

GNU Emacs como editor de código Python

Edison Ibáñez

2018-10-28

Topic

- 1 **Introducción**
- 2 Instalación
- 3 Uso de Emacs
- 4 Configuración Básica
- 5 Programación
- 6 Python
- 7 Enlaces de Interés

¿Que es Emacs?

Es un editor de texto que dispone de gran cantidad de funciones.

En su manual se lo describe como "un editor extensible, personalizable, auto-documentado y de tiempo real.

El **EMACS** original significa, Editor MACroS para el TECO. Fue escrito en 1975 por **Richard Stallman** junto con **Guy Steele**.

Topic

- 1 Introducción
- 2 Instalación**
- 3 Uso de Emacs
- 4 Configuración Básica
- 5 Programación
- 6 Python
- 7 Enlaces de Interés

GNU Linux

La mayoría de las distribuciones de GNU/Linux proporcionan Emacs en sus repositorios, esta es la forma recomendada de Instalarlo, para disponer de la ultima version.

Archlinux

```
$ sudo pacman -S emacs
```

Ubuntu/Debian y derivados

```
$ sudo apt-get install emacs
```

Fedora/Centos y derivados

```
$ sudo yum install emacs
```

GNU Linux

La mayoría de las distribuciones de GNU/Linux proporcionan Emacs en sus repositorios, esta es la forma recomendada de instalarlo, para disponer de la última versión.

Archlinux

```
$ sudo pacman -S emacs
```

Ubuntu/Debian y derivados

```
$ sudo apt-get install emacs
```

Fedora/Centos y derivados

```
$ sudo yum install emacs
```

GNU Linux

La mayoría de las distribuciones de GNU/Linux proporcionan Emacs en sus repositorios, esta es la forma recomendada de instalarlo, para disponer de la última versión.

Archlinux

```
$ sudo pacman -S emacs
```

Ubuntu/Debian y derivados

```
$ sudo apt-get install emacs
```

Fedora/Centos y derivados

```
$ sudo yum install emacs
```

Otros Sistemas

Para mayor información se puede acceder al siguiente enlace: [Emacs](#)

Topic

- 1 Introducción
- 2 Instalación
- 3 Uso de Emacs**
- 4 Configuración Básica
- 5 Programación
- 6 Python
- 7 Enlaces de Interés

Comandos

Emacs utiliza combinaciones de teclas para realizar varias acciones.

- Teclas:
 - Control (C)
 - Alt (Meta) (M)
 - Windows (Super) (s)

Comando	Tecla	Descripción
search-word	C-s	Buscar una palabra en el buffer.
undo	C-/	Deshacer el último cambio.
keyboard-quit	C-g	Abortar el comando actual.
find-file	C-x C-f	Buscar y abrir un archivo.
save-buffer	C-x C-s	Guardar.
save-with-newname	C-x C-w	Guardar como.
cut	C-w	Cortar todo el texto entre el marcador y el cursor.
copy	M-w	Copiar todo el texto entre el marcador y el cursor.
paste	C-y	Pegar texto del portapapeles de Emacs.

Mayor información **C-h t**

El Minibuffer

Es el espacio en el que Emacs pide información. Se puede introducir el texto que se debe encontrar en una búsqueda, el nombre de un fichero para leer o guardar e información similar.

Gestión de ficheros y visualización

Emacs permite la edición y visualización de ficheros mediante el uso de **buffers**, cada vez que se abre un archivo o fichero se crea un nuevo buffer con el nombre del archivo.

Mayor información **Buffers**

Topic

- 1 Introducción
- 2 Instalación
- 3 Uso de Emacs
- 4 Configuración Básica**
- 5 Programación
- 6 Python
- 7 Enlaces de Interés

Archivo de Configuración

La configuración de Emacs se la hace usando el dialecto **Emacs Lisp** que también es llamado **elisp**.

Pero también se la puede hacer usando **ORGMODE** mediante **Programación literaria**. Lo primero que tenemos que tomar en cuenta es donde tenemos que ubicar el archivo de configuración.

En los sistemas **GNU Linux** el archivo se localiza en **\$HOME/.emacs.d/init.el**. Mas información en **Where do I put my init file?**

Repositorios

Antes de empezara a instalar los paquetes para **Emacs** se debe especificar los repositorios que utilizara **ELPA**.

```
;;; init.el --- Emacs Configuration -*- lexical-binding: t -*-
;;; Commentary:
;; This config start here
```

```
;;; Code:
```

```
(setq network-security-level 'high)

(setq package-archives
  '(("gnu" . "http://elpa.gnu.org/packages/")
    ("melpa" . "http://melpa.org/packages/")
    ("org" . "https://orgmode.org/elpa/"))
  package-archive-priorities
  '(("melpa" . 10)
    ("gnu" . 5)
    ("org" . 0)))
```

use-package, bind-key & diminish

`use-package` es una macro que nos permite aislar y mantener mas ordenada la configuración de un paquete.

```
(setq load-prefer-newer t)

(package-initialize)
(when (not package-archive-contents)
  (package-refresh-contents))

(unless (package-installed-p 'use-package)
  (package-install 'use-package))

(eval-when-compile
  (require 'use-package))
(use-package diminish :ensure t)
(use-package bind-key :ensure t)
```

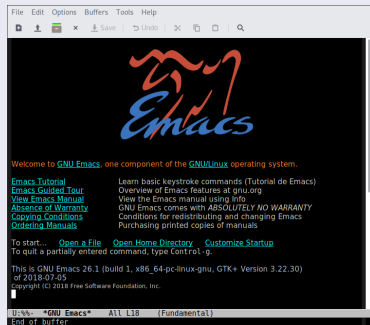

Información Personal

Vamos a indicarle a GNU Emacs nuestros datos de contacto, que serán usados por ejemplo al momento de exportar algún documentos.

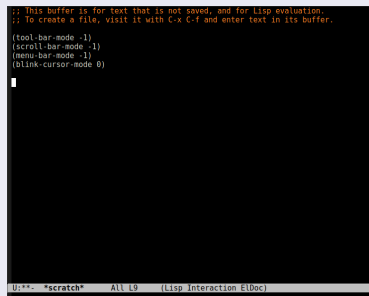
```
(setq user-full-name      "Edison Ibáñez"  
      user-mail-address  "arkhan@disroot.org")
```

Limpiando la Ventana

Interfaz Por Defecto



Interfaz Limpia



Para obtener una interfaz mas limpia debilitamos las barras de la interfaz

```
(tool-bar-mode -1)
(scroll-bar-mode -1)
(menu-bar-mode -1)
(blink-cursor-mode 0)
```

Extras I

Algunos valores por defecto a considerar

```
(setq inhibit-startup-screen t
      initial-scratch-message nil
      visible-bell t
      apropos-do-all t
      large-file-warning-threshold 100000000)
```

```
(fset 'yes-or-no-p 'y-or-n-p)
(toggle-indicate-empty-lines)
(delete-selection-mode)
(blink-cursor-mode -1)
```

```
(defalias 'list-buffers 'ibuffer)
```

```
(set-terminal-coding-system 'utf-8)
(set-keyboard-coding-system 'utf-8)
(set-language-environment 'utf-8)
(set-selection-coding-system 'utf-8)
```

Extras II

```
(setq locale-coding-system 'utf-8)
(prefer-coding-system      'utf-8)
(set-input-method nil)
```

```
(setq auto-save-default nil
      auto-save-list-file-prefix nil
      make-backup-files nil)
```

```
(setq-default indent-tabs-mode nil
               default-tab-width 4
               c-basic-offset 4)
```

```
(setq show-paren-delay 0)
(show-paren-mode t)
```

```
(global-hl-line-mode 1)
```

```
(column-number-mode t)
(setq size-indication-mode t)
```

Extras III

```
(which-function-mode 1)

(setq select-enable-primary t)

(global-set-key "\C-w" 'backward-kill-word)
(global-set-key "\C-x\C-k" 'kill-region)
(global-set-key "\C-c\C-k" 'kill-region)

(add-hook 'text-mode-hook 'turn-on-auto-fill)
(add-hook 'text-mode-hook
  '(lambda() (set-fill-column 80)))

(setq browse-url-browser-function 'browse-url-generic
      browse-url-generic-program "firefox")
```

Ivy I

```
(use-package flx :ensure t)
(use-package smex :ensure t)

(use-package ivy
  :ensure t
  :diminish ivy-mode
  :bind (:map ivy-mode-map
             ("C-' " . ivy-avy))
  :config
  (setq ivy-wrap t
        ivy-virtual-abbreviate 'full
        ivy-use-virtual-buffers t
        ivy-use-selectable-prompt t
        ivy-count-format "(%d/%d) "
        ivy-re-builders-alist
        '((read-file-name-internal . ivy--regex-fuzzy)
          (t . ivy--regex-plus))
        ivy-on-del-error-function nil
```

Ivy II

```
ivy-initial-inputs-alist nil)

(ivy-mode 1))

(setq confirm-nonexistent-file-or-buffer t)

(use-package swiper
  :bind* (("C-s" . swiper)
          ("C-r" . swiper)
          ("C-M-s" . swiper-all))
  :bind (:map read-expression-map
              ("C-r" . counsel-expression-history)))

(use-package counsel
  :bind (("M-x" . counsel-M-x)
        ("C-c b" . counsel-imenu)
        ("C-x C-f" . counsel-find-file)
        ("C-x C-r" . counsel-rg)
        ("C-h f" . counsel-describe-function))
```

Ivy III

```
("C-h v" . counsel-describe-variable)
("C-h b" . counsel-descbinds)
("M-y" . counsel-yank-pop)
("M-SPC" . counsel-shell-history))

:config
(setq counsel-find-file-at-point t
  counsel-rg-base-command "rg -uuu -S --no-heading --line-number --co
```


Tema

```
(use-package ujelly-theme  
  :ensure t  
  :config (load-theme 'ujelly t))
```

Tipo de Letra

```
(set-face-attribute 'default nil :family "DejaVu Sans Mono" :height 90)
(set-fontset-font "fontset-default" nil
  (font-spec :size 20 :name "Symbola"))
```

Barra de Estado

```
(setq line-number-mode t
      column-number-mode t)

(use-package smart-mode-line
  :ensure t
  :config (setq sml/no-confirm-load-theme t
                sml/theme 'dark
                sml/vc-mode-show-backend t
                sml/mode-width 'full
                sml/shorten-modes t)
  (sml/setup))
```

Topic

- 1 Introducción
- 2 Instalación
- 3 Uso de Emacs
- 4 Configuración Básica
- 5 Programación**
- 6 Python
- 7 Enlaces de Interés

Comment-dwim-2

```
(use-package comment-dwim-2
  :ensure t
  :bind* ("M-;" . comment-dwim-2))
```

Company I

```
(use-package company
  :ensure t
  :init
  (setq company-backends '((company-files
                             company-keywords
                             company-capf
                             company-yasnippet)
                           (company-abbrev company-dabbrev)))
  (setq company-auto-complete nil
        company-echo-delay 0
        company-idle-delay 0.2
        company-minimum-prefix-length 1
        company-tooltip-align-annotations t
        company-tooltip-limit 20
        company-transformers '(company-sort-by-occurrence))
  (global-company-mode))

(defun company-mode/backend-with-yas (backend)
```

Company II

```
(if (or (and (listp backend) (member 'company-yasnippet backend)))
    backend
    (append (if (consp backend) backend (list backend))
            '(:with company-yasnippet))))
(add-hook 'company-mode-hook (lambda () (setq company-backends (mapcar #'co

(defun add-pcomplete-to-capf ()
  (add-hook 'completion-at-point-functions 'pcomplete-completions-at-point

(add-hook 'org-mode-hook #'add-pcomplete-to-capf)

(use-package company-quickhelp
  :ensure t
  :after company
  :config (company-quickhelp-mode 1))
```

Fill Column Indicator

```
(use-package fill-column-indicator
  :ensure t
  :commands (fci-mode)
  :init (setq fci-rule-width 5
              fci-rule-column 79))
```


Flycheck

```
(use-package flycheck
  :ensure t
  :bind (("C-c e n" . flycheck-next-error)
        ("C-c e p" . flycheck-previous-error))
  :config
  (add-hook 'after-init-hook #'global-flycheck-mode)

  (setq-default flycheck-disabled-checkers
    (append flycheck-disabled-checkers
      '(javascript-jshint)))

  (setq-default flycheck-disabled-checkers
    (append flycheck-disabled-checkers
      '(json-jsonlist))))
```

Git I

```
(setq vc-follows-symlinks t
      find-file-visit-truename t
      vc-handled-backends nil)

(use-package magit
  :ensure t
  :bind (("C-x g c" . magit-commit)
         ("C-x g e" . magit-ediff-resolve)
         ("C-x g g" . magit-grep)
         ("C-x g l" . magit-file-log)
         ("C-x g p" . magit-push)
         ("C-x g r" . magit-rebase-interactive)
         ("C-x g s" . magit-status)
         ("C-x g u" . magit-pull)
         ("C-x g x" . magit-checkout))
  :init
  (progn
    (setq magit-git-executable "tg"))
```

Git II

```

(delete 'Git vc-handled-backends)
(defadvice magit-status (around magit-fullscreen activate)
  (window-configuration-to-register :magit-fullscreen)
  ad-do-it
  (delete-other-windows))
(defadvice git-commit-commit (after delete-window activate)
  (delete-window))
(defadvice git-commit-abort (after delete-window activate)
  (delete-window))
(defun magit-commit-mode-init ()
  (when (looking-at "\n")
    (open-line 1))))
:config
(progn
  (defadvice magit-quit-window (around magit-restore-screen activate)
    (let ((current-mode major-mode))
      ad-do-it
      (when (eq 'magit-status-mode current-mode)
        (jump-to-register :magit-fullscreen)))))

```

Git III

```
(defun magit-maybe-commit (&optional show-options)
  "Runs magit-commit unless prefix is passed"
  (interactive "P")
  (if show-options
      (magit-key-mode-popup-committing)
      (magit-commit)))
(define-key magit-mode-map "c" 'magit-maybe-commit)

(setq magit-completing-read-function 'ivy-completing-read
      magit-default-tracking-name-function 'magit-default-tracking-name
      magit-status-buffer-switch-function 'switch-to-buffer
      magit-diff-refine-hunk t
      magit-rewrite-inclusive 'ask
      magit-process-find-password-functions '(magit-process-password-au
      magit-save-some-buffers t
      magit-process-popup-time 10
      magit-set-upstream-on-push 'askifnotset
      magit-refs-show-commit-count 'all
      magit-log-buffer-file-locket t)))
```

Git IV

```
(use-package git-gutter
  :ensure t
  :defer 1
  :bind (("C-x C-g" . git-gutter)
        ("C-x v =" . git-gutter:popup-hunk)
        ("C-x p" . git-gutter:previous-hunk)
        ("C-x n" . git-gutter:next-hunk)
        ("C-x v s" . git-gutter:stage-hunk)
        ("C-x v r" . git-gutter:revert-hunk)
        ("C-x v SPC" . git-gutter:mark-hunk))
  :config
  (if (display-graphic-p)
      (use-package git-gutter-fringe
        :ensure t))
  (global-git-gutter-mode t)
  (setq-default fringes-outside-margins t)
  (setq indicate-empty-lines nil)
  (setq git-gutter:lighter ""
```

Git V

```
git-gutter:handled-backends '(git hg bzh svn))
(set-face-foreground 'git-gutter:modified "purple")
(set-face-foreground 'git-gutter:added "green")
(set-face-foreground 'git-gutter:deleted "red"))

(use-package gitconfig-mode
  :ensure t
  :mode ("/*\\.?git/?config$"
        "/*\\.gitmodules$")
  :init (add-hook 'gitconfig-mode-hook 'flyspell-mode))

(use-package gitignore-mode
  :ensure t
  :mode ("/*\\.gitignore$"
        "/*\\.git/info/exclude$"
        "/git/ignore$"))

(use-package gitattributes-mode
  :ensure t
```

Git VI

```
:defer t)

(use-package git-timemachine
  :ensure t
  :commands git-timemachine
  :bind (:map git-timemachine-mode
    ("c" . git-timemachine-show-current-revision)
    ("b" . git-timemachine-switch-branch)))

(use-package smerge-mode
  :ensure t
  :config
  (defun enable-smerge-maybe ()
    (when (and buffer-file-name (vc-backend buffer-file-name))
      (save-excursion
        (goto-char (point-min))
        (when (re-search-forward "^<<<<<<< " nil t)
          (smerge-mode +1))))))
```

Git VII

```
(add-hook 'buffer-list-update-hook #'enable-smerge-maybe))
```


ledit

```
(use-package iedit  
  :ensure t)
```

move-dup

```
(use-package move-dup
  :ensure t
  :diminish move-dup-mode
  :bind (("S-M-<up>" . md/move-lines-up)
         ("S-M-<down>" . md/move-lines-down)
         ("C-M-<up>" . 'md/duplicate-up)
         ("C-M-<down>" . 'md/duplicate-down))
  :init (global-move-dup-mode))
```

Parents I

```
(electric-pair-mode 1)

(use-package paren
  :init (show-paren-mode)
  :config
  (set-face-background 'show-paren-match (face-background 'default))
  (set-face-foreground 'show-paren-match "#def")
  (set-face-attribute 'show-paren-match nil :weight 'extra-bold))

(use-package smartparens
  :ensure t
  :commands
  (smartparens-mode
   smartparens-strict-mode)
  :bind
  (:map smartparens-strict-mode-map
    ("C-}") . sp-forward-slurp-sexp)
    ("M-s") . sp-backward-unwrap-sexp)
```

Parents II

```
      ("C-c [" . sp-select-next-thing)
      ("C-c "]" . sp-select-next-thing-exchange))
:config
(require 'smartparens-config))

(use-package rainbow-delimiters
  :ensure t
  :config
  (add-hook 'prog-mode-hook 'rainbow-delimiters-mode))
```

PO I

```
(use-package po-mode
  :ensure t
  :config
  ;; Fuente: https://www.emacswiki.org/emacs/PoMode
  (defun po-wrap ()
    "Filter current po-mode buffer through `msgcat' tool to wrap all lines.
    (interactive)
    (if (eq major-mode 'po-mode)
        (let ((tmp-file (make-temp-file "po-wrap."))
              (tmp-buf (generate-new-buffer "*temp*")))
          (unwind-protect
              (progn
                (write-region (point-min) (point-max) tmp-file nil 1)
                (if (zerop
                    (call-process
                     "msgcat" nil tmp-buf t (shell-quote-argument tmp-file)
                     (let ((saved (point))
                         (inhibit-read-only t))
```

PO II

```

        (delete-region (point-min) (point-max))
        (insert-buffer tmp-buf)
        (goto-char (min saved (point-max))))
    (with-current-buffer tmp-buf
      (error (buffer-string))))
  (kill-buffer tmp-buf)
  (delete-file tmp-file))))

(defun po-guess-language ()
  "Return the language related to this PO file."
  (save-excursion
    (goto-char (point-min))
    (re-search-forward po-any-msgstr-block-regexp)
    (goto-char (match-beginning 0))
    (if (re-search-forward
        "\n\"Language: +\\(\\.+\\)\\\\\\n\"$"
        (match-end 0) t)
        (po-match-string 1))))

```

PO III

```
(defadvice po-edit-string (around setup-spell-checking (string type expansion)
  "Set up spell checking in subedit buffer."
  (let ((po-language (po-guess-language)))
    ad-do-it
    (if po-language
      (progn
        (ispell-change-dictionary po-language)
        (turn-on-flyspell)
        (flyspell-buffer)))))))
```

Projectile I

```
(use-package projectile
  :ensure t
  :diminish projectile-mode
  :config
  (setq projectile-file-exists-remote-cache-expire (* 10 60)
        projectile-indexing-method 'alien
        projectile-enable-caching t
        projectile-completion-system 'ivy)
  (projectile-mode))

(use-package counsel-projectile
  :ensure t
  :bind ("C-x r R" . counsel-projectile-rg)
  :config
  (setq counsel-projectile-rg-options-history (list "-uuu"))
  (add-hook 'text-mode-hook 'counsel-projectile-mode)
  (add-hook 'prog-mode-hook 'counsel-projectile-mode))
```


Projectile II

```
(use-package term-projectile  
  :ensure t)
```

```
(use-package rg  
  :ensure t  
  :config (setq rg-command-line-flags (list "-uuu")))
```

Rainbow

```
(use-package rainbow-mode
  :ensure t
  :diminish rainbow-mode
  :config
  (add-hook 'prog-mode-hook 'rainbow-mode)
  (add-hook 'conf-mode-hook 'rainbow-mode))
```

Shell

```
(use-package terminal-here
  :ensure t
  :bind (("C-<f5>" . terminal-here-launch)
         ("C-<f6>" . terminal-here-project-launch)))
```

Undo Tree

```
(use-package undo-tree
  :ensure t
  :diminish undo-tree-mode
  :init
  (progn
    (global-undo-tree-mode)
    (setq undo-tree-auto-save-history t
          undo-tree-visualizer-timestamps t
          undo-tree-visualizer-diff t))
  (add-hook 'write-file-functions #'undo-tree-save-history-hook)
  (add-hook 'find-file-hook #'undo-tree-load-history-hook))
```

Yasnippet

```
(use-package yasnippet
  :ensure t
  :defer 2
  :config (yas-global-mode))

(use-package yasnippet-snippets
  :ensure t)
```

Topic

- 1 Introducción
- 2 Instalación
- 3 Uso de Emacs
- 4 Configuración Básica
- 5 Programación
- 6 Python**
- 7 Enlaces de Interés

Elpy I

```
(use-package elpy
  :ensure t
  :diminish elpy-mode
  :config
  (elpy-enable)
  (setq elpy-rpc-backend "jedi"
        elpy-shell-echo-input nil
        elpy-modules (dolist (elem
                              '(elpy-module-sane-defaults
                                elpy-module-company
                                elpy-module-eldoc
                                elpy-module-highlight-indentation
                                elpy-module-pyvenv
                                elpy-module-yasnippet))))
  (add-to-list 'company-backends 'elpy-company-backend)
  (require 'smartparens-python)
  (with-eval-after-load 'python
```

Elpy II

```
(defun python-shell-completion-native-try ()  
  "Return non-nil if can trigger native completion."  
  (let ((python-shell-completion-native-enable t)  
        (python-shell-completion-native-output-timeout  
          python-shell-completion-native-try-output-timeout))  
    (python-shell-completion-native-get-completions  
      (get-buffer-process (current-buffer))  
      nil "_"))))  
  
(when (require 'flycheck nil t)  
  (setq elpy-modules (delq 'elpy-module-flymake elpy-modules))  
  (add-hook 'elpy-mode-hook 'flycheck-mode)))
```


Debug

```
(defun add-breakpoint ()  
  "Add a break point"  
  (interactive)  
  (newline-and-indent)  
  (insert "import pdb; pdb.set_trace()"))  
  
(define-key elpy-mode-map (kbd "C-c C-b") 'add-breakpoint)
```

Isort

```
(use-package py-isort
  :ensure t
  :config
  (setq py-isort-options '--lines=100))
(add-hook 'before-save-hook 'py-isort-before-save))
```

Virtualenv I

```
(use-package pyvenv
  :config (defalias 'workon 'pyvenv-workon))

(use-package auto-virtualenv
  :ensure t
  :config
  (add-hook 'elpy-mode-hook 'auto-virtualenv-set-virtualenv))

(use-package virtualenvwrapper
  :ensure t
  :commands (venv-workon venv-deactivate
               venv-initialize-interactive-shells venv-initialize-eshell)
  :init
  (venv-initialize-interactive-shells)
  (venv-initialize-eshell)
  (setq venv-location "~/virtualenvs")
  :config
  (add-hook 'venv-postmkvirtualenv-hook
```

Virtualenv II

```
(lambda () (shell-command "pip install jedi rope  
isort importmagic autopep8 yapf  
flake8 virtualenvwrapper"))))
```

Varios

```
(use-package pyimport :ensure t)
(use-package pippel :ensure t)
(use-package pip-requirements :ensure t)
```

Topic

- 1 Introducción
- 2 Instalación
- 3 Uso de Emacs
- 4 Configuración Básica
- 5 Programación
- 6 Python
- 7 Enlaces de Interés**

• Enlaces

- [arkhan/dots](#)
- [drymer/emacs.d](#)
- [joedicastro/emacs.d](#)
- <http://emacsrocks.com/>