Evil

Release 1.14.0

Apr 12, 2022

The Evil team thanks everyone at gmane.emacs.vim-emulation for their feedback and contributions.

Contents

CHAPTER 1

Overview

Evil is an extensible vi layer for Emacs. It emulates the main features of Vim,[1] turning Emacs into a modal editor. Like Emacs in general, Evil is extensible in Emacs Lisp.

1.1 Installation via package.el

Evil is available as a package from MELPA stable, MELPA unstable and NonGNU ELPA. This is the recommended way of installing Evil.

To set up *package.el* to work with one of the MELPA repositories, you can follow the instructions on melpa.org.

Alternatively you can use NonGNU ELPA. It is part of the default package archives as of Emacs 28. For older Emacs versions you'll need to add it yourself:

```
(add-to-list 'package-archives
             (cons "nongnu" (format "http%s://elpa.nongnu.org/nongnu/"
                                    (if (gnutls-available-p) "s" ""))))
```

Once that is done, you can execute the following commands:

```
M-x package-refresh-contents
M-x package-install RET evil RET
```

Finally, add the following lines to your Emacs init file:

```
(require 'evil)
(evil-mode 1)
```

[1] Vim is the most popular version of *vi*, a modal text editor with many implementations. Vim also adds some functions of its own, like visual selection and text objects. For more information see the official Vim website.

1.2 Manual installation

First, install *goto-chg* and *cl-lib*. If you have an Emacs version of 24.3 or newer, you should already have *cl-lib*.

Evil lives in a git repository. To download Evil, do:

```
git clone --depth 1 https://github.com/emacs-evil/evil.git
```

Then add the following lines to your Emacs init file:

```
(add-to-list 'load-path "path/to/evil")
(require 'evil)
(evil-mode 1)
```

Ensure that your replace `path/to/evil` with the actual path to where you cloned Evil.

1.3 Modes and states

The next time Emacs is started, it will come up in *normal state*, denoted by `<N>` in the mode line. This is where the main vi bindings are defined. Note that you can always disable normal state with `C-z`, which switches to an "Emacs state" (denoted by `<E>`) in which vi keys are completely disabled. Press `C-z` again to switch back to normal state.

state Evil uses the term *state* for what is called a "mode" in regular vi usage, because *modes* are understood in Emacs terms to mean something else.

Evil defines a number of states by default:

normal state (<N>) This is the default "resting state" of Evil, in which the main body of vi bindings are defined.

insert state (<I>) This is the state for insertion of text, where non-modified keys will insert the corresponding character in the buffer.

visual state (<V>) A state for selecting text regions. Motions are available for modifying the selected region, and operators are available for acting on it.

replace state (<R>) A special state mostly similar to insert state, except it replaces text instead of inserting.

operator-pending state (<O>) A special state entered after launching an operator, but before specifying the corresponding motion or text object.

motion state (<M>) A special state useful for buffers that are read-only, where motions are available but editing operations are not.

Emacs state (<E>) A state that as closely as possible mimics default Emacs behaviour, by eliminating all vi bindings, except for `C-z`, to re-enter normal state.

CHAPTER 2

Settings

Evil's behaviour can be adjusted by setting some variables. The list of all available variables and their current values can be inspected by doing:

```
M-x customize-group RET evil RET
```

To change the value of a variable, you can use this interface, or add a `setq` form to your Emacs init file, preferably before Evil is loaded.[1]

```
(setq evil-shift-width 0)
;; Load Evil
(require 'evil)
```

What follows is a non-exhaustive list of the most relevant customization options.

2.1 The initial state

The initial state of a buffer is determined by its major mode. Evil maintains an association between major modes and their corresponding states, which is most easily modified using the function `evil-set-initial-state`.

`(evil-set-initial-state MODE STATE)`
Set the initial state for major mode *MODE* to *STATE*. This is the state the buffer comes up in.

If no state can be found, Evil uses the default initial state.

`evil-default-state`
The default Evil state. This is the state a buffer starts in when it is not otherwise configured (see `evil-set-initial-state` and `evil-buffer-regexps`). The value may be one of `normal`, `insert`, `visual`, `replace`, `operator`, `motion` and `emacs`.

Default: `normal`

[1] Strictly speaking, the order only matters if the variable affects the way Evil is loaded. This is the case with some variables.

Alternatively, it is possible to select the initial state based on the buffer *name* rather than its major mode. This is checked first, so it takes precedence over the other methods for setting the state.

evil-buffer-regexps
Regular expressions determining the initial state for a buffer. Entries have the form `(REGEXP . STATE)`, where *REGEXP* is a regular expression matching the buffer's name and *STATE* is one of `normal`, `insert`, `visual`, `replace`, `operator`, `motion`, `emacs` and `nil`. If *STATE* is `nil`, Evil is disabled in the buffer.

Default: `(("^ \\*load\\*"))`

2.2 Keybindings and other behaviour

Evil comes with a rich system for modifying its key bindings *Keymaps*. For the most common tweaks, the following variables are available.

evil-toggle-key
The key used to change to and from Emacs state. Must be readable by `read-kbd-macro`. For example: "C-z".

Default: `"C-z"`

evil-want-C-i-jump
Whether `C-i` jumps forward in the jump list (like Vim). Otherwise, `C-i` inserts a tab character.

Default: `t`

evil-want-C-u-delete
Whether `C-u` deletes back to indentation in insert state. Otherwise, `C-u` applies a prefix argument. The binding of `C-u` mirrors Emacs behaviour by default due to the relative ubiquity of prefix arguments.

Default: `nil`

evil-want-C-u-scroll
Whether `C-u` scrolls up (like Vim). Otherwise, `C-u` applies a prefix argument. The binding of `C-u` mirrors Emacs behaviour by default due to the relative ubiquity of prefix arguments.

Default: `nil`

evil-want-C-d-scroll
Whether `C-d` scrolls down (like Vim).

Default: `t`

evil-want-C-w-delete
Whether `C-w` deletes a word in Insert state.

Default: `t`

evil-want-C-w-in-emacs-state
Whether `C-w` prefixes windows commands in Emacs state.

Default: `nil`

evil-want-Y-yank-to-eol
Whether `Y` yanks to the end of the line. The default behavior is to yank the whole line, like Vim.

Default: `nil`

evil-disable-insert-state-bindings
Whether insert state bindings should be used. Bindings for escape, delete and `evil-toggle-key` are always available. If this is non-nil, default Emacs bindings are by and large accessible in insert state.

Default: `nil`

2.3 Search

evil-search-module
The search module to be used. May be either `isearch`, for Emacs' isearch module, or `evil-search`, for Evil's own interactive search module. N.b. changing this will not affect keybindings. To swap out relevant keybindings, see `evil-select-search-module` function.

Default: `isearch`

evil-regexp-search
Whether to use regular expressions for searching in `/` and `?`.

Default: `t`

evil-search-wrap
Whether search with `/` and `?` wraps around the buffer. If this is non-nil, search stops at the buffer boundaries.

Default: `t`

evil-flash-delay
Time in seconds to flash search matches after `n` and `N`.

Default: `2`

evil-ex-hl-update-delay
Time in seconds of idle before updating search highlighting. Setting this to a period shorter than that of keyboard's repeat rate allows highlights to update while scrolling.

Default: `0.02`

2.4 Indentation

evil-auto-indent
Whether to auto-indent when opening lines with `o` and `O`.

Default: `t`, buffer-local

evil-shift-width
The number of columns by which a line is shifted. This applies to the shifting operators > and <.

Default: `4`, buffer-local

evil-shift-round
Whether shifting rounds to the nearest multiple. If non-nil, > and < adjust line indentation to the nearest multiple of `evil-shift-width`.

Default: `t`, buffer-local

evil-indent-convert-tabs
If non-nil, the = operator converts between leading tabs and spaces. Whether tabs are converted to spaces or vice versa depends on the value of `indent-tabs-mode`.

Default: `t`

2.5 Cursor movement

In standard Emacs terms, the cursor is generally understood to be located between two characters. In Vim, and therefore also Evil, this is the case in insert state, but in other states the cursor is understood to be *on* a character, and that this character is not a newline.

Forcing this behaviour in Emacs is the source of some potentially surprising results (especially for traditional Emacs users—users used to Vim may find the default behavior to their satisfaction). Many of them can be tweaked using the following variables.

evil-repeat-move-cursor
Whether repeating commands with . may move the cursor. If nil, the original cursor position is preserved, even if the command normally would have moved the cursor.

Default: `t`

evil-move-cursor-back
Whether the cursor is moved backwards when exiting insert state. If non-nil, the cursor moves "backwards" when exiting insert state, so that it ends up on the character to the left. Otherwise it remains in place, on the character to the right.

Default: `t`

evil-move-beyond-eol
Whether the cursor can move past the end of the line. If non-nil, the cursor is allowed to move one character past the end of the line, as in Emacs.

Default: `nil`

evil-cross-lines
Whether horizontal motions may move to other lines. If non-nil, certain motions that conventionally operate in a single line may move the cursor to other lines. Otherwise, they are restricted to the current line. This applies to `h`, `SPC`, `f`, `F`, `t`, `T`, `~`.

Default: `nil`

evil-respect-visual-line-mode
Whether movement commands respect `visual-line-mode`. If non-nil, `visual-line-mode` is generally respected when it is on. In this case, motions such as `j` and `k` navigate by visual lines (on the screen) rather than "physical" lines (defined by newline characters). If nil, the setting of `visual-line-mode` is ignored.

This variable must be set before Evil is loaded.

Default: `nil`

evil-track-eol
Whether `$` "sticks" the cursor to the end of the line. If non-nil, vertical motions after `$` maintain the cursor at the end of the line, even if the target line is longer. This is analogous to `track-eol`, but respects Evil's interpretation of end-of-line.

Default: `t`

evil-start-of-line
Analogue of vim's `startofline`. If nil, preserve column when making relevant movements of the cursor. Otherwise, move the cursor to the start of the line.

Default: `nil`

2.6 Cursor display

A state may change the appearance of the cursor. Use the variable `evil-default-cursor` to set the default cursor, and the variables `evil-normal-state-cursor`, `evil-insert-state-cursor` etc. to set the cursors for specific states. The acceptable values for all of them are the same.

evil-default-cursor
The default cursor. May be a cursor type as per `cursor-type`, a color string as passed to `set-cursor-color`, a zero-argument function for changing the cursor, or a list of the above.

Default: `t`

2.7 Window management

evil-auto-balance-windows
If non-nil window creation and deletion trigger rebalancing.

Default: `t`

evil-split-window-below
If non-nil split windows are created below.

Default: `nil`

evil-vsplit-window-right
If non-nil vertically split windows with are created to the right.

Default: `nil`

2.8 Parenthesis highlighting

These settings concern the integration between Evil and `show-paren-mode`. They take no effect if this mode is not enabled.

evil-show-paren-range
The minimal distance between point and a parenthesis which causes the parenthesis to be highlighted.

Default: `0`

evil-highlight-closing-paren-at-point-states
The states in which the closing parenthesis at point should be highlighted. All states listed here highlight the closing parenthesis at point (which is Vim's default behavior). All others highlight the parenthesis before point (which is Emacs default behavior). If this list contains the symbol `not` then its meaning is inverted, i.e. all states listed here highlight the closing parenthesis before point.

Default: `(not emacs insert replace)`

2.9 Miscellaneous

evil-want-fine-undo
Whether actions are undone in several steps. There are two possible choices: nil ("no") means that all changes made during insert state, including a possible delete after a change operation, are collected in a single undo step. Non-nil ("yes") means that undo steps are determined according to Emacs heuristics, and no attempt is made to aggregate changes.

For backward compatibility purposes, the value `fine` is interpreted as `nil`. This option was removed because it did not work consistently.

Default: `nil`

evil-undo-system
Undo system Evil should use. If equal to `undo-tree` or `undo-fu`, those packages must be installed. If equal to `undo-tree`, `undo-tree-mode` must also be activated. If equal to `undo-redo`, Evil uses commands natively available in Emacs 28.

Default: `nil`

evil-backspace-join-lines
Whether backward delete in insert state may join lines.

Default: `t`

evil-kbd-macro-suppress-motion-error
Whether left/right motions signal errors in keyboard macros. This variable only affects beginning-of-line or end-of-line errors regarding the motions `h` and `SPC` respectively. This may be desired since such errors cause macro definition or execution to be terminated. There are four possibilities:

- `record`: errors are suppressed when recording macros, but not when replaying them.
- `replay`: errors are suppressed when replaying macros, but not when recording them.
- `t`: errors are suppressed in both cases.
- `nil`: errors are never suppressed.

Default: `nil`

evil-mode-line-format
The position of the state tag in the mode line. If set to `before` or `after`, the tag is placed at the beginning or the end of the mode-line, respectively. If nil, there is no tag. Otherwise it should be a cons cell `(WHERE . WHICH)`, where *WHERE* is either `before` or `after`, and *WHICH* is a symbol in `mode-line-format`. The tag is then placed before or after that symbol, respectively.

Default: `before`

evil-mouse-word
The *thing-at-point* symbol for double click selection. The double-click starts visual state in a special word selection mode. This symbol is used to determine the words to be selected. Possible values are `evil-word` or `evil-WORD`.

Default: `evil-word`

evil-bigword
The set of characters to be interpreted as WORD boundaries. This is enclosed with square brackets and used as a regular expression. By default, whitespace characters are considered WORD boundaries.

Default: `"^ \t\r\n"`, buffer-local

evil-esc-delay
The time, in seconds, to wait for another key after escape. If no further event arrives during this time, the event is translated to `ESC`. Otherwise, it is translated according to `input-decode-map`. This does not apply in Emacs state, and may also be inhibited by setting `evil-inhibit-esc`.

Default: `0.01`

evil-intercept-esc
Whether Evil should intercept the escape key. In the terminal, escape and a meta key sequence both generate the same event. In order to distingush these, Evil uses `input-decode-map`. It is not necessary to do this in a graphical Emacs session. However, if you prefer to use `C-[` as escape (which is identical

to the terminal escape key code), this interception must also happen in graphical Emacs sessions. Set this variable to `always`, `t` (only in the terminal) or nil (never intercept).

Default: `always`

evil-kill-on-visual-paste
Whether pasting in visual state adds the replaced text to the kill ring, making it the default for the next paste. The default, replicates the default Vim behavior.

Default: `t`

evil-echo-state
Whether to signal the current state in the echo area.

Default: `t`

evil-complete-all-buffers
Whether completion looks for matches in all buffers. This applies to `C-n` and `C-p` in insert state.

Default: `t`

evil-want-empty-ex-last-command
Whether to default to evil-ex-previous-command at empty ex prompt.

Default: `t`

CHAPTER 3

Keymaps

Evil's key bindings are stored in a number of different keymaps. Each state has a *global keymap*, where the default bindings for that state are stored. They are named `evil-normal-state-map`, `evil-insert-state-map`, and so on. The bindings in these maps are visible in all buffers currently in the corresponding state.

These keymaps function like ordinary Emacs keymaps and may be modified using the Emacs function `define-key`:

```
(define-key evil-normal-state-map (kbd "w") 'some-function)
```

This binds the key `w` to the command `some-function` in normal state. The use of `kbd` is optional for simple key sequences, like this one, but recommended in general.

Most of Evil's bindings are defined in the file `evil-maps.el`.

To facilitate shared keybindings between states, some states may activate keybindings from other states as well. For example, motion state bindings are visible in normal and visual state, and normal state bindings are also visible in visual state.

Each state also has a *buffer-local keymap* which is specific to the current buffer, and which takes precedence over the global keymap. These maps are most suitably modified by a mode hook. They are named `evil-normal-state-local-map`, `evil-insert-state-local-map`, and so on.

```
(add-hook 'some-mode-hook
          (lambda ()
            (define-key evil-normal-state-local-map
                        (kbd "w") 'some-function)))
```

For convenience, the functions `evil-global-set-key` and `evil-local-set-key` are available for setting global and local state keys.

`(evil-global-set-key STATE KEY DEF)`
: Bind *KEY* to *DEF* in *STATE*.

`(evil-local-set-key STATE KEY DEF)`
: Bind *KEY* to *DEF* in *STATE* in the current buffer.

The above examples could therefore have been written as follows:

```
(evil-global-set-key 'normal (kbd "w") 'some-function)

(add-hook 'some-mode-hook
          (lambda ()
            (evil-local-set-key 'normal (kbd "w") 'some-function)))
```

3.1 evil-define-key

Evil provides the macro `evil-define-key` for adding state bindings to ordinary keymaps. It is quite powerful, and is the preferred method for fine-tuning bindings to activate in specific circumstances.

`(evil-define-key STATE KEYMAP KEY DEF [BINDINGS...])`

Create a *STATE* binding from *KEY* to *DEF* for *KEYMAP*. *STATE* is one of `normal`, `insert`, `visual`, `replace`, `operator`, `motion`, `emacs`, or a list of one or more of these. Omitting a state by using `nil` corresponds to a standard Emacs binding using `define-key`. The remaining arguments are like those of `define-key`. For example:

```
(evil-define-key 'normal foo-map "a" 'bar)
```

This creates a binding from a to `bar` in normal state, which is active whenever `foo-map` is active. Using nil for the state, the following lead to identical bindings:

```
(evil-define-key nil foo-map "a" 'bar)
(define-key foo-map "a" 'bar)
```

It is possible to specify multiple states and/or bindings at once:

```
(evil-define-key '(normal visual) foo-map
  "a" 'bar
  "b" 'foo)
```

If `foo-map` has not been initialized yet, this macro adds an entry to `after-load-functions`, delaying execution as necessary.

KEYMAP may also be a quoted symbol. If the symbol is `global`, the global evil keymap corresponding to the state(s) is used, meaning the following lead to identical bindings:

```
(evil-define-key 'normal 'global "a" 'bar)
(evil-global-set-key 'normal "a" 'bar)
```

The symbol `local` may also be used, which corresponds to using `evil-local-set-key`. If a quoted symbol is used that is not `global` or `local`, it is assumed to be the name of a minor mode, in which case `evil-define-minor-mode-key` is used.

There follows a brief overview of the main functions of this macro.

- Define a binding in a given state

```
(evil-define-key 'state 'global (kbd "key") 'target)
```

- Define a binding in a given state in the current buffer

```
(evil-define-key 'state 'local (kbd "key") 'target)
```

- Define a binding in a given state under the *foo-mode* major mode.

```
(evil-define-key 'state foo-mode-map (kbd "key") 'target)
```

Note that `foo-mode-map` is unquoted, and that this form is safe before `foo-mode-map` is loaded.

- Define a binding in a given state under the *bar-mode* minor mode.

```
(evil-define-key 'state 'bar-mode (kbd "key") 'target)
```

Note that `bar-mode` is quoted, and that this form is safe before `bar-mode` is loaded.

The macro `evil-define-key` can be used to augment existing modes with state bindings, as well as creating packages with custom bindings. For example, the following will create a minor mode `foo-mode` with normal state bindings for the keys `w` and `e`:

```
(define-minor-mode foo-mode
  "Foo mode."
  :keymap (make-sparse-keymap))

(evil-define-key 'normal 'foo-mode "w" 'bar)
(evil-define-key 'normal 'foo-mode "e" 'baz)
```

This minor mode can then be enabled in any buffers where the custom bindings are desired:

```
(add-hook 'text-mode-hook 'foo-mode)  ; enable alongside text-mode
```

3.2 Leader keys

Evil supports a simple implementation of Vim's *leader* keys. To bind a function to a leader key you can use the expression `<leader>` in a key mapping, e.g.

```
(evil-define-key 'normal 'global (kbd "<leader>fs") 'save-buffer)
```

Likewise, you can use the expression `<localleader>` to mimic Vim's local leader, which is designed for mode-specific key bindings.

You can use the function `evil-set-leader` to designate which key acts as the leader and the local leader.

(evil-set-leader STATE KEY [LOCALLEADER])

Set *KEY* to trigger leader bindings in *STATE*. *KEY* should be in the form produced by `kbd`. *STATE* is one of `normal`, `insert`, `visual`, `replace`, `operator`, `motion`, `emacs`, a list of one or more of these, or `nil`, which means all of the above. If *LOCALLEADER* is non-nil, set the local leader instead.

CHAPTER 4

Hooks

A *hook* is a list of functions that are executed when certain events happen. Hooks are modified with the Emacs function `add-hook`. Evil provides entry and exit hooks for all its states. For example, when switching from normal state to insert state, all functions in `evil-normal-state-exit-hook` and `evil-insert-state-entry-hook` are executed.

It is guaranteed that the exit hook will be executed before the entry hook on all state switches.

During the hook execution, the variables `evil-next-state` and `evil-previous-state` contain information about the states being switched to and from, respectively.

CHAPTER 5

Extension

The main functionality of Evil is implemented in terms of reusable macros. Package writers can use these to define new commands.

5.1 Motions

A *motion* is a command which moves the cursor, such as `w` or `e`. Motions are defined with the macro `evil-define-motion`. Motions not defined in this way should be declared with `evil-declare-motion`.

`(evil-declare-motion COMMAND)`
Declare *COMMAND* to be a movement function. This ensures that it behaves correctly in visual state.

`(evil-define-motion MOTION (COUNT ARGS...) DOC [[KEY VALUE]...] BODY...)`
Define a motion command *MOTION*. *ARGS* is a list of arguments. Motions can have any number of arguments, but the first (if any) has the predefined meaning of count. *BODY* must execute the motion by moving point.

Optional keyword arguments are:

- `:type` - determines how the motion works after an operator (one of `inclusive`, `line`, `block` and `exclusive`, or a self-defined motion type)
- `:jump` - if non-nil, the previous position is stored in the jump list, so that it can be restored with `C-o`

For example, this is a motion that moves the cursor forward by a number of characters:

```
(evil-define-motion foo-forward (count)
  "Move to the right by COUNT characters."
  :type inclusive
  (forward-char (or count 1)))
```

The *type* of a motion determines how it works when used together with an operator. Inclusive motions include the endpoint in the range being operated on, while exclusive motions do not. Line motions extend

the whole range to linewise positions, effectively behaving as if the endpoint were really at the end of the line. Blockwise ranges behave as a "rectangle" on screen rather than a contiguous range of characters.

5.2 Operators

An operator is a command that acts on the text moved over by a motion, such as `c` (change), `d` (delete) or `y` (yank or copy, not to be confused with "yank" in Emacs terminology which means *paste*).

`(evil-define-operator OPERATOR (BEG END ARGS...) DOC [[KEY VALUE]...] BODY...)`

Define an operator command *OPERATOR*. The operator acts on the range of characters *BEG* through *END*. *BODY* must execute the operator by potentially manipulating the buffer contents, or otherwise causing side effects to happen.

Optional keyword arguments are:

- `:type` - force the input range to be of a given type (`inclusive`, `line`, `block`, and `exclusive`, or a self-defined motion type).
- `:motion` - use a predetermined motion instead of waiting for one from the keyboard. This does not affect the behavior in visual state, where selection boundaries are always used.
- `:repeat` - if non-nil (default), then `.` will repeat the operator.
- `:move-point` - if non-nil (default), the cursor will be moved to the beginning of the range before the body executes
- `:keep-visual` - if non-nil, the selection is not disabled when the operator is executed in visual state. By default, visual state is exited automatically.

For example, this is an operator that performs ROT13 encryption on the text under consideration:

```
(evil-define-operator evil-rot13 (beg end)
  "ROT13 encrypt text."
  (rot13-region beg end))
```

Binding this to `g?` (where it is by default) will cause a key sequence such as `g?w` to encrypt from the current cursor to the end of the word.

5.3 Text objects

Text objects are like motions in that they define a range over which an operator may act. Unlike motions, text objects can set both a beginning and an endpoint. In visual state, text objects alter both ends of the selection.

Text objects are not directly usable in normal state. Instead, they are bound in the two keymaps `evil-inner-text-ojects-map` and `evil-outer-text-objects-map`, which are available in visual and operator-pending state under the keys `i` and `a` respectively.

`(evil-define-text-object OBJECT (COUNT) DOC [[KEY VALUE]...] BODY...)`

Define a text object command *OBJECT*. *BODY* should return a range `(BEG END)` to the right of point if *COUNT* is positive, and to the left of it if negative.

Optional keyword arguments:

- `:type` - determines how the range applies after an operator (`inclusive`, `line`, `block`, and `exclusive`, or a self-defined motion type).

- `:extend-selection` - if non-nil (default), the text object always enlarges the current selection. Otherwise, it replaces the current selection.

For eample, this is a text object which selects the next three characters after the current location:

```
(evil-define-text-object foo (count)
  "Select three characters."
  (list (point) (+ 3 (point))))
```

For convenience, Evil provides several functions returning a list of positions which can be used for defining text objects. All of them follow the convention that a positive *count* selects text after the current location, while negative *count* selects text before it.

Note: The *thingatpt* library is used quite extensively in Evil to define text objects, and this dependency leaks through in the following functions. A *thing* in this context is any symbol for which there is a function called `forward-THING`[1] which moves past a number of *things*.

`(evil-select-inner-object THING BEG END TYPE [COUNT LINE])`
Return an inner text object range of *COUNT* objects. If *COUNT* is positive, return objects following point; if *COUNT* is negative, return objects preceding point. If one is unspecified, the other is used with a negative argument. *THING* is a symbol understood by *thing-at-point*. *BEG*, *END* and *TYPE* specify the current selection. If *LINE* is non-nil, the text object should be linewise, otherwise it is character wise.

`(evil-select-an-object THING BEG END TYPE COUNT [LINE])`
Return an outer text object range of *COUNT* objects. If *COUNT* is positive, return objects following point; if *COUNT* is negative, return objects preceding point. If one is unspecified, the other is used with a negative argument. *THING* is a symbol understood by *thing-at-point*. *BEG*, *END* and *TYPE* specify the current selection. If *LINE* is non-nil, the text object should be linewise, otherwise it is character wise.

`(evil-select-paren OPEN CLOSE BEG END TYPE COUNT [INCLUSIVE])`
Return a range `(BEG END)` of *COUNT* delimited text objects. *OPEN* and *CLOSE* specify the opening and closing delimiter, respectively. *BEG END TYPE* are the currently selected (visual) range. If *INCLUSIVE* is non-nil, *OPEN* and *CLOSE* are included in the range; otherwise they are excluded.

The types of *OPEN* and *CLOSE* specify which kind of THING is used for parsing with `evil-select-block`. If *OPEN* and *CLOSE* are characters `evil-up-paren` is used. Otherwise *OPEN* and *CLOSE* must be regular expressions and `evil-up-block` is used.

If the selection is exclusive, whitespace at the end or at the beginning of the selection until the end-of-line or beginning-of-line is ignored.

5.4 Range types

A *type* is a transformation acting on a pair of buffer positions. Evil defines the types `inclusive`, `line`, `block` and `exclusive`, which are used for motion ranges and visual selection. New types may be defined with the macro *evil-define-type*.

`(evil-define-type TYPE DOC [[KEY FUNC]...])`
Define type *TYPE*. *DOC* is a general description and shows up in all docstrings.

Optional keyword arguments:

[1] There are many more ways that a *thing* can be defined, but the definition of `forward-THING` is perhaps the most straightforward way to go about it.

- `:expand` - expansion function. This function should accept two positions in the current buffer, BEG and END,and return a pair of expanded buffer positions.
- `:contract` - the opposite of `:expand`. Optional.
- `:one-to-one` - non-nil if expansion is one-to-one. This means that `:expand` followed by `:contract` always return the original range.
- `:normalize` - normalization function. This function should accept two unexpanded positions and adjust them before expansion. May be used to deal with buffer boundaries.
- `:string` - description function. Takes two buffer positions and returns a human-readable string. For example "2 lines"

If further keywords and functions are specified, they are assumed to be transformations on buffer positions, like `:expand` and `:contract`.

5.5 States

States are defined with the macro `evil-define-state`, which takes care to define the necessary hooks, keymaps and variables, as well as a toggle function `evil-NAME-state` and a predicate function `evil-NAME-state-p` for checking whether the state is active.

`(evil-define-state STATE DOC [[KEY VAL]...] BODY...)`

Define an Evil state *STATE*. *DOC* is a general description and shows up in all docstrings; the first line of the string should be the full name of the state.

BODY is executed each time the state is enabled or disabled.

Optional keyword arguments:

- `:tag` - the mode line indicator, e.g. "<T>".
- `:message` - string shown in the echo area when the state is activated.
- `:cursor` - default cursor specification.
- `:enable` - list of other state keymaps to enable when in this state.
- `:entry-hook` - list of functions to run when entering this state.
- `:exit-hook` - list of functions to run when exiting this state.
- `:suppress-keymap` - if non-nil, effectively disables bindings to `self-insert-command` by making `evil-suppress-map` the parent of the global state keymap.

The global keymap of this state will be `evil-test-state-map`, the local keymap will be `evil-test-state-local-map`, and so on.

For example:

```
(evil-define-state test
  "Test state."
  :tag " <T> "
  (message (if (evil-test-state-p)
               "Enabling test state."
             "Disabling test state.")))
```

CHAPTER 6

Frequently Asked Questions

6.1 Problems with the escape key in the terminal

A common problem when using Evil in terminal mode is a certain delay after pressing the escape key. Even more, when pressing the escape key followed quickly by another key the command is recognized as `M-<key>` instead of two separate keys: `ESC` followed by `<key>`. In fact, it is perfectly valid to simulate `M-<key>` by pressing `ESC <key>` quickly (but see below).

The reason for this is that in terminal mode a key sequence involving the meta key (or alt key) always generates a so called "escape sequence", i.e. a sequence of two events sent to Emacs, the first being `ESC` and the second the key pressed simultaneously. The problem is that pressing the escape key itself also generates the `ESC` event. Thus, if Emacs (and therefore Evil) receives an `ESC` event there is no way to tell whether the escape key has been pressed (and no further event will arrive) or a `M-<key>` combination has been pressed (and the `<key>` event will arrive soon). In order to distinguish both situations Evil does the following. After receiving an `ESC` event Evil waits for a short time period (specified by the variable `evil-esc-delay` which defaults to 0.01 seconds) for another event. If no other event arrives Evil assumes that the plain escape key has been pressed, otherwise it assumes a `M-<key>` combination has been pressed and combines the `ESC` event with the second one. Because a `M-<key>` sequence usually generates both events in very quick succession, 0.01 seconds are usually enough and the delay is hardly noticeable by the user.

If you use a terminal multiplexer like *tmux* or *screen* the situation may be worse. These multiplexers have exactly the same problem recognizing `M-<key>` sequences and often introduce their own delay for the `ESC` key. There is no way for Evil to influence this delay. In order to reduce it you must reconfigure your terminal multiplexer.

Note that this problem should not arise when using Evil in graphical mode. The reason is that in this case the escape key itself generates a different command, namely `escape` (a symbol) and hence Evil can distinguish whether the escape key or a `M-<key>` combination has been pressed. But this also implies that pressing `ESC` followed by `<key>` cannot be used to simulate `M-<key>` in graphical mode!

6.2 Underscore is not a word character

An underscore _ is a word character in Vim. This means that word motions like `w` skip over underlines in a sequence of letters as if it was a letter itself. In contrast, in Evil the underscore is often a non-word character like operators, e.g. +.

The reason is that Evil uses Emacs' definition of a word and this definition does often not include the underscore. In Emacs word characters are determined by the syntax-class of the buffer. The syntax-class usually depends on the major-mode of this buffer. This has the advantage that the definition of a "word" may be adapted to the particular type of document being edited. Evil uses Emacs' definition and does not simply use Vim's definition in order to be consistent with other Emacs functions. For example, word characters are exactly those characters that are matched by the regular expression character class `[:word:]`.

If you would be satisfied by having the * and # searches use symbols instead of words, this can be achieved by setting the `evil-symbol-word-search` variable to `t`.

If you want the underscore to be recognised as word character for other motions, you can modify its entry in the syntax-table:

```
(modify-syntax-entry ?_ "w")
```

This gives the underscore the 'word' syntax class. You can use a mode-hook to modify the syntax-table in all buffers of some mode, e.g.:

```
(add-hook 'c-mode-common-hook
          (lambda () (modify-syntax-entry ?_ "w")))
```

This gives the underscore the word syntax-class in all C-like buffers.

Similarly to Emacs' definition of a word, the definition of a "symbol" is also dependent on the syntax-class of the buffer, which often includes the underscore. The default text objects keymap associates kbd::*o* with the symbol object, making kbd::*cio* a good alternative to Vim's kbd::*ciw*, for example. The following will swap between the word and symbol objects in the keymap:

```
(define-key evil-outer-text-objects-map "w" 'evil-a-symbol)
(define-key evil-inner-text-objects-map "w" 'evil-inner-symbol)
(define-key evil-outer-text-objects-map "o" 'evil-a-word)
(define-key evil-inner-text-objects-map "o" 'evil-inner-word)
```

This will not change the motion keys, however. One way to make word motions operate as symbol motions is to alias the `evil-word` *thing*[1] to the `evil-symbol` thing:

```
(defalias 'forward-evil-word 'forward-evil-symbol)
```

[1] Many of Evil's text objects and motions are defined in terms of the *thingatpt* library, which in this case are defined entirely in terms of `forward-THING` functions. Thus aliasing one to another should make all motions and text objects implemented in terms of that *thing* behave the same.

CHAPTER 7

Internals

7.1 Command properties

Evil defines *command properties* to store information about commands[1], such as whether they should be repeated. A command property is a `:keyword` with an associated value, e.g. `:repeat nil`.

`(evil-add-command-properties COMMAND [PROPERTIES...])`
Add *PROPERTIES* to *COMMAND*. *PROPERTIES* should be a property list. To replace all properties at once, use `evil-set-command-properties`.

`(evil-set-command-properties COMMAND [PROPERTIES...])`
Replace all of *COMMAND*'s properties with *PROPERTIES*. *PROPERTIES* should be a property list. This erases all previous properties; to only add properties, use `evil-set-command-property`.

`(evil-get-command-properties COMMAND)`
Return all Evil properties of *COMMAND*. See also `evil-get-command-property`.

`(evil-get-command-property COMMAND PROPERTY [DEFAULT])`
Return the value of Evil *PROPERTY* of *COMMAND*. If the command does not have the property, return *DEFAULT*. See also `evil-get-command-properties`.

`(evil-define-command COMMAND (ARGS...) DOC [[KEY VALUE]...] BODY...)`
Define a command *COMMAND*.

For setting repeat properties, use the following functions:

`(evil-declare-repeat COMMAND)`
Declare *COMMAND* to be repeatable.

`(evil-declare-not-repeat COMMAND)`
Declare *COMMAND* to be nonrepeatable.

`(evil-declare-change-repeat COMMAND)`
Declare *COMMAND* to be repeatable by buffer changes rather than keystrokes.

[1] In this context, a *command* may mean any Evil motion, text object, operator or indeed other Emacs commands, which have not been defined through the Evil machinery.

CHAPTER 8

The GNU Free Documentation License

Version 1.3, 3 November 2008

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document *free* in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

   You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those

of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

   If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

   If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

   If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

   It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

   You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

   A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

   B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

   C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

   D. Preserve all the copyright notices of the Document.

   E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all

of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

   You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

   You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

   A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

   If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will

automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

    The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft.

    Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

    "Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

    "CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

    "Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

    An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

    The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Emacs lisp functions and variables

T

U

V

W