September 28, 1994

emacs: A Practical Tutorial

Luis Fernandes

Emacs is an extensible, customizable, self-documenting real-time editor developed by Richard Stallman, who is also the founder of the Free Software Foundation.
In spite of the derisive comments about emacs: EMACS stands for Esc-Meta-Alt-Ctrl-Shift, Eight Megs And Constantly Swapping, etc., emacs is the most popular editor in existence; it has been ported to almost every know computer system.

1.0 Introduction

This tutorial is aimed at novice emacs users who wish to become proficient in using GNU emacs. It attempts to explain:

- how emacs works;
- how to customize key-bindings (emacs is infinitely customizable);
- how to use the help-system (emacs has a built-in tutorial, and a hyper-text help-system);
- how to work in emacs without using the mouse (the keyboard is faster than the mouse);
- keyboard short-cuts;
- the macro-facility that automates repetitive tasks.

2.0 emacs: A LISP Interpreter

The emacs-binary, about 4.4 Mb for Solaris 2.3, is in essence an interpreter for a language called LISP[1], adding some basic functionality related to editing: moving the cursor, displaying a character (both on a dumb-terminal and graphically, under the X Window System), scrolling text, paging up/down, etc., and the worthless extra bytes for the 3D shaded-menu library.

2.1 "Hello World" in LISP

As proof that emacs is indeed a LISP interpreter, run emacs and note that the initial buffer (called `*scratch*`) is in LISP-mode. In the buffer type:

```
(print "Hello World")^J
```

and instead of pressing Return at the end of the line, type a `^J` (Ctrl-J)[2].

The GWBASIC editor's (for those old enough to remember it) interface is remarkably similar to that of Emacs. Recall that BASIC programs are interactively written and executed on-screen.

2.2 LISP Functions And Libraries

External LISP programs (about 3 Mb worth) are what give emacs its personality. These LISP programs also make emacs *extensible* and *customizable*, which means that additional functionality can be provided by any user who can write LISP programs.

It also means that emacs can be configured to emulate any other editor in existence and any editor that will exist-- emacs can be everything to everybody: play tetris (`getris`), read-mail (`rmail`, `VM`), manage your diary (`calendar`, `diary`), be a WWW-browser (`w3`), emulate `vi` editor (`vip-mode`), the Borland IDE, etc.

All the LISP functions are callable (input) as *meta-commands*; i.e. the command is typed in long-form and the LISP interpreter executes it.

2.2.1 Meta-Commands

All Meta-commands are invoked by first typing:

```
M-x
```

and then typing the full meta-command-name.

(The notation "`M-x`" means hold down the Meta-key[3], and type the character '`x`'; or equivalently, type the "Esc" key and then type the character '`x`'.)

1. LISP was invented by John McCarthy during the DarthMouth Summer Research Project in 1956. It reached the height of its popularity in the early '70's.

2. "Hello World" is printed twice because ^J is bound to eval-print-last-sexp, which evaluates the line (and outputs the 1st "Hello World" and then executes it (the second "Hello World").

For example, to "cursor-right" (i.e. move the cursor 1 character to the right), one *could* issue the meta-command:

```
M-x forward-char RET
```

(The notation `RET` means, type the key marked Return (or Enter) after typing the line.)

Similarly, to "cursor-down", one *could* issue the meta-command:

```
M-x next-line RET
```

Now, emacs wouldn't be a very popular editor with such a retarded interface. It is obviously less tiresome to type a single key-sequence rather than type a whole command-name to perform frequently used commands; which is why emacs allows meta-commands to be bound to keys. By default,

`forward-char` is bound to `C-f`
`backward-char` is bound to `C-b`
`next-line` is bound to `C-n`
`previous-line` is bound to `C-p`
`scroll-up` (page-down) is bound to `C-v`
`scroll-down` (page-up) is bound to `M-v`

(The "`C-`" notation means, hold-down the "Ctrl"-key and type the character that follows.)

So, pressing `C-f` is equivalent to having typed `M-x forward-character`.

3.0 emacs Is Customizable

Since emacs is (infinitely) customizable, users can always re-bind the standard keys to those of their own preferences; these customizations can be put in a `~/.emacs` file which emacs always reads upon start-up.

For example, if you wanted to re-bind the page-down key (by default, `C-v`), to `C-f`, then add the line:

```
(global-set-key "\C-f" 'scroll-up)
```

to your `~/.emacs`. The next time emacs start's-up, `C-f` will page-down; that's all there is to it!

3. The Meta-key is just another modifier key like Ctrl, Alt and Shift. Most high-performance workstations have a Meta-key (Sun Meta-keys are labeled with a diamond, and are found adjacent the Space-bar); the Esc-key is equivalent to the Meta-key on workstations that do not have them.

4.0 emacs Is Self-Documenting

When re-biding keys, it is important to ensure that a key-sequence selected for re-binding does not already perform some important function within emacs.

For example, `C-g` is the emacs key-sequence to abort a command. `C-g` is used to abort meta-commands and to get emacs' attention from a wayward running macro or a LISP function.

Emacs provides a way to query the key-bindings for a certain key-sequence, or for all the currently-bound keys (the current key-map).

To obtain the binding for a particular key-sequence, type:

```
M-x describe-key RET key-seq
```

Emacs will display a buffer showing the function that is bound to the key-sequence you typed-in and a brief description of the function's purpose.

Knowing the name of a LISP function, it is easy to determine the key-sequence that calls it by typing:

```
M-x describe-function RET fn-name RET
```

followed by the name of the function.

Emacs will display a buffer showing the function, the key-sequence, if any, ("unbound" is displayed if the function is not callable from a key-sequence) and a brief description of the function's purpose.

Both `describe-function` and `describe-key` are bound to handy key-sequences (see if you can determine what they are).

Given the hundreds of functions that are built-in to emacs, it is conceivable that, at some point you will remember either the key-sequence or the name of the function can be recalled; in such-cases, the `apropos` command may be used.

When the `apropos` meta-command is supplied a key-word, emacs displays all known functions and variables that contain that key-word.

For example, typing:

```
M-x apropos RET describe RET
```

emacs will display all the functions containing the word "`describe`" along with a one-line description of the each of the function's purpose (the following list is abridged):

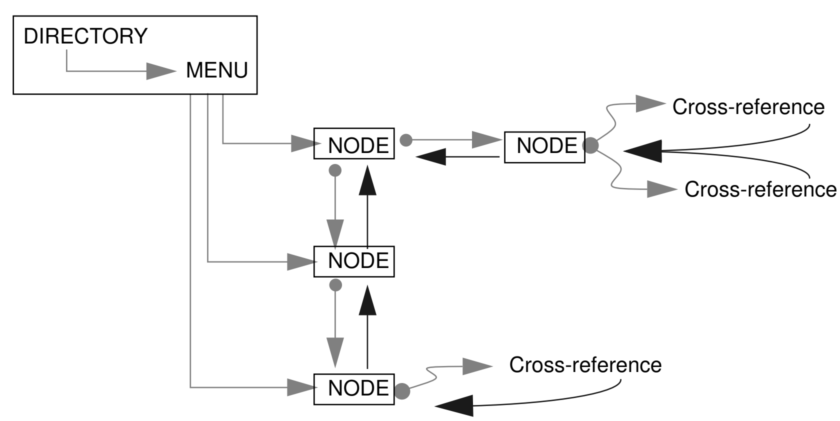
```
describe-function
 Function: Display the full documentation of FUNCTION
(a symbol).
describe-key
 Function: Display documentation of the function
invoked by KEY. KEY is a string.
describe-key-briefly
 Function: Print the name of the function KEY invokes.
KEY is a string.
describe-mode
 Function: Display documentation of current major mode
and minor modes.
describe-no-warranty
 Function: Display info on all the kinds of warranty
Emacs does NOT have.
```

4.1 The On-Line emacs Manual

For even more help, the complete emacs manual, with detailed help on all topics, is available on-line in a hyper-text format. To access it type:

```
M-x info RET
```

FIGURE 1. Structural overview of the Info help-system



The reader is encouraged to go through the brief introductory tutorial on using the info-help system, type:

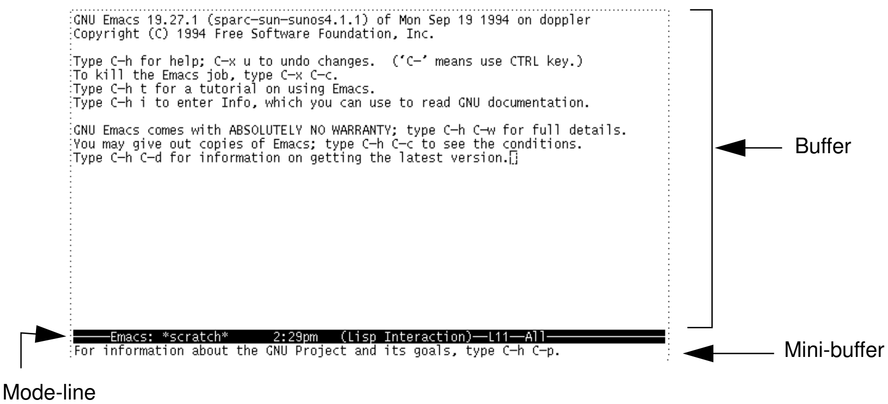
```
C-h i
```

Then follow the instructions (typing “h”) in the tutorial.

5.0 An emacs Window: the buffer, the mode-line and the mini-buffer

When emacs starts-up, it will display a window that contains a buffer, which holds the text; a mode-line, which describes what is going on in the buffer; and the mini-buffer, a one-line buffer at the bottom of the window.

FIGURE 2. The Initial emacs Window



5.1 The Mode-Line

The mode-line displays the name of the buffer and status information about the buffer. An important aspect of the mode-line is that it displays the (Major and Minor) mode[1] the buffer is currently operating in.

Examples of major-modes are: C-mode for editing C programs; LISP for editing LISP programs; TeX, or LaTeX; text-mode or the simplest, Fundamental-mode. Examples of minor-modes are: Fill-mode (where text automatically breaks at the fill-column; Compilation-mode when compiling programs; RCS-mode when files are under revision-control; etc.

The mode-line may be customized to display the date, the time, and the status of one's mail-box; the mode line usually appears in reverse-video.

1. Any mode may be toggled-on or off at any time by an appropriate meta-command:

M-x lisp-mode RET
M-x TeX-mode RET
M-x c-mode RET

5.2 The Mini-Buffer

The the mini-buffer is used to input meta-commands to emacs; emacs uses it to notify the user with messages. The mini-buffer may be exited at any time by typing `C-g` (this also aborts the pending meta-command).

5.2.1 Mini Buffer Completions

The mini-buffer is capable of providing specialized help on-demand, depending on the task being performed. The space-bar the TAB-key and '?' are two keys which perform special magic.

Pressing the Space-bar in the mini-buffer, will cause the mini-buffer to complete the partially typed command to the next syllable in the command. If the input is ambiguous (i.e. there is more than one meta-command that begins with what is typed in the mini-buffer), a buffer will appear, listing the possible completions.

For example, typing "`spe`" and then Space-bar will cause the mini-buffer to complete the word "`spell-`".

Pressing the TAB key will cause the mini-buffer to unambiguously complete the current input. If the input is ambiguous (i.e. there is more than one meta-command that begins with what is typed in the mini-buffer), a buffer will appear, listing the possible completions.

Continuing the previous example, typing TAB after "`M-x spell-`" is displayed, will cause a buffer to appear, listing the possible completions (i.e. commands that have the word "spell") for the spell meta-command:

```
In this buffer, type RET to select the comple-
tion near point.
Possible completions are:
spell-buffer            spell-region
spell-string            spell-word
```

Pressing '?' at anytime in the mini-buffer will cause a buffer to display a list of possible completions for whatever action is being performed in the mini-buffer. If, for example, in the middle of typing a filename in the mini-buffer (i.e. loading a file with C-x C-f), '?' is pressed, a list of possible completions (if any) for that partially-typed file-name will be supplied.

Mini-buffer completions are very handy for recalling the exact syntax of seldom-used meta-commands and accessing strangely-named files.

6.0 The Keyboard Is Quicker Than The Mouse

The original Emacs was written to work on a simple dumb-terminal; it was only later that support for pull-down menus, for graphics and for the mouse was added.

Human-factors studies have proven that any task can be performed faster using only the keyboard (this is why user-interfaces provide keyboard accelerators, in addition to mouse-based menus). The emacs GUI is an unnecessary extension and only serves to steepen the learning-curve (i.e. make it easy) for the novice-user.

The following sections describe alternatives to the urge to reach for the mouse; it is possible to use emacs without ever touching the mouse

6.1 Navigating emacs Buffers

`C-x` is the command-prefix for all built-in emacs commands. This means that all emacs commands begin with C-x followed by some other key or keys.

The following table lists some useful key-bindings that may be used to manipulate emacs windows:

TABLE 1. Window-manipulation key-bindings

Key	Description
`C-x 2`	will split the current buffer horizontally into 2 separate buffers. Different parts of the same file, or a different file may be loaded into the other buffer
C-x 3	will split the current buffer vertically into 2 separate buffers
`C-x o`	("o" is for "other") will switch (the cursor) to the other buffer (including the mini-buffer). If there are more than 2 buffers, multiple `C-x o`'s will cycle through all the available buffers. (If the graphical emacs is being used, then click with the mouse-pointer in the buffer you wish to switch to.) User's should note that using `C-x o` to switch buffers is faster than reaching for the mouse and clicking in the buffer to be switch to.
`C-x b`	will prompt for a buffer-name to switch to. This is the quickest way to change to a particular buffer when the name is known (note that buffer-name completion will work when supplying the name)
`C-x 0`	will remove the current buffer from the display if there is one or more buffer(s) being displayed; the other buffers will enlarge to occupy the space of the buffer that was un-displayed
`C-x 1`	will un-display all but the current buffer; i.e. the current buffer is the sole (i.e. one) buffer that is displayed

(Note that with `C-x 0` and `C-x 1`, the buffers are not deleted; they are merely not displayed.)

6.2 Operating On "Other" Buffers

`C-x 4` is a command-prefix that operates on a buffer other than the current.

Given a buffer-name, when prompted, the command `C-x 4 b` will split the current window and display the specified buffer in the other window.

Similarly, `C-x 4 C-f` will load a file into "another" buffer; this can be used to load a second file you wish to refer to, in another buffer, without losing you place in the current buffer.

When there are 2 buffers present, `M-C-v` can be used to scroll the other buffer without switching to it; for example, if a man-page is loaded, using:

```
M-x man RET topic RET
```

for reference whilst editing some program-code, the man-page buffer can be scrolled without losing the position in the current buffer.

6.3 Short-cuts And Speed-ups For Editing Text

This list of practical tips is useful when editing text (in `text-mode` or `fundamental-mode`).

`M-u`, `M-l`, `M-c` will uppercase, lowercase and capitalize (respectively) the word after point.

A paragraph can be "filled"; i.e. long-lines of text are broken at the fill-margin, with `M-q`. In `auto-fill-mode`, this happens automatically. auto-fill-mode can be automatically toggled on when entering text-mode with the following:

```
(setq text-mode-hook
 '(lambda () (auto-fill-mode 1) (set-fill-column
69)))
```

The fill-column is also set to a reasonable number (69) at the same time.

The fill-column (i.e. right margin) can be set by moving point to the column you wish the right-margin to be and typing `C-x f`. The left-margin can be set by first indenting the text to the correct column and typing `C-x .` (that's C-x followed by a a period); emacs will fill the paragraph with whatever text precedes the cursor when `M-q` is typed.

The following keys-bindings are also very useful and should be committed to memory:

TABLE 2. Keyboard short-cuts

Key	Description
`C-a`	move to the beginning of the line
C-e	move to the end of the line
`C-t`	transpose the letter just before the cursor with the letter proceeding it (occasionally, one types "`teh`" instead of "`the`"; pressing `C-t` just after typing "`teh`" corrects it); `M-t` transpose the 2 words proceeding the cursor (the cursor must be on the second-word)
`M-<`	go to the beginning of the buffer

TABLE 2. Keyboard short-cuts

Key	Description
`M->`	go to the end of the buffer
`C-d`	delete the character after the cursor; M-d delete the word after the cursor (`C-y` will yank back the word)
`C-k`	delete from the cursor to the end of the line (`C-y` will yank back the deletion)

6.4 Exiting Or Suspending emacs

The prevailing custom is to run an emacs when logging-in and exit it when logging-out. This custom is a testimonial to emacs' versatility: *everything* can be done within emacs. If a shell is ever needed, you can get it in an emacs-buffer with:

```
M-x shell RET
```

As a last resort, emacs can be suspended with `C-z`; this will return you to the shell and allow you to run other commands. To resume emacs; either type "`fg`" or type "`emacs`" (`csh`, `zsh`, `bash` support this feature).

`C-x C-c` will completely exit emacs; if there are any unsaved buffers, emacs will prompt, asking to save any unsaved buffers.

6.5 The mark, the point and the region

The area defined by a location known as "the mark" and the location of the cursor, called "the point", is know as "the region".

Once a region is defined in a buffer, various operations may be performed on it. Apropos on the word "region" returns, among other functions:

```
capitalize-region (not bound to any keys)
comment-region (not bound to any keys)
count-lines-region ESC =
delete-region menu-bar edit clear
ispell-region menu-bar edit spell
ispell-region
lpr-region (not bound to any keys)
```

To define the region, move the cursor to the location in the buffer where the region is to begin and type `C-@` (it is customary to bind `C-@` to `C-2` since it is used so often and is faster then holding-down Shift and typing `C-@`); emacs will respond the "Mark set" in the mini-buffer. Once the mark is set the text between the mark and the cursor, at any time after the mark is set, is understood to be the region.

The region can also be defined by clicking and dragging the mouse over the text to be marked.

6.6 Cut and Paste

Once the region is defined, `C-w` may be used to cut the region from the buffer.

`C-y` ("yank") will paste the text back into the buffer.

6.7 Copy and Paste

Once the region is defined, `M-w` may be use to copy the region from the buffer. `C-y` will paste the region back into the buffer. Multiple invocations of `C-y` will paste the contents of the region multiple times.

6.8 The Kill-Ring

Whenever a significant portion of the buffer is deleted from the buffer (via either `C-w` or `M-d`), emacs saves it in a kill-buffer (this is in case you wish to undo the deletion).

In actuality, emacs keeps track of several "cuts" in storage-buffers known collectively as the kill-ring. The most-recently deleted portion of text may be retrieved (pasted) with `C-y`.

Previously deleted text saved in the kill-ring may be retrieved using `M-y` when it immediately precedes `C-y`. Multiple invocations of `M-y` will cycle through the kill-ring replacing the last-yanked portion of text, with the next-portion in the kill-ring.

6.9 The Mark-ring

When-ever a mark is set with `C-@`, emacs remembers the location of that mark in the buffer (the location is saved in a mark-ring). It is possible to return to locations thus marked using `C-x C-@`. Repeat `C-x C-@` to cycle through all marked locations in the current-buffer. If the mark was never set in the buffer, `C-x C-@` will cause emacs to report "`Mark is not set`" in the mini-buffer.

6.10 Undoing Mistakes And Re-doing Commands

`C-x u` will undo the last edit; repeated invocations of `C-x u` will undo the edits even further. Emacs does not keep a complete history of undo-information; there are limits. Undo-information is purged during a garbage-collection, whenever a new file is loaded, etc. On Sun-keyboards, the key marked "Undo" on the left-keypad is bound to execute `C-x u`.

6.11 Macros

Since emacs is only able to recall meta-commands, the only way to re-do simple-editing commands is to "record" them and "play them back" as a macro. Emacs' macro-facility is a remember-what-I-do-and-then-repeat-it type. `C-x (` tells emacs to begin recording a macro, and `C-x )` tells emacs to stop recording it. Emacs remembers exactly what you type between `C-x (` and `C-x )`. To execute the macro, type `C-x e` (for execute).

A useful tip when recording macros is to remember to finish the macro with a command that will put the cursor in a convenient spot that allows the macro to be re-executed; e.g. when operating on lines of text, end the macro with `C-n C-a`, which will place the cursor at the beginning (`C-a`) of the next line (`C-n`).

6.11.1 Saving Macros

If a macro is especially useful, it can be saved in your `~/.emacs` file. After recording a macro you wish to save, you must give it a name:

```
M-x name-last-kbd-macro RET
```

Once named,

```
M-x insert-kbd-macro RET name RET
```

is used to insert the macro into your `~/.emacs` (i.e. load `~/.emacs` into a buffer, make that buffer the current-buffer and type `M-x insert-kbd-macro`. Emacs will insert something like:

```
(fset 'c-comment-line "^A/*^E*/")
```

The macro (named `c-comment-line`) can then be bound to a key-sequence with:

```
(global-set-key "\C-c+" 'c-comment-line)
```

Now, define and save a macro to un-comment a line of C-code (call it `c-uncomment-line`) and bind it to `C-c-` (that's Ctrl-C followed by a minus)

6.12 The Numeric Command-Prefix

Every emacs command (both meta-commands and key-bindings) may be repeatedly executed if preceded by a numeric prefix (also called "*arg*"); i.e. `C-u` optionally followed by a number.

For example, if you wished to insert a line of dashes (79 chars) in a buffer:

```
C-u 79 -
```

would do it faster than typing 79 dashes sequentially.

As another example, a recorded macro can be executed *N* times if it is preceded by a numeric prefix; i.e. `C-u 5 C-x e` will execute the last-recorded macro five times.

And one final example shows how five words following the cursor can be deleted with:

```
C-u 5 M-d
```

(remember that `M-d` deletes one word).

6.13 Searching & Replacing

Simple searching is accomplished with `C-s` followed by the string you wish to search-for (this is an incremental search; emacs will match every portion of the string you type, as it is typed); use `C-r` to search backwards through the buffer. To search for the next-occurrence of the string type `C-s` (or `C-r`) while in search-mode.

Type Return to end the search or `C-g` to abort the search and return to the location where the search began.

`M-%` (you'll have to memorize this because I cannot think of a handy mnemonic to help you remember this binding) is bound to `query-replace` (search and replace); it requests the string to search-for and the string to replace the text with. `query-replace` then proceeds to find all occurrences of the string you are searching for and prompts whether it is to be replaced. Type "`!`" while in query-replace-mode to unconditionally replace all strings in the current buffer otherwise answer appropriately to skip over occurrences you do not wish changed.

7.0 Interesting Features

These are interesting features that you will probably find indispensable in day-to-day use of emacs.

7.1 Grep on files

It is possible to grep for keywords in a set of files with M-x grep. Emacs runs grep as a child process and displays a buffer will all the hits. C-x ' may then be used to visit each file where the keyword occurs; The syntax of grep is:

`M-x grep RET` *`keyword files`* `RET`

For example to search for all the source-code files containing the word `struct`:

```
M-x grep RET struct *.[ch] RET
```

7.2 Edit remote files

Emacs can load files that are located on another host and even on another network; ftp is used to retrieve the file into a buffer. The LISP package that permits this is called `ange-ftp`; emacs loads this package by default.

To retrieve a file, type:

```
C-x C-f /user@host.network:/path/to/file
```

Note the syntax for specifying the file:

- a "/" must precede the used-name (a Unix user-id)
- `host.network` must be a fully-qualified domain-name and must be followed by a ":"

Also note that file-name completion will work as usual.

This technique may be used to ftp files from archives.

7.3 The Directory Editor: dired

The directory-editor provides an interface to all the functions one would normally perform in a shell. Operations may be performed either on one file or on a set of files which are marked (the characters in parentheses invoke the respective command):

- load files (`f`)
- delete files, back-ups and auto-saves (`d, ~, #`); use '`x`' to actually delete them;
- change permissions (`G` (group), `M` (modes);
- copy (`C`) or re-name (`R`) files;
- browse files (`v`); etc.

Dired (`M-x dired` RET) is invoked with `C-x d`.

7.4 Piping region through a shell-command

Once a region is defined, it may be piped through any shell-command; the output is placed in another buffer (called `*Shell Command Output*`).

For example, the following list of user-ids's:

```
pott
leroy
ssaul
sblack
jdear
rehrlich
```

```
jnaughto
pmcquade
```

can be easily sorted by first defining the region around them, and then typing (`shell-command-on-region`):

```
M-| RET sort RET
```

If M-| is given a prefix, then the output of the shell-command will replace the input (i.e. the region).

8.0 Loading external LISP packages

Emacs loads external LISP packages (programs) when they are needed. Packages may also be loaded on-demand or deferred until they are needed.

LISP packages may be compiled (called byte-compilation) to speed-up loading since. A package names hilight.el is byte-compiled into a file called hilight.elc. Whenever a package is to be loaded, emacs first checks if a compiled-version exists and loads it. Failing that, emacs loads the original source and compiles it.

The default load-path (similar to the shell's PATH variable) is:

```
/usr/local/lib/emacs/19.27/lisp
```

followed by

```
/usr/local/lib/emacs/site-lisp
```

Your own LISP-directory can be added to the search-path (usually at the head of the list):

```
(setq load-path (cons
(expand-file-name "~/lisp") load-path ))
```

8.1 The semantics of require vs. load vs. autoload

```
(require 'foo-bar)
```

This will ensure that the feature foo-bar is available. If it is not currently available, it will do a `(load-library "foo-bar")` and then check again if feature foo-bar is available. If so, fine, if not, it issues a warning.

```
(autoload 'foo-bar "foo-bar" "Foo Bar ver1.2" t)
```

This will `(load-library "foo-bar")` the first time function `(foo-bar)` is called. This is good for not-so-often used features since it defers loading the package until it is actually used.

```
(load "foo-bar")
```

This just goes out and loads the file. It doesn't care if it is already loaded.