Emacs 소개

이 휘재hjlee@wisenet.co.kr

처음 Emacs 를 접한 것은 실습실에서 C프로그램을 짜기 시작할때였습니다. 처음 vi를 배우고 평소 DOS를 사용할 때 쓰던 에디터와 사용법이 달라 애를 먹고있던 차에 그나마 평소 쓰던 에디터와 비슷하고 장안에 칭찬이 자자한 이 에디터를 소개 받았던 것입니다. 처음 Emacs 를 쓸 때 Emacs 에서 빠져나오는 방법을 몰라 한참 헤매던 생각을 하면 웃음이 납니다. 그 당시 저는 단지 글자를 삽입하고 지우고 저장하고 하는 정도의 기능만 사용했었습니다. Linux를 설치하고 이제 좀 익숙해 질 무렵, 저는 Emacs 를 잘 배워보고 싶은 생각이 들었습니다. 그리하여 구입한 책이 "Learning GNU Emacs"라는 책입니다. 그 책을 보면서 저는 사람들이 좀더 쉽게 Emacs 를 배울 수 있었으면 좋겠다라는 생각을 했습니다. 그리고, 한글로 정리해 두고 싶은 욕심이 생겼습니다. 전 이 글이 Emacs 를 처음 배우는 사람들과 또한 저 자신에게도 도움이 되기를 바랍니다. 필자는 이 글을 Emacs 를 아직 잘 모르는 사람들을 대상으로 쓸 것이다. 또한, Linux 사용자들을 위한 내용들도 추가 될 것이다. 이 글은 "Learning GNU Emacs" 라는 책의 내용을 많이(거의 대부분) 베꼈다. 만일 영어에 거부감이 별로 없다면, 이 글을 읽기보다는 이 책을 사서 보는 것이 훨씬 도움이 되리라 생각한다. 혹, 영어에 거부감이 있더라도, 사봐서 손해나지는 않을 것이다.

목 록

1 Emacs 소개

GNU Emacs는 오늘날, UNIX 세계에서 가장 널리 사용되는 text 에디터중의 하나이다. 그것은 별로 이쁘지도 않고, 새로운 것도 아니다. 그럼에도 불구하고 많은 사람들이 이 에디터를 좋아하는 이유는 그것이 매우 유용한 tool이라고 생각하기 때문일 것이다. 유용한 기능을 발견할 때마다, 전에 다루던 에디터들에서 만나보지 못한 개념들을 만날 때마다, 필자는 이 에디터의 help문이 아직 모두 영어로 씌여져 있다는 데 대해 한탄을 느낀다. '이 help문들이 한글로 되어 있었거나 영어를 한글 읽듯이 읽을 만한 능력이 나에게 있었다면 진작에 알았을 텐데... 이 help문들이 한글화 되어 진다면 전국의 영어에 거부감을 느끼는 많은 UNIX 사용자들, 프로그래머들, 에디터 사용자, 개발자들, 등등등 에게 한줄기 빛이 될텐데...'

Emacs의 강점은 무엇보다도, 그 유연성과 확장성에 있다고 할 수 있다. 만일 그렇지 않더라도 그렇게 말할 수 있다. 필자는 여러분이 Emacs에 대해 알아갈수록 놀라움을 느끼게 되리라 생각한다.

1.1 화일과 버퍼(buffers)의 이해

(여기서의 버퍼는 우리가 버퍼하면 떠오르는 것 하고 다르다는 사람이 있었다.) 에디터들은 실질적인 화일을 편집하지는 않는다. 대신 화일의 내용을 임시 버퍼에 넣고 그것을 편집한다. 기억하라. 버퍼는 화일과 아주 비슷하게 보일지라도 단지 화일의 복사본만을 보관하는 임시 작업영역일 뿐이다.

화일처럼, 버퍼도 이름을 가지고 있다. 버퍼의 이름은 일반적으로 편집하고 있는 화일의 이름과 같다. 여기에는 몇가지 예외가 있다. 그러나 여기서 중요한 것은 버퍼는 화일의 복제라는 점이다. 버퍼를 수정하고 그 수정이 마음에 들면 화일에 저장할 수 있다. 만일 수정이 마음에 들지 않는다면 화일을 저장하지 않고 Emacs 를 빠져나올 수 있다.

1.2 모드

Emacs 는 다양한 편집모드를 가지고 있다. 이것이 의미하는 것은 하고자 하는 작업에 따라 알맞게 행동한다는 것이다. Emacs는 여러가지 다른작업에 따라 달리 행동하는 에디터가 된다. 모드는 주모드(majar modes)와 부모드(minor modes)가 있다. 버퍼는 한번에 한가지 주모드에 속할 수 있다.

모드	기능
`Fundamantal mode`	일반적인 모드, 특별한 것은 없다.
`Text mode`	글을 입력할 때.
`Indented text mode`	만입이 있는 글 입력.
`Picture mode`	간단한 선 그림을 그릴 때.
`C mode`	C 프로그램을 작성할 때.
`FORTRAN mode`	FORTRAN 프로그램을 작성할 때.

Emacs LISP mode	Emacs LISP 함수를 만들 때.
LISP mode	LISP 프로그램을 작성할 때.
LISP interaction mode	LISP 표현들을 검토하고 작성할 때.
nroff mode	nroff용 화일을 구성할 때.
TeX mode	TeX용 화일을 구성할 때.
LaTeX mode	

부모드는 Emacs의 행동에 특별한 양상을 추가한다.

모드	기능
Abbrev mode	약자의 사용을 가능하게 해준다.
Fill mode	word wrap을 가능하게 해준다.
Overwrite mode	글자를 삽입하는대신 수정한다.
Auto-save mode	화일을 지정된 만큼 자주 자동저장한다.
...	

만일, 특별한 목적을 위한 모드가 필요하다면, 여러분은 그 모드를 프로그램할 수 있다. (Emacs가 행동하는 방식을 조정할 수 있다.) 만일, 그 모드를 필요로하는 사람이 많다면, 벌써 누군가 멋지게 만들어 놓은 것을 구할 수 있을지도 모른다. (일례로 필자는 c++-mode를 구해서 쓰고 있다.)

2 일단 써보기

2.1 Emacs 가동

Emacs를 가동하려면 emacs 라고 치고 뒤에 편집하고자하는 화일의 이름을 치면된다. 만일 존재하지 않는 화일이름을 사용하면 Emacs는 새로운 커서는 “point” 또는 “dot”라고도 불린다. Emacs의 온라인 도움말 기능은 이 단어를 사용한다. 많은 편집기 들과는 달리(대표적으로 vi) Emacs는 글자를 삽입하고 명령을 내리는 모드를 구분하지 않는다. 화면의 맨 아랫줄은 “미니버퍼”로 사용되고 아래에서 두번째 줄은 여러가지 정보를 표시해 준다. 이 줄은 “모드라인” 이라고 불린다. 맨 왼쪽에 두개의 별표(**)가 있다면, 버퍼가 수정 되었다는 것을 나타내 준다. (이 위치에 %%가 있으면 읽기 전용 버퍼임을 나타낸다.) Emacs:뒤에 버퍼(혹은 화일)의 이름이 나온다. 괄호 안에 현재의 모드가 나오고, 그 뒤에 화면에 보이는 부분이 전체중 어디쯤인지를 표시해 준다. 버퍼의 처음일 경우 'Top', 끝일 경우 'Bot' 그렇지 않을 경우에는 %로 표시해 주고, 버퍼의 내용 전체가 화면에 보일 경우에는 'ALL'이라고 표시해 준다.

Emacs의 숙련자가 되면, 때때로 여러개의 버퍼를 같이 열어 놓고 작업을 할 때도 있을 것이다. 이러한 경우에도 각각의 버퍼는 각각의 모드라인을 가지게 된다.

2.2 Emacs 명령어들

Emacs 는 몇천 아니면 몇백개의 명령어들을 가지고 있다. 물론 독자들로 예상하고 있겠지만 binding이 이루어지지 않은 명령어 들이나, 어떤 키에 binding이 이루어져 있는지 모르는 경우에는 M-x “명령어 이름” RETURN을 치면 된다.

Emacs 는 사용자가 자기만의 binding을 가지거나, 기존의 binding을 바꿀 수 있도록 해준다. 자세한 것은 “Learning GNU Emacs” 9장을 참고하거나, Emacs Help를 참고하기 바란다. (아니면, 이 글이 혹시 그 내용을 다루게 될지도 모르므로 그 때까지 기다리면 된다.)

2.3 기본 명령들

자 이제 가장 기본적인 명령어들에 대해 알아보자. 여기서 부터는 명령어 요약을 먼저 제시하고 보충설명이 필요한 부분은 뒤에 설명하는 방식으로 진행하겠다. 명령어 요약은 명령어 단축키를 먼저 제시하고 난 후 명령어 이름을 써 놓았다.

2.3.1 C-x C-f (find-file)

같은 이름의 버퍼를 만들고, 화일을 읽어온다. 만일 화일이 존재하지 않으면 새로운 화일을 만든다. (다른 에디터들의 'file open' 과 유사) C-x C-v (find-alternate-file) : 방금 읽은 화일 대신 다른 화일을 읽어온다. 잘못된 화일을 읽었을 때 사용한다.

2.3.2 help

Emacs help는 사용자와의 대화를 위하여 종종 다른 윈도우를 열거나, 미니버퍼를 사용하거나, 때로는 전체 윈도우를 차지하기도 한다. 미니 버퍼에 나오는 내용들은 종종 중요한 지식을 얻게 해주므로, 미니 버퍼의 내용들을 유심히 보는 것도 Emacs 를 빨리 익히는 한 방법일 것이다. 예를 들어 C-h 를 눌러보자. ? 를 눌러보라고 한다. ?를 눌러보자. C-h를 한번 더 누르면 자세한 정보를 주겠다고 한다. 눌러보자. 설명이 나온다. 그리고 화면에 다 안 나오면 space 를 누르면 화면이 스코롤된다고 알려준다. (C-h를 마구 눌러도 같은 결과가 생긴다.) t를 눌러서 TURTORIAL을 선택해 보자. 이제 TURTORIAL 을 이용해서 Emacs 공부를 해보자.

2.3.3 C-h f

Describe function: 이라고 나온다. 여기다 describe-function 이라고 쳐보자.(이러한 경우에도 completion이 적용된다. 한번 시도해 보자.) 그러면, 화면이 갈라지면서 아래 윈도우에 describe-function 이라는 명령에 대한 설명이 나오고, 미니버퍼에는 친절하게도, C-x 1 을 누르면 다시 화면이 하나가 된다는 이야기가 나온다. (이 내용은 버젼에 따라 조금 다를 수도 있을 것이다. 어쨌든 지금 필자가 쓰고 있는 Emacs 는 이렇게 동작한다.)

앞에서 필자는 "Learning GNU Emacs" 라는 책을 소개했었다. '음, 잘못된 내용이 적으려면 책뿐만아니라 실제로 사용해서 확인해보고, 또 Emacs 자체의 설명도 읽어보는 것이 좋겠지.' 그래서, 필자는 Emacs 의 'info' 를 읽어보기 시작했던 것이다. '그래, Emacs 를 익히는데 도움이 많이 되겠군.'

마음 같아선 그 내용들을 다 번역해서 올리고 싶지만, 참기로 했다. 어쨌든 일단 info 로 들어가 보자. ('C-h i' 를 누른다.) Emacs 노드로 가자. (g 를 누르고 emacs RETURN 을 누른다. 아니면, '* Emacs: (emacs)' 라고 된 줄을 찾아가서 RETURN 을 눌러도 된다.)

2.3.4 삽입

삽입을 위해서는 그냥 문자를 치면 된다. 방금 친 문자를 지우려면 DEL 키를 사용하면 된다. "Auto Fill" 모드를 사용하면 줄이 길어질 경우 자동으로 줄이 나누어지게 할 수 있다. 만일 당신이 문자가 삽입되어지는 대신에 덮어씌워지기를 바란다면 Overwrite 모드를 사용할 수 있다.

control 문자나 8진수 200 이상의 문자 코드를 가지는 문자등 직접 삽입이 힘든 문자를 입력하려면 'C-q' 명령을 사용하면 된다.

- **C-q** 다음 입력문자를 읽어서 삽입한다. 아니면 삽입될 문자의 코드를 3자리 8진수로 입력할 수 있다. 예를들어 C-g 문자를 입력하려면, 'C-q C-g' 혹은 'C-q 0 0 7' 을 치면 된다.

2.3.5 이동 명령

- **C-f** 앞으로 한칸 이동. ('forward-char')
- **C-b** 뒤로 한칸 이동. ('backward-char')
- **C-p** 전 줄로 이동. ('previous-line')
- **C-n** 다음 줄로 이동. 만일 버퍼의 끝이면 새로운 줄을 만든다. ('next-line')
- **M-f** 한 단어 앞으로 이동. ('forward-word')

- **M-b** 한 단어 뒤로 이동. (‘backward-word’)
- **C-a** 줄의 처음으로 이동. (‘beginning-of-line’)
- **C-e** 줄의 맨 뒤로 이동. (‘end-of-line’)
- **M-a** 한 문장 뒤로 이동. (‘backward-sentence’)
- **M-e** 한 문장 앞으로 이동. (‘forward-sentence’)
- **M-[** 한 절 뒤로 이동. (‘backward-paragraph’)
- **M-]** 한 절 앞으로 이동. (‘forward-paragraph’)
- **C-v** 한 화면 위로 이동. (‘scroll-up’)
- **M-v** 한 화면 아래로 이동. (‘scroll-down’)
- **C-x** [한 쪽 위로 이동. (‘backward-page’)
- **C-x**] 한 쪽 아래로 이동. (‘forward-page’)
- **M->** 버퍼의 끝으로 이동. (‘end-of-buffer’)
- **M-<<** 버퍼의 시작으로 이동. (‘begging-of-buffer’)
- **C-l** 현재 줄이 화면의 중앙에 오도록 하고 화면을 다시그린다. (‘recenter’) 숫자 인수 는화면이나 윈도우의 맨 위부터 몇번째 줄에 포인트가 위치할지 말해준다. (0은 맨 위줄). 음수 인수는 화면의 맨 아래서부터 줄을 센다. (-1 은 맨 아래줄)
- **M-r** point 를 화면이나 윈도우의 반 위치의 왼쪽끝에 놓는다. 화면의 글은 움직이지 않는다. (‘move-to-window-line’) 숫자 인수는 화면이나 윈도우의 맨 위부터 몇번째 줄에 포인트가 위치할지 말해준다. (0은 맨 위줄). 음수 인수는 화면의 맨 아래서부터 줄을 센다. (-1 은 맨 아래줄)
- **C-t** 커서 전과 뒤의 두 문자를 바꾼다. (‘transpose-chars’).
- **M-x goto-char** 숫자 N 을 읽어서 커서를 N 번째 문자로 옮긴다. 1 번 위치가버퍼의 시작이다.
- **M-x goto-line** 숫자 N 을 읽어서 커서를 N 번째 줄로 옮긴다. 1 번 줄이 버퍼의 시작이다.

2.3.6 숫자 인수

자, 이 글을 읽으면서 실습을 해보기 바란다. 그냥 읽기만 하는것 보다는 한번 해보는 것이 훨씬 빨리 익힐 수 있는 방법이니까.

위의 ‘M-r’ (‘move-to-window-line’) 명령을 보면 숫자 인수를 받아들인다는 내용이 있는데, 명령에 어떻게 숫자 인수를 줄까?

- **‘C-u’** 다음명령에 숫자 인수를 준다.(혹은 다음 명령을 반복한다.)(‘universal- argument’). C-u 뒤에 마이너스 부호(-)나 숫자가 오면 숫자 인수를 만든다. C-u 뒤에 마이너스 부호(-)나 숫자가 오지 않으면 기본값은 4이다. 마이너스 부호나 숫자 없이 C-u를 반복해서 누르면 누를 때마다 인수를 4배 한다.

  예를 들어서 ‘C-u C-u C-u a’를 누르면 ‘a’ 가 4*4*4 개 찍힌다. 그리고, ‘C-u 30 C-n’을 누르면 30줄 아래로 ‘M-<숫자n>’ <숫자n>을 다음 명령의 인수로 준다. (혹은 반복한다.)

  예를 들어서 ‘M-3 M-5 C-f’ 를 누르면 35문자 앞으로 이동한다.

2.3.7 삭제 명령

- **DEL** 커서 바로 앞의 문자를 지운다. (‘delete-backward-char’).
- **C-d** 커서 위의 문자를 지운다. (‘delete-char’).
- **C-k** 줄의 끝까지 지운다. (‘kill-line’).
- **M-d** 커서 다음에 나오는 단어를 지운다.(‘kill-word’).
- **M-DEL** 커서 앞에 나오는 단어를 지운다.(‘backward-kill-word’).
- **M-k** 커서로부터 문장끝까지 지운다 지운다.(‘kill-sentence’)
- **C-x DEL** 커서로부터 이전 문장의 시작까지 지운다.

2.3.8 변화 취소(Undoing Changes)

‘Emacs 는 문서나 버퍼의 변화를 어떤 양까지(8000자 정도) 취소하는 것을 가능하게 해준다고? 각각의 버퍼에 대해 취소가 따로 적용된다. 일반적으로 각각의 명령은 하나의 취소 기록을 남기지만, ‘query-replace’ 처럼 여러 기록을 남기는 명령도 있고 묶여서 기록되는 경우도 있다.’

- **C-x u** 한단계 취소를 행한다. (‘undo’).
- **C-_** 같다.

계속적인 ‘C-_’ 또는 ‘C-x u’ 명령은 기록된 한계까지 이전의, 또 그 이전의 변화를 취소한다. 만일 기록된 것이 없으면 에러 메시지를 낸다. 취소 명령이 아닌 어떤 명령도 취소 명령의 연속을 끊는다. 이 시점에서 이전의 취소 명령들은 변화로 인식되어 취소 명령에의해 취소 될 수 있다. 취소를 취소하려면, ‘C-f’ 등의 명령을 내리고 취소 명령을 내리면 된다. (좀더 깊은 내용을 알고 싶으시다면 Emacs 의 info 를 참조하세요.)

2.3.9 미니버퍼

화면의 맨 아랫줄은 “minibuffer window” 혹은 “echo area” 라고 불리운다. 이 영역은 사용자에게 간단한 정보(에러 메시지등)을 전달하거나, 화일명, 버퍼명, Lisp 함수명, 등등의 복잡한 인수를 넘겨받는데 사용된다. 때때로 미니버퍼는 디폴트 인수를 괄호 안에 제시하기도 한다. 이 때 이 디폴트 인수를 이용하려면 그냥 RETURN 을 누르면 된다. 미니버퍼를 사용하는 명령을 취소하는 간단한 방법은 ‘C-g’를 누르는 것이다. 이것은, 미니버퍼를 사용하지 않는 명령이라도 여러키들을 누르는 명령의 경우에 사용할 수 있다.

- **C-g** quit 신호를 보낸다. (‘keyboard-quit’).

미니버퍼를 사용하는 명령을 반복하는 방법이 있다.

- **C-x ESC** 최근의 미니버퍼 명령을 다시 실행한다.(‘repeat-complex-command’).
- **M-p** ‘C-x ESC’ 상태에서 이전에 기록된 명령으로 이동한다. (‘previous-complex-command’).
- **M-n** ‘C-x ESC’ 상태에서 다음(좀더 최근)에 기록된 명령으로 이동한다. (‘next-complex-command’).
- **M-x list-command-history** 모든 명령 기록을 보여준다. ‘C-x ESC’ 로 반복할 수 있는 모든 명령을 보여준다. 가장 최근의 것이 앞에 온다.

2.3.10 도움말 기능

Emacs 의 도움말 명령은 모두 'C-h' 로 시작한다. h 는 help 라는 단어에서 따왔다.

- **C-h a STRING RET** 이름에 STRING 을 포함하는 명령의 목록을 보여준다. ('command-apropos').
- **C-h b** 현재 효과있는 키 "binding" 을 모두 보여준다. ('describe-bindings').
- **C-h c KEY** KEY 가 실행시키는 명령의 이름을 표시해 준다. ('describe-key-briefly'). 'c' 는 'character' 를 나타낸다.
- **C-h f FUNCTION RET** FUNCTION 이라는 이름의 Lisp 함수에 대한 설명을 보여준다.('describe-function'). 명령은 Lisp 함수 이므로 명령이름이 사용될 수도 있다.
- **C-h i** Info 를 실행한다. ('info').
- **C-h k KEY** KEY 가 실행시키는 명령의 이름과 내용을 보여준다. ('describe-key').
- **C-h l** 당신이 친 마지막 100개의 문자를 표시해준다. ('view-lossage').
- **C-h m** 현 주모드(major mode) 에 대해 설명해준다. ('describe-mode').
- **C-h n** Emacs 의 변화를 알려준다. ('view-emacs-news').
- **C-h s** 현 syntax table 의 내용과 그것의 의미를 보여준다. ('describe-syntax').
- **C-h t** Emacs tutorial 을 보여준다. ('help-with-tutorial').
- **C-h v VAR RET** Lisp 변수 VAR 에 대한 설명을 해준다. ('describe-variable').
- **C-h w COMMAND RET** 어떤 키가 COMMAND 명령을 실행시키는지 알려준다. ('where-is').

Emacs 에 조금 익숙해졌다면 이 도움말 기능을 많이 활용하기를 적극 권한다. 뭔가 궁금하다면 도움말 기능을 어떻게 이용할까 생각하는 습관을 가지시길. Emacs 를 보다 더 잘 사용할 수 있는 비결일 것이다.

2.3.11 간단한 설정 변환

'customize' 란 단어를 찾아보면 '주문에 따라 만들다.' 라는 뜻이 있다. 'customization' 이란 단어를 좀 더 정확히 표현하자면 '주문에 따라 만드기' 정도가 될 터인데, 그냥 '설정 변환'이라는 좀 애매한 단어를 사용하였다. 전회에 말했듯이 여기에 필자의 '.emacs' 화일을 공개한다. '.emacs' 화일은 Emacs 가 가동되면서 하여야 할 일들을 기술해 놓는 화일이다. Emacs 의 자체 Lisp 언어로 기술 되어져있지만, 잘 살펴보면 다른 변화를 주는 것도 그다지 어렵지는 않으리라 생각된다. 그러나, 원래의 표준 키 binding 은 되도록이면 바꾸지 말 것을 권한다.

```
;
;   hwijae의
;  .emacs 화일
;  줄에서 ';' 문자의 뒷 부분은 주석임.
;  Linux SLS 1.03 설치하면서 있던 .emacs 화일에 살을 붙임.
; text 모드가 될 때 turn-on-auto-fill 을 실행하라는 뜻.
; (즉, text 모드가 되면 auto-fill 모드가 됨.)
(setq text-mode-hook 'turn-on-auto-fill)
(setq make-backup-files t)              ; backup file 을 만들어라.
(put 'eval-expression 'disabled nil)    ; 원래 있던 내용. (?)
; 화일 명에 따라 모드가 선택됨.
; 둘째줄은 .c 로 끝나는 화일명을 가진 화일을 열면 c-mode 로
; 들어가라는뜻.
(setq auto-mode-alist (mapcar 'purecopy
```

```
'(("\\.c$" . c-mode)
 ("\\.h$" . c-mode)
 ("\\.C$" . c++-mode)
 ("\\.cc$" . c++-mode)
 ("\\.H$" . c++-mode)
 ("\\.hh$" . c++-mode)
 ("\\.cxx$" . c++-mode)
 ("\\.hxx$" . c++-mode)
 ("\\.cpp$" . c++-mode)
 ("\\.hpp$" . c++-mode)
 ("\\.tex$" . TeX-mode)
 ("\\.txi$" . Texinfo-mode)
 ("\\.el$" . emacs-lisp-mode)
 ("\\.a$" . c-mode))))

; 추가된 키 binding.
; `M-[' (`backward-paragraph')명령을 사용하지 못하게 됨.
; 화살표키와 Insert, Page Up, Page Down 등의 키가 그 의미에 맞게
; 행동한다.
; 기능키들에 (function keys) 원하는 기능을 넣을 수 있다.

(defvar cursor-map-2 (make-keymap)"for ESC-[")      ; ?
(fset 'Cursor-Map-2 cursor-map-2)                   ; ?
(define-key esc-map "[" 'Cursor-Map-2)              ; ?
(define-key esc-map "[3~" 'delete-backward-char)    ; del
(define-key esc-map "[2~" 'overwrite-mode)    ; insert
(define-key esc-map "[5~" 'scroll-down)       ; page-up
(define-key esc-map "[1~" 'beginning-of-line) ; home, X 상에서는 안됨.
(define-key esc-map "[4~" 'end-of-line)       ; end,      ''         .
(define-key esc-map "[6~" 'scroll-up)         ; page-down

(setq f1-function 'count-lines-page)
(setq f2-function 'find-file)
(setq f3-function 'other-window)
(setq f4-function 'delete-other-windows)
(setq f5-function 'split-window-vertically)
(setq f6-function 'switch-to-buffer)
(setq f7-function 'recenter)
(setq f8-function 'recenter)
(setq f9-function 'recenter)
(setq f10-function 'goto-line)
(setq f11-function 'where-is)
(setq f12-function 'describe-function)
(define-key esc-map "[11~" f1-function) ; F1
(define-key esc-map "[12~" f2-function) ; F2
(define-key esc-map "[13~" f3-function) ; F3
(define-key esc-map "[14~" f4-function) ; F4
(define-key esc-map "[15~" f5-function) ; F5
(define-key esc-map "[17~" f6-function) ; F6
(define-key esc-map "[18~" f7-function) ; F7
(define-key esc-map "[19~" f8-function) ; F8
(define-key esc-map "[20~" f9-function) ; F9
(define-key esc-map "[21~" f10-function) ; F10
(define-key esc-map "[23~" f11-function) ; F11
(define-key esc-map "[24~" f12-function) ; F12
```

```
(define-key esc-map "[[A"  f1-function) ; F1
(define-key esc-map "[[B"  f2-function) ; F2
(define-key esc-map "[[C"  f3-function) ; F3
(define-key esc-map "[[D"  f4-function) ; F4
(define-key esc-map "[[E"  f5-function) ; F5

(define-key esc-map "[A" 'previous-line)             ; up-arrow
(define-key esc-map "[B" 'next-line)                 ; down-arrow
(define-key esc-map "[C" 'forward-char)              ; right-arrow
(define-key esc-map "[D" 'backward-char)             ; left-arrow

(define-key esc-map "OA"        'previous-line) ;up-arrow
(define-key esc-map "OB"        'next-line)     ;down-arrow
(define-key esc-map "OC"        'forward-char)  ;right-arrow
(define-key esc-map "OD"        'backward-char) ;left-arrow

;
; 설치할 때부터 있던 내용. 필자는 잘 모름.
;
(setq load-path
      (cons (expand-file-name "PATH_FOR_AUC-TEX") load-path))
(autoload 'tex-mode
          "auc-tex" "Automatic select TeX or LaTeX mode" t)
(autoload 'plain-tex-mode "auc-tex" "Mode for Plain TeX" t)
(autoload 'latex-mode "auc-tex" "Mode for LaTeX" t)
(autoload 'LaTeX-math-mode    "tex-math"      "Math mode for TeX." t)
(autoload 'outline-minor-mode "outline-m" "Minor Outline Mode." t)

(defun Ctl-C-prefix ()
  Ctl-C-keymap)
(setq Ctl-C-keymap (make-keymap))
(global-set-key "\C-c" (Ctl-C-prefix))

(define-key Ctl-C-keymap "m" 'LaTeX-math-mode)
(define-key Ctl-C-keymap "o" 'outline-minor-mode)

(make-variable-buffer-local 'outline-prefix-char)
(setq-default outline-prefix-char "\C-l")
(make-variable-buffer-local 'outline-regexp)
(setq-default outline-regexp "[*\l]+")
(make-variable-buffer-local 'outline-level-function)
(setq-default outline-level-function 'outline-level-default)

(autoload 'ispell-word "ispell"
  "Check the spelling of word in buffer." t)
(global-set-key "\e$" 'ispell-word)
(autoload 'ispell-region "ispell"
  "Check the spelling of region." t)
(autoload 'ispell-buffer "ispell"
  "Check the spelling of buffer." t)
(autoload 'ispell-complete-word "ispell"
  "Look up current word in dictionary and try to complete it." t)
(autoload 'ispell-change-dictionary "ispell"
  "Change ispell dictionary." t)
```

```
(setq-default abbrev-mode t) ; 시작과 동시에 abbrev 모드가 된다.
(setq save-abbrevs t)

(if (equal (nth 0 command-line-args) "mule")     ; 만일 mule이면
    (setq load-path
          (cons (expand-file-name "/usr/local/emacs/lisp") load-path))

())
(load "c++mode")                        ; c++mode 를 읽는다.

; c 에서 indentation (만입?) 형태.
; 필자는 네칸을 좋아함.
(setq c-indent-level 4)
(setq c-continued-statement-offset 4)
(setq c-continued-brace-offset -4)

; c 와 c++ 모드에서 RETURN 을 누르면
; 다시 indent 하고 줄 바꾸고 indent 함.
(define-key c++-mode-map "\r" 'reindent-then-newline-and-indent)
(define-key c-mode-map "\r" 'reindent-then-newline-and-indent)

(setq explicit-shell-file-name "/bin/bash") ; shell 모드에서 사용할
                                            ; shell 을 명확히 해줌.
(read-abbrev-file "~/.abbrev_defs")     ; 약어들을 읽어들임.

(if (equal (nth 0 command-line-args) "mule") ; 만일 mule 이면,
(let ()
; 송재경님의 원 .emacs 화일
; 미닉스가 조금 고침.
; 필자가 많이 삭제하고(앞에서 겹치는 부분) 조금 고침.
;
; 두벌식 사용자를 위한 키보드 정의
; 한글영문전환은 알트+스페이스로 정의되어 있음.
; /usr/local/hanauto 화일이 두벌식입력오토마타로 쓰임.
;(load-library "/usr/local/mule/lisp/hangul-mode")
;

;세벌식 사용자를 위한 키보드정의
;자체 입력오토마타를 사용함.
;세벌식을 쓰기 위해서는 이 줄 위로 네번째의 첫칸에 ; 표시를하고
;아래 두줄의 첫칸에 있는 ; 를 제거하면 됨. (필자는 세벌식을 사용함.)
      (require 'quail)
      (load "quail/hangul3")
      (setq kanji-input-code 0)
      )
  ())
;
; 끝
;
```

2.3.12 마크와 영역(The Mark and the Region)

Emacs 에는 현 버퍼의 특정 연속된 부분에 행하여지는 명령이 많이 있다. 이러한 명령이 행하여질 부분을 지정하기 위해 한쪽 끝에 “mark” 를 지정하고, 포인트를 다른 쪽 끝으로 옮겨야 한다. 포인트와 마크 사이의 부

분을 “region” 이라고 한다. (이하 “영역”이라 하겠다.) 마크가 한번 지정된 이후에는, 그것이 다른 위치에 다시 지정될 때까지 남아있는다. 마크는 텍스트가 입력되거나 삭제될 때 그 앞의 문자에 따라 위치가 고정되어 있는다. 각각의 Emacs 버퍼는 각각 자신의 마크를 가지고 있다. ‘C-y’ (‘yank’) 와 ‘M-x insert-buffer’ 와 같이 텍스트를 삽입하는 많은 명령들은 마크를 삽입되는 텍스트의 한 끝에 위치시키고 다른 한 끝에 포인트를 위치시킨다. 따라서, 영역은 바로 삽입된 텍스트가 된다. 영역을 설정하는 것과는 별개로 마크는 되돌아갈 위치를 설정하는대로 유용하다. 이것을 좀더 유용하게하기 위해 Emacs 는 16개의 이전 마크 위치를 ‘mark ring’ 에 기억한다.

- **C-SPC** 포인트가 있는 위치에 마크를 지정한다. (‘set-mark-command’).
- **C-@** 같다.
- **C-x C-x** 마크와 포인트를 바꾼다. (‘exchange-point-and-mark’). 즉, 현재의 포인트가 마크가 되고 마크가 있던 위치로 포인트가 이동한다.

위의 명령들을 먼저 기억하는 것이 좋을 것 같다.

- **M-@** 단어의 끝에 마크를 지정한다. (‘mark-word’). 이 명령과 다음의 명령은 포인트를 이동시키지는 않는다.
- **C-M-@** Lisp 표현의 끝에 마크를 지정한다. (‘mark-sexp’).
- **M-h** 현재 문단을 영역으로 만든다. (‘mark-paragraph’).
- **C-M-h** 현재 Lisp defun 을 영역으로 만든다. (‘mark-defun’).
- **C-x h** 버퍼 전체를 영역으로 만든다. (‘mark-whole-buffer’).
- **C-x C-p** 현재 page 를 영역으로 만든다. (‘mark-page’).

그 외에 ‘M-<’ 와 같이 멀리 이동하는 명령의 경우에 이동하기 이전의 포인트 위치에 마크가 된다. 이 경우에 echo area 에 “Mark set” 이라는 메시지가 나온다. 이제 마크를 돌아갈 위치를 기억하는데 이용해보자. Emacs 는 이전 16개 마크의 위치를 “mark ring”에 기억한다. 마크를 지정하는 대부분의 명령은 이전의 마크를 이 “mark ring” 에 push 한다. (쉽게말해서 기억한다.) 이제 마크된 위치로 찾아가는 명령에 대해 알아보자.

‘C-u C-SPC’ 혹은 ‘C-u C-@’ 포인트를 마크 위치로 옮기고 ring 의 이전 마크 위치로 마크를 옮긴다. 멀리 이동하는 명령은 이동하기 전의 위치에 마크를 함으로 되돌아 가는 것을 쉽게 만든다. 변수 ‘mark-ring -max’ 는 mark ring 이 기억하는 최대 마크의 갯수를 나타낸다.

2.3.13 Deletion and Killing

버퍼로부터 텍스트를 지우는 대부분의 명령은 마음이 바뀌었을 때 다시 돌리거나 옮기거나 복사하기 위해 그것들을 저장한다. 이런 명령들은 “kill” 명령이라 알려져있다. 지워지는 텍스트를 저장하지 않는 명령들은 “delete” 명령이라 알려져있다. (이하 kill 은 삭제, delete 는 제거라 구분하겠다.) 제거 명령은 한 글자만 지우는 명령이나, space 나 newline 만을 지우는 명령들이 포함된다. 실수로 글을 삭제, 혹은 제거했을 경우에는 ‘C-x u’ (‘undo’) 명령을 사용해서 취소할 수 있다.

제거

- **C-d** 커서 위의 문자를 지운다. (‘delete-char’).
- **DEL** 커서 바로 앞의 문자를 지운다. (‘delete-backward-char’).
- **M-** 포인트 주위의 space 들과 tab 들을 지운다. (‘delete-horizontal-space’).

- **M-SPC** 포인트 주위의 space 들과 tab 들을 space 하나만 남기고 지운다. ('just-one-space').
- **C-x C-o** 현 줄 주위의 빈 줄들을 지운다. ('delete-blank-lines').
- **M-^** 사이의 newline 과 따르는 indentation 을 지워서 두 줄을 하나로 합친다. ('delete -indentation').

삭제 명령

- **C-k** 줄의 나머지 부분이나 몇개의 줄을 지운다. ('kill-line').
- **C-k** 에 숫자 인수를 주면 여러줄을 한꺼번에 지운다. 만일 음수 인수를 주면 포인트 앞쪽을 지운다. 만일 0을 인수로 주면 포인트에서 줄의 시작까지 지운다.
- **C-w** (포인트에서 마크까지의) 영역을 지운다. ('kill-region').
- **M-d** 단어를 지운다. ('kill-word').
- **M-DEL** 단어를 뒷방향으로 지운다. ('backward-kill-word').
- **C-x DEL** 뒤로 문장의 시작까지 지운다. ('backward-kill-sentence').
- **M-k** 문장의 끝까지 지운다. ('kill-sentence').
- **M-z CHAR** CHAR 가 나오기 전 위치까지 지운다. ('zap-to-char').

붙이기 (Yanking)

- **C-y** 마지막으로 삭제된 텍스트를 삽입한다. ('yank').
- **M-y** 이전에 삭제된 텍스트로 다시 삽입한다. ('yank-pop').
- **M-w** 영역을 실제로 삭제하지는 않고 마지막으로 삭제된 텍스트로 만든다. ('copy-region-as-kill').
- **C-M-w** 뒤 따라 내리는 삭제 명령에 의해 삭제되는 텍스트를 마지막에 삭제된 텍스트에 붙인다. ('append-next-kill').

'C-y', 'M-y' 명령만 잠시 살펴보자.

예를 들어 A-B-C-D 네개의 문단을 가지는 글에서 B 문단을 삭제(kill) 했다고 가정해보자. 이제 A-C-D 와 같은 글이 될 것이다. 이제 C, D 문단의 순서를 바꾸고 싶다. 그러면 C를 삭제하고, D의 끝에서 'C-y' 를 누르면 된다. 이제 글은 A-D-C의 구조를 가지게 된다. 그런데, 그렇게 하고나니 B의 내용을 A 앞에 놓고 싶어졌다. 그러면, A 의 앞에가서 'C-y'를 누른다. 그러면, C가 삽입될 것이다. 이 때, 'M-y' 를 누르면, C의 내용이 삽입된 자리에 B의 내용이 들어갈 것이다. 'M-y' ('yangk-pop') 명령은 'C-y' 혹은, 그에 이은 'M-y' 명령 뒤에서만 동작한다. 즉, 'C-y' 를 누르면 가장 최근에 삭제된 내용이 삽입되고, 'M-y'를 누를 때마다, 그 이전에 삭제된 내용으로 바뀌어 삽입되는 것이다. 이렇듯 삭제되는 내용은 "kill ring" 에 저장되는데, 저장되는 삭제된 텍스트의 최대 갯수는 kill-ring-max 라는 변수에 저장된다. 보통 기본값은 30이다.

만일 'M-y' 를 몇번 눌러야 원하는 글이 나올지 알고 있다면, 'C-y' 에 숫자 인수를 주어서 바로 꺼내올 수 있다. 예로 'C-u 2 C-y' 는 'C-y M-y'와 같다.

2.3.14 직사각형

직사각형 영역을 다루는 명령들을 살펴보자. 직사각형을 영역을 설정하는 것은 보통 영역을 설정하는 것과 비슷하다. 한쪽 끝에 포인트, 한쪽 끝에 마크를 위치 시키면 된다.

- **M-x delete-rectangle** 직사각형 영역을 지운다.

- **M-x kill-rectangle** 위의 명령과 비슷하다. 그러나, 지워지는 내용을 "last killed rectangle" 로 저장한다.
- **M-x yank-rectangle** “last killed rectangle”을 그것의 위 왼쪽 끝이 포인트 위치에 오도록 삽입한다.
- **M-x open-rectangle** 직사각형영역을 빈 공간으로 채운다. 원래의 내용은 직사각형 영역 오른쪽으로 밀려난다.
- **M-x clear-rectangle** 직사각형영역의 내용을 없에고 스페이스를 채운다.

2.3.15 등록부(Registers)

Emacs “등록부” 는 텍스트나 위치를 다음 사용을 위해 저장하는 장소이다. 등록된 텍스트는 버퍼에 원하는 만큼 복사될 수 있다; 위치는 다음에 그 위치로 돌아오기 위해 등록할 수 있다. 직사각형 영역들도 등록부에 등록될 수 있다. 각 등록부들은 한 문자로 된 이름을 가지고 있다. 각각의 등록부는 하나의 텍스트나 위치, 혹은 직사각형 영역을 저장할 수 있다. 하지만, 한 등록부에는 한 시점에 한가지만 저장할 수 있다. 등록부에 무엇을 등록하든지 그것들은 등록부에 다른것을 저장할 때까지 보존된다.

- **M-x view-register RET R** 등록부 R 에 저장된 것을 보여준다.
- **C-x / R** 포인트의 위치를 등록부 R 에 저장한다. (‘point-to-register’)
- **C-x j R** 등록부 R 에 저장된 위치로 이동한다. (‘register-to-point’).
- **C-x x R** 영역을 등록부 R 에 복사한다. (‘copy-to-register’).
- **C-x r R** 직사각형 영역을 등록부 R 로 복사한다. (‘copy-region-to-rectangle’). 숫자 인수가 주어지면 복사하고 지운다.
- **C-x g R** 등록부 R 의 내용을 삽입한다. (‘insert-register’).

2.3.16 점진적 탐색(Incremental Search)

점진적 탐색은 찾는 문자열의 첫번째 문자를 치면 시작된다. 찾는 문자열을 입력함에따라 이막스는 문자열이 있을만한 곳을 보여준다. 만일 찾고자 하는 위치를 판별하기에 충분한 문자열을 입력했다면, 멈출 수 있다.

- **C-s** 앞쪽으로 점직전 탐색 (‘isearch-forward’).
- **C-r** 뒷쪽으로 점진적 탐색 (‘isearch-backward’).

만일 탐색문자열을 입력하는데 실수를 했다면, DEL키를 이용하여 문자들을 지울 수 있다. 만일 원하는 위치에 도달했다면 ESC를 눌러서 탐색을 멈추고 그 위치로 갈 수 있다. 탐색에 관계없는 명령을 입력할 경우에는 그 위치에 커서를 멈추고 그 명령을 실행한다. 예를 들어 ‘C-a’를 입력하면 커서가 있던 줄의 앞으로 이동한다. 만일 다음 명령이 문자입력이나, DEL, ESC 나 탐색에 의미가 있는 특별한 제어 명령(‘C-q’, ‘C-w’, ‘C-r’, ‘C-s’, ‘C-y’ 등)일 경우 ESC를 누르는 것은 필수적이다.

때때로 찾고자 하는 문자열이 여러군데 존재하고 처음에 찾은 위치가 원하는 위치가 아닐 경우, ‘C-s’를 한번 더 눌러서 다음의 그 문자열이 있는 위치로 이동할 수 있다. 만일 ‘C-s’를 너무 눌러서 원하는 위치를 지나쳤을 경우에는 DEL로 ‘C-s’들을 취소할 수 있다. 탐색을 멈춘후, 같은 문자열을 또 찾고 싶을 경우에는 ‘C-s C-s’를 누르면 된다. 첫번째 ‘C-s’는 점진적 탐색을 시작하는 것이고, 두번째 ‘C-s’는 “재 탐색”을 의미한다. 만일 탐색 문자열을 찾을 수 없는 경우, 에코영역에 ‘Failing I-Search’ 라고 씌여진다. 커서는 탐색 실패 이전의 위치에 머물러 있는다. 예로‘FOOT’을 찾는데 ‘FOOT’은 없고 ‘FOOL’이 있다면, 커서는 ‘FOOL’의 FOO뒤에 위치한다. 이 경우에 할 수 있는 일은 여러가지가 있다. 만일 찾고자 하는 문자열을 잘못 입력했다면 탐색문자열을 수정할 수 있다. 아니면 그 위치에서 탐색을 멈추기 위해 ESC나 기타 명령을 내릴 수 있다. 아니면, ‘C-g’를 눌러서

찾을 수 없는 문자들을 없앨 수 있다. 위의 예에서는 'FOOT'의 'T'가 된다. 그리고 탐색문자열은 'FOO'가 된다. 그리고 'C-g'를 두번째 누르면 탐색은 취소가 되고, 커서는 탐색을 시작했던 위치로 되돌아간다. 탐색이 실패했을 경우에 'C-s'를 누르면, 버퍼의 시작에서부터 다시 탐색을 시작한다. 'C-r'을 계속 눌러서 뒷쪽으로 탐색에하는데 실패한 경우에 'C-r'을 누르면 버퍼의 끝에서 부터 다시 탐색을 시작한다. 이것은 "wrapping aroun"라고 불리우고, 'Wrapped'라고 표시된다. 'C-s', DEL, ESC 같은 제어문자를 탐색하려면 'C-q'를 먼저 쳐야한다. 이것은 다음의 문자를 탐색문자열의 일부로 간주하게 만든다. 탐색을 뒷쪽으로 하려면, 'C-r'을 누르면 된다. 마찬가지로 탐색방향을 다시 앞쪽으로 하려면, 'C-s'를 누르면 된다. 처음부터 뒷쪽으로 탐색을 하려면 처음에 'C-r'을 눌러서 탐색을 시작할 수 있다. 점진적 탐색에서 'C-y' 와 'C-w' 는 버퍼의 텍스트를 탐색문자열로 잡아오는데 사용된다. 'C-w'는 현재 포인트 이후의 단어부분을 탐색문자열로 복사해 온다. 'C-y'는 현재의 단어 전체를 탐색문자열로 복사해 온다. (무슨말인지 잘 모르겠을 때는 직접 해보면 된다.) 점진적 탐색에서 사용되는 특수문자들은 다음의 변수들을 수정해서 바꿀 수있다.

- **search-delete-char** 한 문자를 지우는 키 (보통 DEL).
- **search-exit-char** 점진적 탐색을 빠져나오는 키 (보통 ESC).
- **search-quote-char** 특수 문자를 탐색 문자열에 포함시킬 경우 사용하는 키 (보통 'C-q').
- **search-repeat-char** 점진적 탐색을 앞쪽으로 반복 (보통 'C-s').
- **search-reverse-char** 점진적 탐색을 뒷쪽으로 반복 (보통 'C-r').
- **search-yank-line-char** 버퍼의 줄을 탐색문자열로 복사 (보통 'C-y').
- **search-yank-word-char** 버퍼의 단어를 탐색문자열로 복사 (normally 'C-w').

(* 변수 값을 바꾸는 명령 : 'M-x set-variable')

2.3.17 비점진적 탐색(Nonincremental Search)

Emacs 는 또한 탐색 문자열을 모두 입력한 뒤에 탐색을 시작하는 전통적인 비점진적 탐색방법을 제공한다.

- **C-s ESC STRING RET** STRING 을 탐색한다.
- **C-r ESC STRING RET** STRING 을 뒷쪽으로 탐색한다.

2.3.18 단어 탐색(Word Search)

단어 탐색은 단어들의 연속을 단어들이 어떻게 분리되어 있는가에 관계없이 탐색을 한다. 좀 더 정확히 말하면, 단어들을 하나의 공간(space)을 이용하여 분리하면, 단어 사이에 여러개의 공간이 있거나, 새줄(newline)이 있거나 마침표가 있거나 다 찾아 준다.

- **C-s ESC C-w WORDS RET** WORDS 를 탐색한다. 구두법의 차이를 무시한다.
- **C-r ESC C-w WORDS RET** WORDS 를 뒷쪽으로 탐색한다. 구두법의 차이를 무시한다.

비점진적 탐색이나 단어 탐색은 각각 'search-forward' 와 'search-backward', 'word-search- forward' 와 'word-search-backward' 로 구현되어 있다. 이들은 새로운 키에 바인딩 될 수도 있지만, 점진적 탐색이 이들을 부를 수 있도록 만들어져있기 때문에 위와 같은 키 연속이 Emacs 에서 이들을 부르는 일반적인 방법이다.

2.3.19 정규식 탐색(Regular Expression Search)

이것은 점직적으로 할 수도 있고, 비 점진적으로 할 수도 있다. 점진적 탐색은 'C-M-s' ('isearch-forward-regexp') 를 누르면 불려진다. 이 명령은 'C-s' 처럼 정규식을 점진적으로 받아들이지만, 정확히 일치하는 표현을 찾는 것이 아니라, 탐색 문자열을 정규식으로 간주한다. 'isearch-backward-regexp' 명령도 존재하지만, 어떤 키에 바인딩 되어 있지는 않다.

일반적인 점진적 탐색에서 적용되는 제어문자들이 여기에도 적용된다. 비점진적 정규식 탐색은 're-search-forward' 와 're-search-backward' 함수로 구현되어 있다. 이 함수들은 'M-x'를 이용하거나, 어떤 키에 바인딩하거나 하는 방법으로 부를 수 있다. 아니면, 'C-M-s ESC'로 불러도 된다.

2.3.20 정규식의 형식(Syntax of Regular Expressions)

정규식은 약간의 특별한 기호를 가지고 있고 나머지는 일반 문자로 취급된다. 일반 문자는 단순히 그 문자와 일치 여부를 판정하는 간단한 정규식이다. 특수 문자는 '$', '^', '.', '*', '+', '?', '[', ']' , '\' 이다. 정규식에 나타나는 다른 문자는, '\'가 앞에 있는 경우를 제외하고는 일반 문자이다.

- **. (Period)** 는 새줄(newline)을 제외하고는 어떤 문자하고도 일치한다.
- **a.b** 는 'a'로 시작해서 'b'로 끝나는 어떤 세글자 문자열과도 일치한다.
- ***** 는 그 자체로는 의미가 없다. 이것은 앞서는 식이 몇번이고 나올 수 있다는 뜻의 접미문자이다. 'fo*' 에서 '*'는 'o'에 적용된다. 따라서, 몇개의 'o'가 뒤따르는 'f'와 일치한다. 'o'가 영개 있는 경우도 허용된다: 'fo*' 는 'f'와 일치한다. '*' 는 항상 앞의 식 중에 가능한한 가장 작은 식에 적용된다. 따라서, 'fo*' 는 'fo' 의 반복이 아니라 'o' 의 반복을 의미한다.
- **+** 는 접미문자로 앞서는 식이 적어도 한번은 나와야 한다는 것을 제외하고는 '*'와 같은 의미이다. 예로 'ca+r' 은 'car' 'caaaar'과는 일치하지만 'cr'과는 일치하지 않는다. 그러나, 'ca*r'은 위의 세가지 문자열과 모두 일치한다.
- **?** 는 앞서는 식이 한번 나오거나 전혀 안나온다는 것을 의미하는 접미문자이다. 예로 'ca?r' 은 'car' 나 'cr' 하고만 일치한다.
- **[...]** '[' 는 ']'로 끝나는 "문자군(character set)"의 시작을 의미한다. 간단한 경우에 두 기호 사이의 문자들이 문자군을 이룬다. '[ad]'는 'a' 나 'd'와 일치한다. '[ad]*'는 'a'나 'd'로만 이루어진 모든 문자열과 일치한다. 'c[ad]*r' 은 'cr', 'car', 'cdr', 'caddaar' 등등과 일치한다. 두 문자사이에 '-'를 써서 문자군에 포함될 문자 범위를 나타낼 수 있다. 따라서, '[a-z]'는 모든 소문자와 일치한다. 범위는 각각의 문자와 함께 쓰일 수 있다. '[a-z$%.]' 는 모든 소문자와 '$', '%',혹은, 마침표와 일치한다. 문자군 안에서 ']', '-', '^' 외의 특수 문자는 더이상 특별한 의미를 가지지 않는다. ']'를 문자군에 포함 시키려면, 첫번째 문자로 포함시켜라. 예로 '[]a]'는 ']'나 'a'와 일치한다. '-'를 포함시키려면, '-'만을 포함하는 범위인 '—'를 써라. '^'를 포함시키려면, 문자군의 첫번째 문자로 쓰지마라.
- **[^ ...]** '[^' 로 시작해서 ']'로 끝나는 문자군은 나열되지 않은 문자와 일치한다. 따라서, '[^a-z0-9A-Z]' 는 영문자와 숫자가 아닌 모든 문자와 일치한다. 첫문자가 아닌 '^'는 특별한 의미를 가지지 않는다. 또한, '^'바로 뒤의 문자는 첫번째 문자로 간주된다. ('-'와 ']' 는 이 위치에서 특별한 의미를 가지지 않는다.) 이 문자군은 newline 이 문자군의 하나로 지칭되지 않으면 newline과 일치할 수 있다.
- **^** 는 줄(line)의 시작과 일치하는 특수 문자이다. 따라서 '^foo'는 줄의 시작에 나오는 'foo'와 일치한다.
- **$** 는 줄의 끝과 일치한다. 따라서 'xx*$'는 하나 혹은 그 이상의 'x'가 줄의 끝에 있으면 일치한다.
- **** 는 두가지 기능을 가지고 있다. 그것은 특수 문자('\'를 포함한)를 따오거나(정규식에 포함시키거나), 다른 특별한 구문을 제공한다. 예로, '\$'는 '$'문자하고만 일치하는 정규식이고 '\['는 '[' 하고만 일치하는 정규식이고 하는 식이다.

주: 특수문자들은 그들의 특별한 의미가 의미가 없는 위치에서는 일반 문자로 취급된다. 예로, '*foo' 의 '*'는 일반 문자로 취급된다. 그러나 이러한 양상을 이용하는 것은 좋은 습관이 아니다.

'\'구문 표는 다음과 같다.

- \| '\|' 이전과 이후 식에 모두 일치한다. 따라서, 'foo\|bar' 는 'foo' 나 'bar' 하고 일치한다. '\|'는 가능한 가장 큰 식에 적용된다. '\(... \)' 의 그룹핑 만이 '\|'의 그룹핑을 제한할 수 있다. '\|'의 다중 사용에 대한 full backtracking 능력이 존재한다.
- **\(... \)** 는 그룹핑(grouping) 구문으로 다음의 세가지 용도를 제공한다.
  1. '\|'의 범위를 제한하기 위해서 쓰인다. 따라서,'\(foo\|bar\)x'는 'foox' 나 'barx'와 일치한다.
  2. 복잡한 식을 '*'류의 접미사에 적용시키기 위해 사용한다. 따라서, 'ba\(na\)*' 는 'banana' 등과 같이 'na'가 몇번이고 반복되는 문자열과 일치한다.
  3. 다음의 참조를 위하여 사용한다.
- **\DIGIT** 각각의 '\(... \)' 구문에 대하여, 이 구문에 일치하는 텍스트의 시작과 끝이 기억된다. 따라서, 그 이후의 정규식에서 '\'뒤에 숫자(DIGIT)를 씀으로써 DIGIT번째 '\(... \)' 구문에 일치하는 텍스트라는 의미가 된다. 열린 괄호의 순서에 따라, 아홉개의 '\(... \)' 구문이 1번부터9번까지의 번호에 할당된다. 예로, '\(.\)\1' 은 'aa' ,'bb'등 같은 문자가 두개 나오는 경우와 일치한다.
- **\‘** 버퍼의 시작과 일치한다.
- **\’** 버퍼의 끝과 일치한다.
- **\b** 단어의 시작이나 끝과 일치한다. 따라서 '\bfoo\b' 는 분리된 단어로써의 'foo'와 일치한다. '\bballs?\b'는 분리된 단어로써의 'ball'이나 'balls'와 일치한다.
- **\B** 단어의 시작과 끝이 아닌위치와 일치한다.
- **\<** 단어의 시작과 일치한다.
- **\>** 단어의 끝과 일치한다.
- **\w** matches any word-constituent character. The editor syntax table determines which characters these are.
- **\W** matches any character that is not a word-constituent.
- **\sCODE** matches any character whose syntax is CODE. CODE is a character which represents a syntax code: thus, 'w' for word constituent, '-' for whitespace, '(' for open-parenthesis, etc. *Ref Syntax::. '\SCODE' matches any character whose syntax is not CODE.

다음은 Lisp 문법에서 사용된 복잡한 정규식의 한 예이다. Lisp 문법에서 문자열은 큰따옴표(")로 시작해서 큰따옴표로 끝난다. '\"'는 정규식의 부분으로써의 큰따옴표를, '\\'는 정규식의 부분으로써의 역슬래쉬를, '\t'는 탭문자를, '\n'은 새줄문자를 의미한다.

```
"[.?!][]\"')]*\\($\\|\t\\|  \\)[ \t\n]*"
```

이것은 네부분으로 나누어져 있다: '.'(마침표)나, '?', '!'와 일치하는 문자집합; ']', '"'(큰따옴표), '''(작은따옴표), ')' 문자집합이 몇번이고 반복되는 부분; 줄의 끝이거나 탭이거나 두개의 스페이스와 일치하는 부분; 스페이스나 탭, 새줄등의 화이트스페이스문자가 몇번이고 반복되는 부분. 위의 예는 Emacs Lisp 프로그램에서 정규식을 어떻게 사용할 것이가에 대한 예이고, 're-search-forward' 등의 명령을 통하여 정규식을 입력할 경우에는 좀 다르게 표시해야한다.

```
[.?!][]"')]*\($\|^Q^I\|  \)[ ^Q^I^Q^J]*     ('^'는 CTRL키를 누르라는 의미이다.)
```

3 이맥스 20.2 이상에서 한글 사용하기

글쓴이: 천리안 리눅스 동호회 회원 송상원님

글꼴을 고치기 위해서 일단 su 가 된 다음, "/usr/X11R6/lib/X11/app-defaults" 로 디렉토리를 옮긴다. 거기에 보면 "Emacs" 라는 화일이 있을 것이다. 이 화일은 모든 사용자들이 이맥스를 실행시킬때 프로그램이 읽는 화일이다. 이 "Emacs" 화일을 열어보면

```
Emacs.Font:         *-fontset-16
Emacs.Fontset-0:    -*-*-medium-r-normal-*-12-*-*-*-*-*-fontset-12
Emacs.Fontset-1:    -*-*-medium-r-normal-*-14-*-*-*-*-*-fontset-14
Emacs.Fontset-2:    -*-*-medium-r-normal-*-16-*-*-*-*-*-fontset-16
Emacs.Fontset-3:    -*-*-medium-r-normal-*-18-*-*-*-*-*-fontset-18
Emacs.Fontset-4:    -*-*-medium-r-normal-*-20-*-*-*-*-*-fontset-24
Emacs.Fontset-5:    -*-*-medium-r-normal-*-24-*-*-*-*-*-fontset-24
```

위와 같은 것이 있을 것이다. 첫줄은 이맥스가 사용할 폰트셋을 지정하는 것이고, 나머지 아래 6줄은 사용가능한 각각의 폰트셋들을 셋팅해 놓은것이다. 위의 예는 이맥스가 "fontset-16" 을 쓴다. 위의 내용을 주석처리(!) 내지는 삭제를 하고, 다음과 같이 고친다.

```
Emacs.Font:      *-fontset-myeongjo
!Emacs.Font:     *-fontset-gothic

Emacs.Fontset-0:-*-*-medium-r-normal-*-16-*-*-*-*-*-fontset-myeongjo, \
          ascii:-*-*-bold-r-normal--16-*-*-*-*-*-iso8859*-*, \
 korean-ksc5601:-misc-myeongjo-medium-r-normal--16-*-*-*-*-*-ksc5601*-*

Emacs.Fontset-1:-*-*-medium-r-normal-*-16-*-*-*-*-*-fontset-gothic, \
          ascii:-*-*-bold-r-normal--16-*-*-*-*-*-iso8859*-*, \
 korean-ksc5601:-misc-gothic-medium-r-normal--16-*-*-*-*-*-ksc5601*-*
```

위의 예는 명조체와 고딕체의 폰트셋을 만들었고, 그중 명조체를 이맥스에 사용한 경우이다. 물론, 자신이 원한다면 Fontset 을 얼마든지 더 만들수가 있다. 위와 같이 자신이 좋아하는 글자체로 한글폰트를 바꾸고 저장한 후, 이맥스를 실행시키면 바로 글자체가 바뀌어 실행된다. 이때 주의할 점은 폰트의 종류에 따라 이맥스에서 제대로 나오지 않는 폰트들이 있으니, 확인을 해 볼것.