b \b [BS]
\n \n matches [NL] Not available yet! Will be used
for multi-line patterns
~ \~ matches the last given substitute string
```

4.3. visual mode

vim에서 [ctrl]-v 를 누른 채로 이동을 하면 반전된 영역이 생기는 것을 볼 수 있습니다. 처음에 [ctrl]-v를 누르면 block으로 영역이 생깁니다. 이때 v를 입력 하면 영역이 문자 단위로 잡히고, V를 누르면 라인단위로 영역이 생깁니다.

```
선택된 영역이 생겼을 때 쓸 수 있는 명령들
~ 대문자 <-> 소문자 전환
d 삭제
c 치환
y 버퍼에 복사
> set shiftwidth로 정의해 놓은 만큼 우로 이동
< set shiftwidth로 정의해 놓은 만큼 좌로 이동
: 선택한 영역에 대해서 ex 명령을 수행
J 줄을 합친다.
U 대문자로 만든다.
u 소문자로 만든다.
^] tag를 찾는다.
```

대문자 <-> 소문자 전환 삭제 치환 버퍼에 복사 set shiftwidth로 정의해 놓은 만큼 우로 이동 set shiftwidth로 정의해 놓은 만큼 좌로 이동 선택한 영역에 대해서 ex 명령을 수행 줄을 합친다. 대문자로 만든다. 소문자로 만든다. tag를 찾는다.

4.4. tag stack

tag란 한 마디로 설명하자면 tags 파일에 존재하는 직접 이동을 위한 표시입 니다. (내용을 보면 이것도 매크로입니다.--) 프로그램을 작성하다보면 함수의 내용 을 알고 싶은 경우가 생기죠. 이런 때 그 함수로 직접 이동을 했다가 다시 편집하던 곳으로 이동을 할 수 있도록 해 줍니다. 다음과 같은 프로그램이 있다고 합시다.

```
main.c

void main(){
hello()
bye()
```

```
}

my.h

void hello(){
puts(Hello, vi user)
}
void bye(){
puts(Good bye)
}
```

자 이제 ctags로 tags 파일을 만듭니다. 다음과 같이 하면 되겠지요. :!ctags *.[ch] 혹은 다른 셸상에서 [nu11@foo vi-test]$ ctags *.[ch] 이제 tag stack의 기능을 활용해 봅시다. vi main.c로 main.c를 열어봅시다. hello() 앞으로 이동을 합니다. 어떻게 갈지는 알아서들 하시길... 이 위치에서 [ctrl]] 키를 누릅니다. (^]입니다. 컨트롤키를 누른 상태에서]) 화면이 바뀌면서 void hello(){ 의 맨 앞에 hello가 위치하게 되죠. 함수의 내용을 확인하였으면 다시 [ctrl]t를 누 릅니다. 다시 원래 위치로 복귀를 합니다. 다시 bye()가 있는 줄에서 [ctrl]]를 누르 면 다시 my.h가 열리면서 void bye()줄에 커서가 위치합니다. 다시 복귀는 [ctrl]t. vim의 HELP에도 tags화일이 존재합니다. :help로 help 문서를 보다보면 ||로 둘러싸인 것들이 있는데 이 위에서 ^]를 누르면 HYPER TEXT처럼 그 파일로 이동을 합 니다. 다시 ^t를 누르면 복귀구요.

4.5. 프로그래밍을 위한 명령들

vim에는 tag stack이외에도 프로그래밍에 편리한 기능을 제공합니다. 바로 컴파일 후에 에러가 난 곳으로 바로 이동을 할 수 있도록 해 주는 기능입니다. 우선 컴파일시에 만들어진 error 파일을 저장해야 합니다. 만약에 Makefile이 만들어져 있 다면 :make 명령 한 번만 내리면 자동으로 errorfile이 만들어지고 에러가 발생한 행 으로 이동을 합니다. :cl을 입력하면 에러 리스트가 나오고 :cn을 입력하면 에러가 난 다음 행으로 자동으로 이동을 합니다. :cN 은 반대 방향으로 에러가 난 행으로 이 동을 합니다. 여러 파일로 이루어진 소스의 경우에도 다 열어가면서 들어갑니다.

```
main.c

void added(){
This makes error
}
void main(){
hello()
bye()
added()
}

my.h


void hello(){
```

```
this also makes error
puts(Hello, vi user)
}
void bye(){
puts(Good bye)
}

Makefile

all : main.c my.h
gcc -o test main.c my.h
clean : test
rm -f test
```

이렇게 만들어 놓고 vi main.c에서 :make를 해 봅시다. 에러가 한 번 출력된 후에 my.h의 this also makes error 라인에 커서가 위치하고 vi의 맨 아래줄에는 (4 of 12): ‘this’ undeclared (first use this function) 라고 출력이 될 껍니다. :cn을 해 봅시다. (5 of 12): (Each undeclared identifier is reported only once로 메 시지가 바뀌었지요. 몇 번 :cn을 하다보면 다시 main.c의 This makes error 라인에 커서가 위치하죠. 몇 번 해보시면 충분히 이해하리라 생각합니다. Makefile 파일을 만들지 않고 할 수 있는 방법은 없는가? 분명히 있겠죠. 단순히 컴파일 에러 메시지가 들어 있는 파일만 있으면 동작을 하니까요. 해보도록 하 죠. :!gcc -c main.c >& err.out 이렇게 하면 error 메시지가 err.out에 저장이 되겠 지요. 다음에 할 일은 파일을 지정해 주는 것입니다. :cf err.out 자 이제 :make했던 것과 같이 :cn을 하면서 디버깅을 할 수 있습니다.

5. 기타 팁

- 윈도우키 매핑하기 - xmodmap 이란 명령어를 이용해서 106키보드에서 사용하지 않는 windows키를 [esc]키 대용으로 사용하는 방법입니다. xmodmap -e keycode 115=Escape
- bash에서 vi모드로 사용하기- bash를 사용하고 계시다면 command line 편집기가 vi의 숙적인 emacs모드로 세팅되어 있습니다. 명령을 하나 치고 [ctrl]-a를 누르면 커서가 명령의 맨 처음으로 이동을 하지요? 그럼 emacs모드 맞습니다. set -o vi 로 당장 vi 모드로 바꿔주시기 바랍니다. ^^ 자 이제 command line상에서 키 이동은 지금까지 배워왔던 키 입력과 같습니다.

  [nu11@foo nu11]$ cd /usr/X11R6/lib/X11/ko/app-defaults/-

  상태라고 합시 다. 이걸 [nu11@foo nu11]$ du /usr/X11R6/lib/X11/ko/app-defaults/ 로 바 꾸려면 [esc]0cwdu를 차례로 누르면 되겠죠. 이전에 사용했던 명령을 한 번 찾아볼까요? [esc]를 누르고 /cut 라고 해 보 세요. 이전에 사용했던 cut란 명령어가 현재 줄에 나타나네요. [nu11@foo nu11]$ ls -l *.EXE |grep ^- |cut -c33-41 이게 아니라고요? n키를 누르면 검색을 계속 하겠죠? [nu11@foo nu11]$ ls -l *.EXE |grep ^- |cut -c35- 가 나오는군요. 특정 파일이 들어가 있는 명령을 찾을 때도 그렇고 command mode에서 사용했던 명령을 모두 검색할 수 있습니다. 사실은 .bash-history 파일을 검색하는 것이겠죠. [esc]/doc 라고 하니까 저는 cd doc가

가장 먼저 나타나는군요. 행이동 명령들도 거의 가능합니다. 계속 검색은 물론 n, N이겠죠. [esc]11G 등을 누 르면 .bash-history 파일에 들어가 있는 11번째 항목으로 이동이 되는 것을 볼 수 있 습니다.

- tcsh & csh에서 vi모드로 사용하기 - 프롬프트 상에서 다음과 같이 입력하면 됩니다. bindkey -v 다시 emacs모드로 사용하시려면 bindkey -e