Evil-mode manual

for version 1.3.0-snapshot

Eivind Fonn, Frank Fischer, Vegard Øye

This manual is for Evil (version 1.3.0-snapshot), an extensible vi layer for Emacs.

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

Table of Contents

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 and MELPA unstable. To set up `package.el` to work with one of these repositories, you can follow the instructions on melpa.org (`https://melpa.org/#/getting-started`).

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

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

then choose 'evil' when queried for a package name. Finally, add the following lines to your Emacs init file:

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

1.2 Manual installation

First, install `undo-tree`, `goto-chg` and `cl-lib`. If you have an Emacs version newer than 24.2, 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 you 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.

Evil uses the term *state* for what is called a "mode" in vi, since *mode* already has its own meaning in Emacs. Evil defines a number of states, such as normal state (`<N>`), insert state (`<I>`), visual state (`<V>`), replace state (`<R>`), operator-pending state (`<O>`), motion state (`<M>`) and Emacs state (`<E>`). Each state has its own keymaps and customization variables.

[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: Vim official website (`https://vim.org`).

2 Settings

Evil's behaviour can be adjusted by setting some variables. The current values may 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 8)
;; Load Evil
(require 'evil)
```

Note that if a variable is buffer-local, you should use `setq-default` instead of `setq` to change its global value.

`evil-want-fine-undo` [Variable]

If true, a change-based action like `cw` may be undone in several steps. Otherwise (default), it is undone in one single operation.

`evil-backspace-join-lines` [Variable]

If true (default), backspace in insert state may join lines together. Otherwise, this is prevented.

`evil-kbd-macro-suppress-motion-error` [Variable]

This variable controls whether the motions `h` and `l` trigger end-of-line or beginning-of-line errors in macros. If equal to `record`, errors are only suppressed when recording macros, but not when replaying them. If equal to `replay`, errors are only suppressed when replaying macros, but not when recording them. If equal to `t`, errors are always suppressed. If equal to `nil` (default), errors are never suppressed.

`evil-mode-line-format` [Variable]

Determines the position of the state identifier tag in the mode line. If equal 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 may be a cons cell where the *car* is either `before` or `after`, and the *cdr* is a symbol already present in the mode line. The state identifier tag is then placed before or after that symbol, respectively.

2.1 Wants

`evil-want-C-i-jump` [Variable]

If true (default), then `C-i` jumps forward in the jump list. Otherwise, `C-i` inserts a tab character.

`evil-want-C-u-scroll` [Variable]

If true, then `C-u` scrolls the buffer, as in vi. Otherwise (default), `C-u` applies a prefix argument. The binding of `C-u` mirrors Emacs behaviour by default due to the relative ubiquity of prefix arguments.

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

2.2 Search settings

`evil-regexp-search` [Variable]

If true (default), then `/` and `?` use regular expressions for searching. Otherwise, the search expression is interpreted as plain text.

`evil-search-wrap` [Variable]

If true (default), then `/` and `?` wrap the search around the buffer. Otherwise, they stop at the buffer boundaries.

`evil-flash-delay` [Variable]

The number of seconds to flash search matches when pressing `n` and `N`.

2.3 Indentation settings

`evil-auto-indent` [Variable]

If true (default), inserted line will be indented when using the `o` and `O` operators. Buffer-local.

`evil-shift-width` [Variable]

The number of columns by which a line is shifted when using the `<` and `>` commands. Buffer-local.

`evil-shift-round` [Variable]

If true (default), lines shifted by `<` and `>` are rounded to the nearest multiple of `evil-shift-width`. Otherwise they are always *shifted* by a multiple of `evil-shift-width`. Buffer-local.

`evil-indent-convert-tabs` [Variable]

If true (default), the `=` indendation operator converts between leading tabs and spaces, according to the value of `indent-tabs-mode`. Otherwise, tabs and spaces are left alone.

2.4 Cursor movement settings

`evil-repeat-move-cursor` [Variable]

If true (default), repeating commands with `.` may move the cursor. Otherwise, the original position is preserved.

`evil-move-cursor-back` [Variable]

If true (default), 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.

`evil-move-beyond-eol` [Variable]

If true, the cursor is allowed to move one past the last character of a line, as in Emacs. Otherwise (default), it cannot move past the last character.

`evil-cross-lines` [Variable]

If true, certain motions that conventionally operate in a single line may move to other lines. Otherwise (default), they are restricted to the current line. This applies to `h`, `l`, `f`, `F`, `t`, `T`, and `~`.

`evil-respect-visual-line-mode` [Variable]

If true, `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). Otherwise (default), `visual-line-mode` is ignored.

`evil-repeat-find-to-skip-next` [Variable]

If true (default), repeating a `t` or `T` motion using `;` or `,` skips adjacent matches. Otherwise, you may find that repeated motions do not move the cursor.

`evil-track-eol` [Variable]

If true (default), vertical motions after `$` maintain the cursor at the end of the line. This is analogous to `track-eol`, but respects our interpretation of end-of-line.

2.5 Cursor display settings

`evil-default-cursor` [Variable]

Specifies the default cursor to use. This may be either a cursor type as per the requirements outlined by `C-h v cursor-type`, a color string, a function that sets the cursor, or a list of such.