

Configuración *Emacs* de Álvaro González

alvaro

December 4, 2018

Contents

1	Cómo funciona este fichero	3
1.1	init.el	3
1.2	Algunos problemas	3
2	Utilidades externas al <i>PATH</i>	4
3	Carga de paquetes	4
3.1	use-package	5
3.2	Paquete image+	7
3.3	Paquete dumb-jump	7
3.4	Paquete bm	8
3.5	Paquete multiple-cursors	8
3.6	Paquete yasnippet	8
3.7	Paquete smartparents	8
3.8	Paquete company	9
3.9	Paquete neotree	10
3.10	Paquete org	10
3.11	Paquete quickrun	11
3.12	Paquete doc-view	11
3.13	Correo electrónico	11
3.14	tramp	11
3.15	<i>Backup</i> de ficheros	12
3.16	Latex	12
3.17	Paquete which-key	13
3.18	Paquete auto-highlight-symbol	13
3.19	Paquete helm	13
3.20	projectile	14
4	Edición	15
4.1	Tabuladores <i>vs</i> espacios	15
4.2	Comportamiento de la selección	15
4.3	Línea nueva al final de fichero	16
4.4	Historia del portapapeles	16
4.5	Recarga de ficheros modificados	16
4.6	Comandos que se consideran <i>avanzados</i>	16

5	Navegación	16
5.1	Ratón en <code>xterm</code>	17
6	Visualización	17
6.1	<code>scroll bars</code>	17
6.2	Cambiar el tamaño de fuente de todo <i>emacs</i> (no solo el buffer actual)	17
6.3	Marcar la línea actual.	17
6.4	Respuestas de confirmación más cortas	17
6.5	Desactivar la campana (<i>bell</i>)	18
6.6	<i>flycheck</i>	18
6.7	<code>scroll</code>	18
6.8	Barra de menú	18
6.9	Ancho de la página de man	18
6.10	Mostrar paréntesis asociados	18
6.11	Servidor emacs	18
6.12	<i>Modeline</i>	19
6.13	<i>Minimap</i>	19
7	Atajos de teclado	19
8	Utilidades	21
8.1	Convierto el buffer actual a una frame nueva	21
8.2	Inicio de una selección rectangular usando el ratón (lo uso poco, prefiero <code>C-x spc</code>)	21
8.3	Abrir el fichero del buffer actual con un programa externo	21
8.4	Copiar el nombre del fichero actual al portapapeles	21
8.5	Arrancar el servidor http de emacs en el directorio actual	22
8.6	Al visitar un fichero, reabrir el bufer como root, incluso a través de tramp. Hay una versión para emacs25 y otra para emacs26.	22
8.7	<i>Transmission</i>	23
8.8	Generar <i>reveal</i> y PDF	23
8.9	Función para decodificar una URL	24
8.10	Horario	24
8.11	Proxy de educamadrid	24
8.12	Inserta la imagen del portapapeles en orgmode.	24
8.13	Eliminar el resto de buffers y ventanas	25
8.14	Convertir la selección en un bloque de código de orgmode	25
8.15	Diferencias con la última versión de git, usando <code>git-gutter</code>	25
9	Apariencia	25
9.1	Líneas vacías al final, como vim	25
9.2	Nivel de indentación	26
9.3	Indicación de cambios de <i>git</i>	26
9.4	Salto de página	26
9.5	Modo proyección o modo trabajo	26
9.6	<i>Fringes</i>	27
9.7	Temas	27
10	Customize	27

1 Cómo funciona este fichero

La configuración de Emacs se realiza con código elisp. Al contrario que otros editores, que están pensados para ser usados sin demasiada customización, los usuarios de Emacs solemos cambiarlo de forma bastante *extrema* (¡por eso nos gusta!).

El problema es que, como con cualquier otro programa que se va modificando durante años, acabas olvidando el por qué de ciertas líneas de código, o dónde se realiza alguna configuración. Para evitar remediarlo, se puede utilizar org-mode para crear al mismo tiempo la documentación y el código de la configuración (como en literate programming). Descubrí en el blog de [Sacha Chua](#) que podía mantener la configuración en un fichero orgmode y así documentar fácilmente cada opción.

Cada bloque de código de tipo `emacs-lisp` se ejecuta al inicio. Algunos bloques están deshabilitados, marcándolos con el tipo `lisp`. Así mantienen su apariencia, pero no son interpretados.

1.1 `init.el`

Emacs comienza cargando el fichero `~/.emacs.d/init.el`. Este fichero simplemente inicializa el sistema de paquetes y carga el paquete `org`, que es el que permite a Emacs manejar este tipo de ficheros. Después, carga este fichero interpretando los bloques de código.

```
;;; Code:
(defun carga-config-org (refresh)
  "Carga la configuración, refrescando la lista de paquetes si se indica REFRESH."
  (setq debug-on-error t)

  (package-initialize nil)
  (setq package-check-signature nil)
  (setq package-archives
    '(
      ("org" . "http://orgmode.org/elpa/")
      ("gnu" . "http://elpa.gnu.org/packages/")
      ("melpa" . "http://melpa.org/packages/") ) )

  (package-initialize t)

  (if refresh
      (progn
        (package-refresh-contents)
        (package-install 'org)
        (package-install 'use-package)))

  (require 'org)
  (require 'ob-tangle)
  (require 'use-package)
  (org-babel-load-file (expand-file-name "~/emacs.d/config.org"))

  (byte-compile-file (expand-file-name "~/emacs.d/config.el"))

  ;; DESACTIVAR EL DEBUG
  (setq debug-on-error nil))

(carga-config-org nil)
```

1.2 Algunos problemas

- A veces, se empieza usando una versión de `org` (de gnu), y después se utiliza la del repositorio de `orgmode`, lo que puede dar problemas. En ese caso, se necesitan dos reinicios.

-
- Los ficheros `org` dan problemas sin tienen un `title:` definido y la versión de `org` instalada no se recompiló. Para solucionarlo:

```
rm $(find ~/.emacs.d/elpa | grep .elc$)
```

Después es necesario volver a compilar los ficheros

```
(byte-recompile-directory (expand-file-name "~/.emacs.d/elpa") 0 t)
```

- En instalaciones nuevas, la carga final de `custom.el` falla si no existe.

```
touch ~/.emacs.d/custom.el
```

2 Utilidades externas al *PATH*

En el directorio `~/.emacs.d/bin/` guardo algunas utilidades que me interesa tener en todos los entornos de trabajo. Lo siguiente es para que estén disponibles desde las *shells* que arranque desde emacs

```
(setq exec-path-separator  
  (if (string-equal system-type "windows-nt") ";" ":"))  
(setenv "PATH"  
  (concat (expand-file-name "~/.emacs.d/bin") exec-path-separator (getenv "PATH")))  
(setq exec-path (append exec-path ' (~/.emacs.d/bin)))
```

Las utilidades pueden consultarse en el [repositorio en github](#), y son:

```
addtopath.sh  
ec  
ghpages.sh  
gifttolatex  
githubclone.sh  
keymon.sh  
plantuml.1.2018.11.jar  
rmtemplatex.sh  
shrinkpdf.sh  
svg2pdf.sh  
vncserver.sh  
wake-alvarogonzalez.sh
```

3 Carga de paquetes

Utilizo tres repositorios de paquetes:

- **melpa**: el más habitual
- **gnu**: me hizo falta para algo que no recuerdo, lo tengo actualmente deshabilitado.
- **org**: las versiones nuevas de `org-mode` se publican antes en este repositorio

3.1 use-package

use-package es una utilidad para la carga y configuración de paquetes con las siguientes ventajas:

- Puede definir dependencias entre paquetes (requires y after)
- Permite la carga diferida, lo que acelera el arranque y el uso de memoria
- Agrupa la configuración de cada paquete
- Instala el paquete si no está instalado

Para actualizar siempre los paquetes, utilizo auto-package-update

```
(use-package auto-package-update
:ensure t
:defer nil
:config
(setq auto-package-update-delete-old-versions t)
(setq auto-package-update-hide-results t)
(auto-package-update-maybe))
```

1. Tabla de paquetes Esta tabla contiene los paquetes que utilizo sin configuración adicional

Table 1: Tabla mis-paquetes

2048-game	Indentación visual de las líneas respecto a su prefijo
adaptive-wrap	
ag	Resalte automático del símbolo bajo el cursor
alert	
auto-highlight-symbol	
bind-key	
calfw	
calfw-ical	
chess	
color-theme-sanityinc-tomorrow	
company	
company-auctex	
company-c-headers	
company-emoji	
company-flx	
company-lsp	
company-quickhelp	
company-restclient	
company-shell	
company-web	
crappy-jsp-mode	
default-text-scale	
diffview	
dired-narrow	
ensime	
esup	
expand-region	
flycheck	

```
;; ADDITIONAL DUMBJUMP RULES
(use-package dumb-jump
  :ensure t
  :config

  (add-to-list 'dumb-jump-find-rules
    '(:type "something" :supports ("ag" "grep" "rg" "git-grep") :language "sql"
      :regex ": \\bJJJ\\j"))
  (add-to-list 'dumb-jump-find-rules
    '(:type "something" :supports ("ag" "grep" "rg" "git-grep") :language "org"
      :regex ": \\bJJJ\\j")))
```

3.4 Paquete **bm**

Siempre me gustaron los *bookmarks* dentro de un fichero de Microsoft Visual C++

```
(use-package bm
  :ensure t
  :bind ((("<C-f2>" . bm-toggle)
        ("<f2>" . bm-next )
        ("<S-f2>" . bm-previous)))
```

3.5 Paquete **multiple-cursors**

```
(use-package multiple-cursors
  :ensure t
  :bind ((("<C-f2>" . bm-toggle)
        ("<f2>" . bm-next )
        ("<S-f2>" . bm-previous))
  :custom (mc/always-run-for-all t))
```

3.6 Paquete **yasnippet**

Plantillas para introducción rápida de partes del texto. *yasnippet* interfiere con otros modos en su uso del tabulador, así que cambio su combinación.

```
(use-package yasnippet
  :ensure t
  :config
  (yas-global-mode 1)
  (define-key yas-minor-mode-map (kbd "<tab>") nil)
  (define-key yas-minor-mode-map (kbd "TAB") nil)
  (define-key yas-minor-mode-map (kbd "C-c TAB") 'yas-expand))

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

3.7 Paquete **smartparens**

Este modo cierra automáticamente los paréntesis y otros bloques

```
(use-package smartparens
  :ensure t
  :config
  (smartparens-global-mode 1))
```

3.8 Paquete **company**

Utilizo **company** como mecanismo de autocomplección. Distingo entre modos de programación y `org-mode`.

```
(require 'company)
(company-flx-mode +1)

(defvar my-company-backends-prog-mode
  '(
    (
      company-web-html
      company-files
      company-dabbrev-code
      company-capf
      company-keywords
      company-lsp
      company-yasnipet
      company-emoji
      company-capf
    )
  )
)

(defvar my-company-backends-org-mode
  '(
    (
      company-files
      company-dabbrev-code
      company-dabbrev
      company-keywords
      company-yasnipet
      company-emoji
      company-capf
    )
  )
)

(defvar my-company-backends my-company-backends-org-mode)

;; set default 'company-backends'
(setq company-backends my-company-backends)
(company-auctex-init)

(add-hook 'after-init-hook 'global-company-mode)

(company-quickhelp-mode 1)

(defun my-company-backends-org-mode-function ()
  (interactive)
  (set (make-local-variable 'company-backends) my-company-backends-org-mode))

(add-hook 'org-mode-hook #'my-company-backends-org-mode-function)

(defun my-company-backends-prog-mode-function ()
  (interactive)
  (set (make-local-variable 'company-backends) my-company-backends-prog-mode))

(add-hook 'prog-mode-hook #'my-company-backends-prog-mode-function)

(define-key company-active-map [escape] 'company-abort)
(global-company-mode)
```

Prefiero que `dabbrev` funcione en comentarios y cadenas. Y que tenga en cuenta el *case*

```
(setq company-dabbrev-code-everywhere t)
(setq company-dabbrev-code-ignore-case nil)
(setq company-dabbrev-everywhere t)
```

```
(setq company-dabbrev-ignore-case 'keep-prefix)
(setq company-dabbrev-downcase nil)
```

3.9 Paquete neotree

En neotree, quiero ver todos los ficheros, y no me importa el ancho fijo de la ventana.

```
(use-package neotree
:ensure t
:defer 1
:config

; https://github.com/jaypei/emacs-neotree/issues/149
(defun neotree-project-root-dir-or-current-dir ()
  "Open NeoTree using the project root, using projectile, or the
current buffer directory."
  (interactive)
  (let ((project-dir (ignore-errors (projectile-project-root)))
        (file-name (buffer-file-name))
        (neo-smart-open t))
    (if (neo-global--window-exists-p)
        (neotree-hide)
        (progn
         (neotree-show)
         (if project-dir
             ; (neotree-dir project-dir)
             (neotree-projectile-action))
         (if file-name
             (neotree-find file-name))))))

(setq neo-show-hidden-files t)
(setq neo-window-fixed-size nil)
(setq neo-hidden-regexp-list (quote ("\\.pyc$" "~$" "^#.*$" "\\elc$"))))
```

3.10 Paquete org

Con estos cambios, se tienen en cuenta los formatos de orgmode en electric-pair-mode

```
(use-package org
:ensure t
:config
  (modify-syntax-entry ?~ "(~" org-mode-syntax-table)
  (modify-syntax-entry ?= "(=" org-mode-syntax-table)
  (modify-syntax-entry ?* "(*" org-mode-syntax-table)
  (modify-syntax-entry ?/ "(/" org-mode-syntax-table))
```

1. Lenguajes org-babel Habilito varios lenguajes que pueden ejecutarse directamente desde los bloques de orgmode.

```
(setq org-plantuml-jar-path (expand-file-name "~/emacs.d/bin/plantuml.1.2018.11.jar"))
(setq plantuml-jar-path org-plantuml-jar-path)

(setq org-babel-load-languages '((scala . t) (shell . t) (emacs-lisp . t) (dot . t) (plantuml . t) (C .
→ t)))

(org-babel-do-load-languages
 'org-babel-load-languages
 '(
  (C . t)
  (dot . t)
  (plantuml . t)
  (scala . t)
  (shell . t)))
```

Además, no pido confirmación para varios lenguajes

```
(defun my-org-confirm-babel-evaluate (lang body)
  (not (member lang '("dot" "emacs-lisp" "shell"))))
(setq org-confirm-babel-evaluate 'my-org-confirm-babel-evaluate)
```

2. Listas alfabéticas

```
(setq org-list-allow-alphabetical t)
```

3. Listados *Latex* Utilizo el paquete `listings` de *Latex* en vez de bloques *verbatim*.

```
(setq org-latex-listings t)
```

4. Selección con mayúsculas

```
(setq org-support-shift-select t)
```

3.11 Paquete **quickrun**

Quickrun ejecuta el buffer actual. Aumento el tiempo límite de la ejecución antes de matar el proceso.

```
(use-package quickrun
  :ensure t
  :config
  (setq quickrun-timeout-seconds 100))
```

3.12 Paquete **doc-view**

Para visualizar documentos desde Emacs, aumento su resolución y anchura.

```
(require 'doc-view)
(setq doc-view-continuous t)
(setq doc-view-image-width 1600)
(setq doc-view-resolution 400)
```

3.13 Correo electrónico

Para enviar email utilizo `sendmail` (lo suelo tener configurado con un *smarthost*)

```
(setq send-mail-function (quote sendmail-send-it))
```

3.14 **tramp**

`tramp` intenta optimizar las conexiones, enviando en línea los ficheros pequeños. Esto me da problemas en algunos sistemas, así que indico que los ficheros se copien a partir de 1 byte de tamaño:

```
(setq tramp-copy-size-limit 1)
(setq tramp-debug-buffer t)
(setq tramp-verbose 10)
```

En ocasiones, `tramp` no consigue conectar con un usuario que tiene `zsh` como shell. Para ello, hay que añadir lo siguiente al fichero `.zshrc` remoto:

```

EN .zshrc PARA QUE FUNCIONE tramp
if [[ "$TERM" == "dumb" ]]
then
  unsetopt zle
  unsetopt prompt_cr
  unsetopt prompt_subst
  unfunction precmd
  unfunction preexec
  PS1='$ '
fi

```

3.15 Backup de ficheros

Emacs guarda una copia de seguridad de los ficheros editados. Si no se configura, crea la copia en el mismo directorio.

Las copias de seguridad son interesantes aunque se utilice un control de versiones. Por ejemplo, se guardan versiones de ficheros del sistema y de los editados con Tramp.

Prefiero guardar todas las copias en un directorio, manteniendo varias versiones de cada fichero.

Tampoco me interesan los ficheros de *lock*.

```

(setq backup-directory-alist `(("." . "~/ .saves")))
(setq backup-by-copying t)
(setq delete-old-versions t
      kept-new-versions 6
      kept-old-versions 2
      version-control t)

(setq create-lockfiles nil)

```

3.16 Latex

```

(setq TeX-auto-save t)
(setq TeX-parse-self t)
(setq TeX-save-query nil)
(setq TeX-PDF-mode t)

```

Para que funcione correctamente el resaltado de sintaxis, hay que informar a Auctex de los entornos *verbatim* utilizados:

```

(setq LaTeX-verbatim-environments
  '("verbatim" "verbatim*" "listadotxt" "PantallazoTexto" "listadosql"))

```

En Ubuntu, Evince puede sincronizarse con Emacs para saber a qué parte de código corresponde una parte del PDF y viceversa

```

(setq TeX-source-correlate-mode t)
(setq TeX-source-correlate-start-server t)

```

Modifico el comando Latex para incluir `-shell-escape`, de forma que Latex pueda arrancar programas de ayuda (por ejemplo, **Inkscape** para convertir SVG a PDF)

```

(setq LaTeX-command-style
  (quote (((" " "%(PDF)%(latex) %(file-line-error) -shell-escape %(extraopts) %S%(PDFout)")))))

```

Se pueden previsualizar los entornos `tikzpicture` y `tabular` directamente en el buffer de Emacs (<https://www.gnu.org/software/auctex/manual/preview-latex.html>)

```
(eval-after-load "preview"
  '(add-to-list 'preview-default-preamble "\\PreviewEnvironment{tikzpicture}" t) )
(eval-after-load "preview"
  '(add-to-list 'preview-default-preamble "\\PreviewEnvironment{tabular}" t) )
(eval-after-load "preview"
  '(add-to-list 'preview-default-preamble "\\PreviewEnvironment{homeworkProblem}" t) )
```

3.17 Paquete **which-key**

Ayuda interactiva de teclado

```
(use-package which-key
  :ensure t
  :config
  (which-key-mode t))
```

3.18 Paquete **auto-highlight-symbol**

Resaltar el símbolo bajo el cursor de forma dinámica. Antes lo resaltaba en todo el buffer, para que se pueda navegar por todas las ocurrencias del fichero, pero ralentizaba bastante. Ahora uso smartscan.

```
(use-package auto-highlight-symbol
  :ensure t
  :config
  (setq ahs-default-range 'ahs-range-display))
```

3.19 Paquete **helm**

helm es un sistema para seleccionar una opción entre varias posibilidades, que se puede usar para casi todo

- Buscar un comando
- Cambiar de buffer
- Navegar por la historia del portapapeles
- Visualizar las ocurrencias de un patrón en un buffer
- ... y más

```
;; HELM
(use-package helm
  :ensure t
  :defer 1
  :config
  (setq helm-split-window-inside-p t)
  (setq helm-display-header-line nil)
  (setq helm-autoresize-max-height 30)
  (setq helm-autoresize-min-height 30)
  (setq projectile-completion-system 'helm)
  (helm-autoresize-mode 1)
  (helm-mode 1)
  (helm-flx-mode +1)
  (setq helm-echo-input-in-header-line t)
  (setq helm-display-buffer-reuse-frame t)
  (setq helm-use-undecorated-frame-option t)
  :after (tramp)) ;; PARA EVITAR EL ERROR Symbols value as variable is void: tramp-methods
```

```
(use-package helm-projectile
  :ensure t
  :defer 1
  :config
  (helm-projectile-on)
  :after (helm))
```

1. *Child frame* helm se muestra en una nueva ventana. Esta ventana puede estar en una nueva *child frame* para no cambiar la disposición de la *frame* original. Esta opción es bastante lenta en algunos sistemas de ventanas.

```
(setq helm-display-function 'helm-display-buffer-in-own-frame
  helm-display-buffer-width 120)
```

swiper es un sistema de búsqueda de patrones en el buffer, con visualización simultánea de todas las ocurrencias, y también usa helm. Ahora estoy valorando si me quedo con swiper o swoop. Lo siguiente es para hacer que también aparezca en una *child frame*.

```
(use-package swiper
  :ensure t
  :defer 1
  :after (helm)
  :config
  (setq swiper-helm-display-function helm-display-function))

(use-package swoop
  :ensure t
  :defer 1
  :after (helm)
  :config
  (setq helm-swoop-split-window-function helm-display-function))
```

3.20 projectile

projectile necesita conocer su tecla de prefijo (utilizo la tradicional).

```
(use-package projectile
  :ensure t
  :config
  (define-key projectile-mode-map (kbd "C-c p") 'projectile-command-map)
  (projectile-mode 1))
```

1. Paquete ox-reveal Cuando exporto un fichero org a reveal.js tengo problemas en la forma en que se escapan los caracteres > y < de los bloques de código. Con esta redefinición de la función org-reveal-src-block queda solucionado

```
;; ESCAPE HTML IN REVEAL
(setq mi-org-html-protect-char-alist
  ' ((" &" . "&amp;")
    (" <" . "&lt;")
    (" >" . "&gt;")
    (" \\%" . "%37;") ))

(defun mi-org-html-encode-plain-text (text)
  "Convert plain text characters from TEXT to HTML equivalent.
Possible conversions are set in 'org-html-protect-char-alist'."
  (dolist (pair org-html-protect-char-alist text)
    (setq text (replace-regexp-in-string (car pair) (cdr pair) text t t))))

(use-package ox-reveal
```

```

:ensure t
:defer 1
:config

(defun org-reveal-src-block (src-block contents info)
  "Transcode a SRC-BLOCK element from Org to Reveal.
  CONTENTS holds the contents of the item. INFO is a plist holding
  contextual information."
  (if (org-export-read-attribute :attr_html src-block :textarea)
      (org-html--textarea-block src-block)
      (let* ((use-highlight (org-reveal--using-highlight.js info))
             (lang (org-element-property :language src-block))
             (caption (org-export-get-caption src-block))
             (not-escaped-code (if (not use-highlight)
                                     (org-html-format-code src-block info)
                                     (cl-letf (((symbol-function 'org-html-htmlize-region-for-paste)
                                                #'buffer-substring))
                                       (org-html-format-code src-block info))))))
        (code (mi-org-html-encode-plain-text not-escaped-code))
        ; (code not-escaped-code)

        (frag (org-export-read-attribute :attr_reveal src-block :frag))
        (code-attrs (or (org-export-read-attribute
                          :attr_reveal src-block :code_attrs) ""))
        (label (let ((lbl (org-element-property :name src-block)))
                  (if (not lbl) ""
                      (format " id=\"%s\"" lbl))))))
      (if (not lang)
          (format "<pre %s>\n%s</pre>"
                  (or (frag-class frag info) " class=\"example\"")
                  label
                  code)
          (format
            "<div class=\"org-src-container\">\n%s\n</div>"
            (if (not caption) ""
                (format "<label class=\"org-src-name\">%s</label>"
                        (org-export-data caption info)))
            (if use-highlight
                (format "\n<pre %s><code class=\"%s\" %s>%s</code></pre>"
                        (or (frag-class frag info) "")
                        label lang code-attrs code)
                (format "\n<pre %s>%s</pre>"
                        (or (frag-class frag info)
                            (format " class=\"src src-%s\"" lang))
                        label code))))))

```

4 Edición

4.1 Tabuladores *vs* espacios

No utilizo tabuladores en las indentaciones.

```

(setq-default indent-tabs-mode nil)
(setq tab-width 2)

```

4.2 Comportamiento de la selección

Al comenzar a escribir con una selección, se borra lo seleccionado.

```

(delete-selection-mode 1)

```

Al copiar la selección, mantener la selección

```

(defadvice kill-ring-save (after keep-transient-mark-active ())
  "Override the deactivation of the mark."

```

```
(setq deactivate-mark nil)
(ad-activate 'kill-ring-save)
```

4.3 Línea nueva al final de fichero

Los ficheros deben tener una línea nueva al final. Además, indicar el fin de fichero como en vim.

```
(setq indicate-empty-lines t require-final-newline t)
```

4.4 Historia del portapapeles

Una de las ventajas de Emacs es su *kill ring*, donde se guarda la historia del portapapeles. Con esta opción, añado a esta historia el portapapeles del sistema. Descubierto en <https://writequit.org/org/settings.html#sec-1-33>

```
(setq save-interprogram-paste-before-kill t)
```

4.5 Recarga de ficheros modificados

Encuentro más conveniente que los ficheros se recarguen si un programa externo los modifica, sin preguntas.

```
(global-auto-revert-mode 1)
(setq global-auto-revert-non-file-buffers t)
(setq auto-revert-verbose nil)
```

4.6 Comandos que se consideran *avanzados*

Emacs tiene algunos comandos considerados confusos deshabilitados. Hay opciones útiles que prefiero que estén activadas por defecto.

```
(put 'narrow-to-region 'disabled nil)
(put 'upcase-region 'disabled nil)
(put 'downcase-region 'disabled nil)
```

5 Navegación

Scroll con teclas de avance de página hasta el extremo del fichero. Sin esta opción, *Emacs* no avanza hasta la primera línea si al dar a RePag no quedan páginas por retroceder.

```
(setq scroll-error-top-bottom t)
```

Utilizo smartscan para localizar ocurrencias de símbolos.

```
(use-package smartscan
  :ensure t
  :defer 1
  :config
  (global-smartscan-mode 1))
```

Algunas ventanas tienen menor *importancia* que otras, ya que tienden a ser temporales (por ejemplo, las ventanas de ayuda). Con popwin, estas ventanas ocupan menos espacio en pantalla y desaparecen con C-g

```
(use-package popwin
:ensure t
:defer 1
:config
  (popwin-mode 1))
```

Agrupo los buffers por proyecto de projectile

```
(use-package ibuffer-projectile
:ensure t
:defer 1
:config
  (add-hook 'ibuffer-hook #'ibuffer-projectile-set-filter-groups)
  (add-hook 'ibuffer-sidebar-mode-hook #'ibuffer-projectile-set-filter-groups)
:after (projectile))
```

Retroceder en la historia de disposición de ventanas y búferes

```
(use-package winner
:ensure t
:defer 1
:config
  (winner-mode 1))
```

Grabar la disposición de buffers y ventanas para la siguiente sesión

```
(setq desktop-save t)
(desktop-save-mode)
```

5.1 Ratón en **xterm**

El ratón también puede utilizarse en un **xterm**

```
(xterm-mouse-mode)
```

6 Visualización

6.1 *scroll bars*

```
(scroll-bar-mode -1)
```

6.2 Cambiar el tamaño de fuente de todo *emacs* (no solo el buffer actual)

```
(default-text-scale-mode 1)
```

6.3 Marcar la línea actual.

Está deshabilitado porque no funciona bien con *overlays*

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

6.4 Respuestas de confirmación más cortas

```
(fset 'yes-or-no-p 'y-or-n-p)
```

6.5 Desactivar la campana (*bell*)

Tanto la señal auditiva como la visual

```
(setq visible-bell 1)
(setq ring-bell-function 'ignore)
```

6.6 *flycheck*

Marca errores en el fichero

```
;; VALIDACIONES
(add-hook 'after-init-hook #'global-flycheck-mode)
```

6.7 *scroll*

El *scroll* de *emacs* es de media en media pantalla, heredado de los terminales modo texto que costaba refrescar. Con los ordenadores actuales, mejor un *scroll* suave

```
(setq scroll-margin 0
      scroll-step 1
      scroll-conservatively 10000
      scroll-preserve-screen-position 1)
```

6.8 Barra de menú

La barra de menú y la de herramientas es de lo primero que se quita al personalizar *emacs*, lo mismo que esa pantalla de inicio.

```
(setq inhibit-startup-message t)
(menu-bar-mode -1)
(tool-bar-mode -1)
```

6.9 Ancho de la página de *man*

```
(setenv "MANWIDTH" "80")
```

6.10 Mostrar paréntesis asociados

```
;; MOSTRAR LOS PARENTESIS ASOCIADOS
(show-paren-mode)
```

6.11 Servidor *emacs*

Arranco el servidor para utilizar *emacsclient*

```
(server-force-delete)
(server-start)
```

6.12 Modeline

Mi línea de estado (modeline)

```
(setq-default mode-line-format
  (list
    " "
    mode-line-modified
    " [%[ " mode-line-buffer-identification " %] "
    " | " ' (vc-mode vc-mode)
    " | %m "
    " | %n "
    " | " mode-line-coding-system-map
    " | " mode-line-misc-info
    " | %IB %Z "
    " | %l:%c "
    mode-line-end-spaces
  ) )
```

6.13 Minimap

El minimap parece una buena idea, pero no funciona demasiado bien

```
(require 'sublimity)
(require 'sublimity-map)
(require 'sublimity-attractive)
(sublimity-map-set-delay 2)
```

7 Atajos de teclado

Cuando quiero cerrar un buffer, prefiero que no pregunte.

```
(defun kill-this-buffer-dont-ask ()
  (interactive)
  (kill-buffer (current-buffer)))
(global-set-key (kbd "C-x k") 'kill-this-buffer-dont-ask)
```

En una búsqueda incremental, utilizo los cursores para ir a otras búsquedas anteriores o para navegar entre las ocurrencias en el fichero

```
;; TECLAS PARA ISEARCH
(progn
  ;; set arrow keys in isearch. left/right is backward/forward, up/down is history. press Return to exit
  (define-key isearch-mode-map (kbd "<up>") 'isearch-ring-retreat )
  (define-key isearch-mode-map (kbd "<down>") 'isearch-ring-advance )

  (define-key isearch-mode-map (kbd "<left>") 'isearch-repeat-backward)
  (define-key isearch-mode-map (kbd "<right>") 'isearch-repeat-forward)

  (define-key minibuffer-local-isearch-map (kbd "<left>") 'isearch-reverse-exit-minibuffer)
  (define-key minibuffer-local-isearch-map (kbd "<right>") 'isearch-forward-exit-minibuffer))
```

A veces es fácil perderse entre comandos a medio introducir y ventanas popup. Me gusta que la tecla escape cancele cualquier acción. Con el siguiente código hago que se cancelen incluso más acciones que con C-g.

```
;; (define-key global-map [escape] 'keyboard-escape-quit)
;; (define-key key-translation-map (kbd "ESC") (kbd "C-g")) // PROBLEMAS CON EL TERMINAL
(defun super-escape()
  (interactive)
  (keyboard-escape-quit)
  (keyboard-quit)
  (setq quit-flag t))
```

```
(define-key global-map [escape] 'super-escape)

(define-key company-active-map [escape] 'company-abort)
```

Algunas teclas definidas a nivel global son sobrescritas por algunos modos (por ejemplo, prefiero que C-Z sea "deshacer"). Para poder definir teclas con prioridad sobre los demás modos defino un modo con mis atajos.

```
;; MIS TECLAS
(defvar mis-teclas-minor-mode-map
  (let ((map (make-sparse-keymap)))
    ; (define-key map (kbd "C-i") 'some-function)
    (define-key map (kbd "C-e") 'er/expand-region)
    (define-key map (kbd "C-S-e") 'er/contract-region)
    (define-key map (kbd "C-z") 'undo)
    (define-key map (kbd "C-x C-d") 'dired)
    (define-key map (kbd "C-x d") 'dired-other-frame)
    (define-key map (kbd "C-x C-b") 'ibuffer)
    (define-key map (kbd "C-x b") 'ibuffer)
    ; (define-key map (kbd "C-f") 'swiper-helm)
    (define-key map (kbd "C-f") 'helm-swoop)
    (define-key map (kbd "C-S-f") 'helm-multi-swoop-all)
    (define-key map (kbd "C-<f5>") 'reveal-y-pdf)
    (define-key map (kbd "<backtab>") 'psw-switch-buffer)
    (define-key map (kbd "M-I") 'popup-imenu)
    (define-key map (kbd "<f7>") 'imenu-list-smart-toggle)

    (define-key map (kbd "M-S-<up>") 'enlarge-window)
    (define-key map (kbd "M-S-<down>") 'shrink-window)
    (define-key map (kbd "M-S-<left>") 'shrink-window-horizontally)
    (define-key map (kbd "M-S-<right>") 'enlarge-window-horizontally)

    (define-key map (kbd "<f5>") 'transpose-frame)

    (define-key map (kbd "<f9>") 'magit-status)

    (define-key map (kbd "C-S-c C-S-c") 'mc/edit-lines)
    (define-key map (kbd "C->") 'mc/mark-next-like-this)
    (define-key map (kbd "C-<") 'mc/mark-previous-like-this)
    (define-key map (kbd "C-S-<mouse-1>") 'mc/add-cursor-on-click)
    (define-key map (kbd "C-S-c C-S-v") 'mc/mark-all-like-this)

    (define-key map (kbd "M-x") 'helm-M-x)
    (define-key map (kbd "C-x M-x") 'execute-extended-command)

    (define-key map (kbd "<menu>") 'helm-M-x)
    (define-key map (kbd "C-x C-f") 'helm-find-files)
    (define-key map (kbd "<f6>") 'helm-mini)
    (define-key map (kbd "M-y") 'helm-show-kill-ring)
    (define-key map (kbd "C-x r b") 'helm-filtered-bookmarks)

    (define-key map (kbd "<f8>") 'neotree-project-root-dir-or-current-dir)
    (define-key map (kbd "C-<f8>") 'ibuffer-sidebar-toggle-sidebar)

    (define-key map (kbd "C-x o") 'switch-window)

    (define-key map (kbd "C-o") 'dumb-jump-go)

    (define-key map (kbd "C-.") 'company-complete)

    (define-key map (kbd "C-S-l") 'toggle-truncate-lines)

    map)
  "mis-teclas-minor-mode keymap")

(define-minor-mode mis-teclas-minor-mode
  "A minor mode so that my key settings override annoying major modes.")
```

```
:init-value t
:lighter "mis-teclas")

(mis-teclas-minor-mode 1)
```

8 Utilidades

8.1 Convierto el buffer actual a una frame nueva

```
(defun saca-a-nueva-frame()
  (interactive)
  (let ((buffer (current-buffer)))
    (unless (one-window-p)
      (delete-window))
    (display-buffer-pop-up-frame buffer nil)))
```

8.2 Inicio de una selección rectangular usando el ratón (lo uso poco, prefiero C-x spc)

```
;; https://emacs.stackexchange.com/questions/7244/enable-emacs-column-selection-using-mouse
(defun mouse-start-rectangle (start-event)
  (interactive "e")
  (deactivate-mark)
  (mouse-set-point start-event)
  (rectangle-mark-mode +1)
  (let ((drag-event))
    (track-mouse
     (while (progn
              (setq drag-event (read-event))
              (mouse-movement-p drag-event))
            (mouse-set-point drag-event)))))

(global-set-key (kbd "S-<down-mouse-1>") #'mouse-start-rectangle)
```

8.3 Abrir el fichero del buffer actual con un programa externo

```
;; http://pages.sachachua.com/.emacs.d/Sacha.html
(defun abrir-programa-externo (arg)
  "Open visited file in default external program.

With a prefix ARG always prompt for command to use."
  (interactive "P")
  (when buffer-file-name
    (async-shell-command (concat
                          "setsid -w "
                          (cond
                           ((and (not arg) (eq system-type 'darwin)) "open")
                           ((and (not arg) (member system-type '(gnu gnu/linux gnu/kfreebsd))) "xdg-open")
                           (t (read-shell-command "Open current file with: ")))
                          " "
                          (shell-quote-argument buffer-file-name)))
      (run-at-time "2" nil
        (lambda() (winner-undo))))))
```

8.4 Copiar el nombre del fichero actual al portapapeles

```
;; http://pages.sachachua.com/.emacs.d/Sacha.html
(defun copiar-nombre-fichero-actual ()
  "Copy the current buffer file name to the clipboard."
  (interactive)
  (let ((filename (if (equal major-mode 'dired-mode)
                      default-directory
                      (buffer-file-name))))
    (when filename
      (kill-new filename)
      (message "Copied buffer file name '%s' to the clipboard." filename))))
```

8.5 Arrancar el servidor http de emacs en el directorio actual

```
(defun servidor-httpd-aqui (directory host port)
  "Abre un servidor http en un directorio."
  (interactive (list
    (read-directory-name "Root directory: " default-directory nil t)
    (read-string "Host: " "127.0.0.1" )
    (read-number "Port: " 8080)))

  (setq httpd-root directory)
  (setq httpd-host host)
  (setq httpd-port port)
  (httpd-start)
  (browse-url (concat "http://localhost:" (number-to-string port) "/")))
```

8.6 Al visitar un fichero, reabrir el bufer como root, incluso a través de tramp. Hay una versión para emacs25 y otra para emacs26.

```
(defun abrir-como-root-emacs25 ()
  "Reabre el fichero actual como root, incluso via tramp."
  (interactive)
  (let*
    ((sudo (/= (call-process "sudo" nil nil "-n true") 0))
     (file-name
      (if (tramp-tramp-file-p buffer-file-name)
          (with-parsed-tramp-file-name buffer-file-name parsed
            (tramp-make-tramp-file-name
              (if sudo "sudo" "su")
              "root"
              parsed-host
              parsed-localname
              (let ((tramp-postfix-host-format "|")
                    (tramp-prefix-format))
                (tramp-make-tramp-file-name
                  parsed-method
                  parsed-user
                  parsed-host
                  ""
                  parsed-hop))))
          (concat (if sudo
                     "/sudo::"
                     "/su::")
                  buffer-file-name)))
     (find-alternate-file file-name)))

;; REABRIR COMO ROOT
(defun abrir-como-root ()
  "Reabre el fichero actual como root, incluso via tramp."
  (interactive)
  (let*
    ((sudo (/= (call-process "sudo" nil nil "-n true") 0))
     (file-name
      (if (tramp-tramp-file-p buffer-file-name)
```

```

(with-parsed-tramp-file-name buffer-file-name parsed
  (tramp-make-tramp-file-name
    (if sudo "sudo" "su")
    "root"
    nil ; domain
    parsed-host
    nil ; port
    parsed-localname
    (let ((tramp-postfix-host-format "|")
          (tramp-prefix-format))
      (tramp-make-tramp-file-name
        parsed-method
        parsed-user
        nil ; domain
        parsed-host
        nil ; PORT
        parsed-hop))))

(concat (if sudo
          "/sudo:/"
          "/su:/"
          buffer-file-name))))
(find-alternate-file file-name))

```

8.7 *Transmission*

Cuando hay que añadir muchos torrents similares, es muy útil hacerlo desde un buffer de emacs.

```

;; CONECTAR A TRANSMISSION
(defun conectar-a-transmission ()
  (interactive)

  (setq transmission-host (read-string "Transmission host: " "192.168.1.254" ))
  (setq transmission-user (read-string "Transmission user: " "transmission"))
  (setq transmission-pass (read-passwd "Transmission password: "))

  (message "Conectando a %s@%s" transmission-user transmission-host)

  (setq transmission-rpc-auth (list 'username transmission-user 'password transmission-pass))

  (transmission))

```

8.8 Generar *reveal* y PDF

Generalmente utilizo un fichero orgmode para hacer transparencias y materiales para clase, y quiero generar a la vez las transparencias, la versión HTML y el PDF.

```

(defun reveal-y-pdf ()
  "Crea transparencias de reveal y hace el pdf a la vez."
  (interactive)
  (org-html-export-to-html)
  (let* (
    (filename (buffer-file-name))
    (html-filename (concat (file-name-sans-extension filename) ".html"))
    (html-wp-filename (concat (file-name-sans-extension filename) ".wp.html")) )
    (message "Copiando fichero: %s -> %s" html-filename html-wp-filename)
    (copy-file html-filename html-wp-filename t) )

  (org-reveal-export-to-html)
  (let* (
    (filename (buffer-file-name))
    (html-filename (concat (file-name-sans-extension filename) ".html"))
    (html-reveal-filename (concat (file-name-sans-extension filename) ".reveal.html")) )
    (message "renombrando fichero: %s -> %s" html-filename html-reveal-filename)
    (rename-file html-filename html-reveal-filename t))

  (org-latex-export-to-pdf)

```

```
(let* (
  (filename (buffer-file-name))
  (tex-filename (concat (file-name-sans-extension filename) ".tex")))

(message "Borrando fichero: %s" tex-filename)
(delete-file tex-filename) ) )
```

8.9 Función para decodificar una URL

```
(defun url-decode-region (start end)
  "Replace a region with the same contents, only URL decoded."
  (interactive "r")
  (let ((text (url-unhex-string (buffer-substring start end))))
    (delete-region start end)
    (insert text)))
```

8.10 Horario

Este es mi horario lectivo (sin incluir guardias y otras horas que no son de docencia directa a alumnos)

```
(defun horario()
  (interactive)
  (cfw:open-ical-calendar "https://calendar.google.com/calendar/ical/ags.iesavellaneda%40gmail.com/
  ↪ private-8d8f10c04ef7daee164d8d8a8f4707d5/basic.ics"))
```

8.11 Proxy de educamadrid

Durante una temporada, en los colegios de la Comunidad de Madrid era obligatorio el uso de un proxy.

```
(defun quitar-proxy()
  (interactive)
  (setq url-proxy-services ' ()))

(defun proxy-educamadrid()
  (interactive)
  (setq url-proxy-services
    ' (("no_proxy" . "^\\(localhost\\|10\\..*|192\\..*\\)")
      ("http" . "213.0.88.85:8080")
      ("https" . "213.0.88.85:8080"))))
```

8.12 Inserta la imagen del portapapeles en orgmode.

No lo uso mucho, quizás si cambio el nombre autogenerado sea más útil.

```
(defun org-insert-clipboard-image()
  "Save the image in the clipboard into a time stamped unique-named file in the same directory as the
  ↪ org-buffer and insert a link to this file."
  (interactive)
  ; (setq tilde-buffer-filename (replace-regexp-in-string "/" " \" (buffer-file-name) t t))
  (setq filename
    (concat
      (make-temp-name
        (concat buffer-file-name
          "_ "
          (format-time-string "%Y%m%d_%H%M%S_")) ) ".png"))
  ;; Linux: ImageMagick:
  ; (call-process "/bin/bash" nil (list filename "kk") nil "-c" "xclip -selection clipboard -t image/png -o")
  (call-process "xclip" nil (list :file filename) nil "-selection" "clipboard" "-t" "image/png" "-o")
  (insert (concat "[[file:" filename "]]"))
  (org-display-inline-images))
```

8.13 Eliminar el resto de buffers y ventanas

```
(defun kill-other-buffers ()
  "Kill all other buffers."
  (interactive)
  (mapc
   'kill-buffer
   (delq (current-buffer)
        (remove-if-not
         ' (lambda (x)
              (or (buffer-file-name x)
                  (eq 'dired-mode (buffer-local-value 'major-mode x))))
         (buffer-list)))))
```

8.14 Convertir la selección en un bloque de código de orgmode

```
(defun org-code-block-from-region (beg end &optional results-switches inline)
  "Copiado de org-babel-exemplify-region"
  (interactive "*r")
  (let ((maybe-cap
        (lambda (str)
          (if org-babel-uppercase-example-markers (upcase str) str))))
    (if inline
        (save-excursion
          (goto-char beg)
          (insert (format org-babel-inline-result-wrap
                          (delete-and-extract-region beg end))))
        (let ((size (count-lines beg end)))
          (save-excursion
            (cond ((= size 0)) ; do nothing for an empty result
                  (t
                   (goto-char beg)
                   (insert (if results-switches
                              (format "%s%s\n"
                                      (funcall maybe-cap "#+begin_src")
                                      results-switches)
                              (funcall maybe-cap "#+begin_src\n"))))
                   (let ((p (point)))
                     (if (markerp end) (goto-char end) (forward-char (- end beg)))
                     (org-escape-code-in-region p (point)))
                     (insert (funcall maybe-cap "#+end_src\n")))))))))
```

8.15 Diferencias con la última versión de git, usando git-gutter

```
(defun diferencias-git (&optional diffinfo)
  "Popup current diff hunk."
  (interactive)
  (git-gutter:awhen (or diffinfo
                        (git-gutter:search-here-diffinfo git-gutter:diffinfos))
    (save-selected-window
     ;; (pop-to-buffer (git-gutter:update-popuped-buffer it))
     (display-buffer-pop-up-frame (git-gutter:update-popuped-buffer it) nil)
     )
  )
)
```

9 Apariencia

9.1 Líneas vacías al final, como vim

```
(add-hook 'prog-mode-hook 'vim-empty-lines-mode)
(add-hook 'org-mode-hook 'vim-empty-lines-mode)
```

9.2 Nivel de indentación

```
(setq highlight-indent-guides-method 'fill)
```

9.3 Indicación de cambios de *git*

Utilizo *git* para casi todos mis ficheros. *git-gutter* marca en el margen izquierdo las líneas cambiadas, añadidas o borradas respecto de la versión de la rama actual. indico que se refresquen los buffers cada 10 segundos.

```
(global-git-gutter-mode +1)
(setq git-gutter:update-interval 10)
```

9.4 Saltos de página

Mostrar $\wedge L$ (saltos de página) como una línea horizontal

```
(global-page-break-lines-mode)
```

9.5 Modo proyección o modo trabajo

Utilizo emacs de dos modos muy distintos: para trabajar y para proyectar en clase. Estas dos funciones cambian opciones de visualización adecuadas para cada ocasión. He deshabilitado la indentación en los modos de programación, ralentiza bastante en los ficheros grandes.

```
(defun bonito-para-proyector()
  (interactive)
  (bonito-para-codigo)
  (toggle-truncate-lines -1)
  (highlight-indent-guides-mode 0)
  (if (>= emacs-major-version 26)
      (display-line-numbers-mode 0))
  (org-display-inline-images))

(defun bonito-para-org()
  (interactive)
  (bonito-para-proyector)
  (electric-pair-local-mode 1))

(defun bonito-para-codigo()
  (interactive)
  (toggle-truncate-lines 1)
  (highlight-indent-guides-mode 0)
  (toggle-word-wrap 1)
  (if (>= emacs-major-version 26)
      (display-line-numbers-mode 1))
  (auto-highlight-symbol-mode 1)
  (yafolding-mode 1)
  (adaptive-wrap-prefix-mode 1))

(add-hook 'prog-mode-hook 'bonito-para-codigo)
(add-hook 'text-mode-hook 'bonito-para-proyector)
(add-hook 'org-mode-hook 'bonito-para-org)
(add-hook 'tex-mode-hook 'bonito-para-codigo)
```

9.6 *Fringes*

Prefiero ocultar las flechas que indican que una línea se sale de la pantalla, y solo mostrar las de la derecha.

```
(fringe-mode '(0 . nil))
```

9.7 Temas

Tengo dos temas, claro y oscuro. El tema `alvaro` cambia algunos tamaños de letra (no colores).

Hay que marcar los temas como seguros. Para eso se deben registrar sus huellas en `custom-safe-themes` (lo he copiado del fichero `custom.el`).

```
(setq custom-safe-themes (quote
("6e219d6b6a3f7e22888b203fd5492e12133ba40512be983858f05b42806fa573"
"1b8d67b43ff1723960eb5e0cba512a2c7a2ad544ddb2533a90101fd1852b426e"
"b53db91fd0153783f094a2d5480119824b008f158e07d6b84d22f8e6b063d6e2" default)))

(defun tema-oscuro()
  (interactive)
  (disable-theme 'intellij)
  (load-theme 'sanityinc-tomorrow-bright)
  (load-theme 'alvaro t))

(defun tema-claro ()
  (interactive)
  (disable-theme 'sanityinc-tomorrow-bright)
  (load-theme 'intellij t)
  (load-theme 'alvaro t))
```

Pongo uno detrás de otro para "limpiar" lo que haya podido quedarse de alguna customización. Por lo visto, un tema siempre añade cambios, pero al quitarse no se deshacen completamente. Algunas configuraciones solo se tienen en cuenta al reiniciar *Emacs* o al reaplicar un modo: por ejemplo, los colores de `highlight-indent-guides` necesitan reabrir el buffer.

```
(tema-claro)
(tema-oscuro)
```

10 *Customize*

El fichero de `customize` lo mantengo aparte del `init.el`, para separar entornos y mejor integración con el control de versiones.

```
(setq custom-file "~/emacs.d/custom-file.el")
(load custom-file)
```

11 Futuras adiciones

En <https://github.com/caisah/emacs.dz> hay una colección de configuraciones *Emacs* muy interesantes. Esta es una lista de paquetes a investigar, extraída de los que se usan en esas configuraciones

- Elfeed
- Zoom-frm