Configuração do Emacs

fonte

Introdução

O objetivo principal deste documento é demonstrar um pouco de Literate Programming utilizando o Emacs juntamente com o org-mode. Ao término será possível gerar a documentação e o fonte do arquivo de configuração **eu** que estou utilizando no momento para o Emacs.

Esta configuração não é específica para os usuários do Vim (pelo menos para os que querem algo mais semelhante). Se você é um feliz usuário do Vim e deseja aprender Emacs para usar suas facilidades como o org-mode, etc., você deverá visitar o Spacemacs, Spacelite, Doom e outros. Eles configuram o Emacs de forma que você se sinta mais a vontade com relação aos atalhos do Vim e/ou visualmente.

Só no Melpa, existem mais de 3.600 pacotes para você configurar seu Emacs de modo a ficar mais adequado para o que você vai fazer. O arquivo gerado por este documento pode servir de base. Partir do zero pode ser complicado. Pode ser melhor você procurar na internet o que ou como os usuários configuraram o ambiente para uma determinada finalidade. Se você trabalha com Ruby, pode verificar como este usuário configurou o ambiente dele. Ou uma configuração mais específica para PHP ou para Python, Elixir, Haskell, etc.. É a melhor configuração para ele mas já é um bom começo. No site da Sacha sempre tem as novidades da semana que incluem pacotes, dicas, etc.. Comece com alguma configuração existente, procure por outras e vá adaptando para o seu gosto pessoal.

Se você deseja utilizar esta configuração como base, leia normalmente este texto para ver a estrutura de cada etapa. O tópico de Adaptações pode auxiliá-lo sobre o que tirar, deixar e alterar.

Configuração do Emacs

A ordem dos arquivos onde o Emacs busca pelas configurações é:

- ~/emacs
- ~/emacs.el
- ~/.emacs.d/init.el

Os arquivos são em Emacs Lisp ou elisp, um dialeto de Lisp. Não é o objetivo ensinar a linguagem, mas é bem simples. Basicamente (função argumentos ...). Por exemplo, em (* 1 2 3 4 5), a função multiplicação recebe os argumentos 1 2 3 4 e 5. Nem é preciso dizer que o resultado será 120. Os parênteses são utilizados para definir a prioridade.

Geração dos arquivos

Configuração (.emacs)

Para a geração da configuração é necessário abrir este arquivo com o Emacs (uma instalação nova já seria o suficiente) e digitar `C-c C-v t`. Significa pressionar `Control+c`, `Control+v` e a tecla `t`. Todos os códigos desejados serão gravados no arquivo **~/.emacs**. Para que a nova configuração tenha efeito, o modo mais simples é sair do Emacs (`C-x C-c`) e iniciá-lo novamente.

Documentação (README.html)

Para a geração do arquivo de configuração, basta pressionar `C-e h h`. Para uma apresentação melhor dos documentos é necessário uma formatação adequada. A formatação utilizada aqui é a ReadTheOrg encontrada em org-html-themes.

README para o GitHub

Não é necessário a geração. O GitHub entende (não totalmente) o formato .org. Este é o motivo de gerar a documentação como README.html. Nada impediria que você trocar o nome do arquivo. Por praticidade, deixei assim.

Estrutura do arquivo .emacs

Aspecto visual

Inicialmente iremos definir o tema, tamanho da janela e outros aspectos visuais.

```
;; Gerado por:
;; https://github.com/guaracy/.emacs.d
;;
;; texto da barra de título
(setq frame-title-format '(buffer-file-name "%f" ("%b")))
;; tamanho da janela
(setq initial-frame-alist
      '((width . 130)
        (height . 40)))
;; elementos visuais
(tool-bar-mode -1)
(scroll-bar-mode -1)
;; apresentação/entrada do texto
(setq inhibit-startup-message t)
(setq inhibit-splash-screen t)
(setq initial-scratch-message nil)
(setq truncate-lines t)
(show-paren-mode 1)
(setq-default indent-tabs-mode nil)
(setq-default tab-width 4)
(global-linum-mode t)
(global-hl-line-mode 1)
(ido-mode t)
;; modeline
(line-number-mode 1)
(column-number-mode 1)
(setq initial-scratch-message
```

```
      ";; Nada neste buffer será salvo. Use:\n;; Ctrl+x Ctrl+r /
Ctrl+x Ctrl+f para ler um arquivo.\n")
```

Repositórios

O próximo passo é definir onde estarão nossos pacotes. Existem pacotes no GitHub e em diversos outros locais. O MELPA é um local que agrega os pacotes facilitando a localização dos mesmo.

```
(require 'package)
(setq package-archives
      '(("gnu" . "https://elpa.gnu.org/packages/")
        ("melpa" . "http://melpa.org/packages/")))
(package-initialize)
(when (not package-archive-contents)
  (package-refresh-contents))
```

Instalação dos pacotes

Existem diversas formas de instalar os pacotes. Será utilizado o use-package para deixar as coisas mais ordenadas e facilitar a inclusão/exclusão (a exclusão pode ser temporária)

```
(unless (package-installed-p 'use-package)
  (package-refresh-contents)
  (package-install 'use-package))
(require 'use-package)
(use-package org-ioslide
  :disabled
  :ensure t)
```

Configurações para auxiliar a digitação

swiper

Facilita a busca e localização no arquivo.

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

which-key

Mostra os atalhos disponíveis quando o usuário digita `Ctrl-x`, `Ctrl-c`, `Ctrl-h`, etc. e não digita imediatamente a sequência.

```
(use-package which-key
  :ensure t
  :config
  (which-key-mode)
  (setq which-key-idlw-delay 0.5))
```

smex

Facilita a execução de comandos quando o usuário digita `Alt-x` (comandos gerais) ou `Alt-X` (comandos do modo atual do buffer).

```
(use-package smex
  :ensure t
  :bind (("M-x" . smex)
         ("M-X" . smex-major-mode-commands)))
```

iedit

Parecido com multiple-cursors, permite a edição simultânea de multiplas regiões. Pessoalmente, acho interessante o fato de =C-'= esconder o resto do texto para uma vizualização geral das ocorrências, editar apenas as ocorrências de uma função ou linha.

```
(use-package iedit
  :ensure t
  :bind (("C-;" . iedit-mode)
         ("C-RET" . iedit-rectangle-mode)))
```

multiple-cursors

Trabalha com multiplos cursores. Parecido com iedit anterior mas tem algumas vantagens como os cursores mais flexíveis. Na página tem mais informações. Este vídeo tem um exemplo muito interessante para a utilização. Existem diversos vídeos no site cobrindo outros assuntos.

```
(use-package multiple-cursors
  :ensure t
  :bind (("C-S-c C-S-c" . mc/edit-lines)
         ("C->" . mc/mark-next-like-this)
         ("C-<" . mc/mark-previous-like-this)))
```

ido-yes-or-no

Facilita respostas com yes/no permitindo apenas o pressionamento de y/n e enter.

```
(use-package ido-yes-or-no
  :ensure t
  :config
  (ido-yes-or-no-mode t))
```

auto-complete

Mostra um menu para auxiliar a completar o que está sendo digitado.

```
(use-package auto-complete
  :ensure t
  :config
  (ac-config-default))
```

Configurações visuais

zerodark-theme

Achei o thema interessante. Você pode ir em Emacs Themes e ver outros que você pode gostar mais.

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

ido-grid-mode

Apenas para combinar mais com a apresentação do which-key. Mostra as opções em grade em vez de linha.

```
(use-package ido-grid-mode
  :ensure t
  :config
  (setq ido-enable-flex-matching t)
  (setq ido-everywhere t)
  (ido-mode t)
  (ido-grid-mode t))
```

ido-select-window

Facilita a seleção da janela quando existem diversos buffers visualizados simultaneamente.

```
(use-package ido-select-window
  :ensure t
  :bind ("C-x o" . ido-select-window))
```

hlinum-mode

Extende a seleção da linha atual para o número da linha. Veja outras configurações para linum

```
(use-package hlinum
  :ensure t
  :config
  (hlinum-activate))
```

linum-off

Desabilita número de linhas em determnados modos.

```
(use-package linum-off
  :ensure t)
```

indent-guide

Mostra uma linha para indentação do código para facilitar a visualização. Outra opção é o highlight-indentention-for-Emacs

```
(use-package indent-guide
  :ensure t
  :config
  (indent-guide-global-mode))
```

rainbow-delimiters <>

Optei por utilizar apenas duas cores utilizando as cores do tema para **keyword** e **string**.

```
(use-package rainbow-delimiters
  :ensure t
  :config
  (add-hook 'prog-mode-hook #'rainbow-delimiters-mode)
  (require 'color)
  (defvar my-paren-dual-colors    (ref:sc-rd1)
    '((face-attribute 'font-lock-string-face :foreground)
      (face-attribute 'font-lock-keyword-face :foreground)))
  (setq rainbow-delimiters-outermost-only-face-count 0)
  (setq rainbow-delimiters-max-face-count 2)
  (set-face-foreground 'rainbow-delimiters-depth-1-face
                       (eval (elt my-paren-dual-colors 1)))
  (set-face-foreground 'rainbow-delimiters-depth-2-face
```

```
                        (eval (elt my-paren-dual-colors 0)))
(ref:sc-rd2)
                        )
```

Utilitários

restart-emacs

Permite reicializar o Emacs de dentro do Emacs.

```
(use-package restart-emacs
  :ensure t
  :config
  (setq restart-emacs-restore-frames t))
```

htmlize

Permite exportar o conteúdo de um buffer para uma página .html com o mesmo tema utilizado pelo emacs bem como salientar a sintaxe dos fontes no org-mode.

```
(use-package htmlize
  :ensure t
  :defer t)
```

neotree

Permite a utilização do neotree para navegar no sistema de arquivos em vez do dired. Assinala `Ctrl-\` para mostrar/esconder neotree.

```
(use-package neotree
  :ensure t
  :bind ("C-\\" . neotree-toggle))
```

paradox

Um gerenciador de pacotes melhorado.

```
(use-package paradox
  :ensure t
  :config
  (paradox-enable))
```

magit

Interface com o Git

```
(use-package magit
  :ensure t
  :bind ("C-x g" . magit-status))
```

el2org e ox-gfm

O `el2org` é necessário para o `org-webpage` e o `ox-gfm` é necessário para `el2org`.

```
(use-package ox-gfm
  :ensure t)
(use-package el2org
  :ensure t)
```

org-webpage <>

Para gerenciamento do blog estático no github.

```
(use-package org-webpage
  :ensure t
  :defer t)
(org2web-add-project
 '("cadafalso"
   :repository-directory "~/projetos/org2blog"
   :remote (git
"https://github.com/guaracy/guaracy.github.com.git" "master")
   :site-domain "http://guaracy.github.io/"
   :site-main-title "Cadafalso"
   :site-sub-title "Apenas mais um blog"
   :default-category "blog"
   :theme (org)
```

```
   :source-browse-url ("Github"
"https://github.com/guaracy/guaracy.github.com")
   :web-server-port 7654))
```

Linguagens

Pacotes para auxiliar a entrada de programas/arquivos em determinada linguagem

red.el <>

Salientar sintaxe para arquivos Red e Red/System. Utilizo a cópia local do GitHub. Como Red ainda está em desenvolvimento, ainda faltam diversas coisas. Mas o básico como salientar a sintaxe e executar/exportar códigos em documentos .org estão funcionando.

```
(use-package red
  :load-path "~/github/guaracy/red.el"
  :config
  (autoload 'red-mode "red.el" "Major mode for Red development"
t)
  (add-to-list 'auto-mode-alist '("\\.red$" . red-mode))
  )
```

less-css-mode

Para edição de arquivo .less para gerar .css (como não gosto e não precisei de s[a|c]ass). Para compilar e gerar o .css basta pressionar `C-c C-c`.

```
(use-package less-css-mode
  :ensure t)
```

Configurações adicionais

Algumas configurações deixadas para o final

```
(setq org-confirm-babel-evaluate nil)
(setq org-support-shift-select t)
```

```
(setq org-support-shift-select 'always)
(setq org-html-htmlize-output-type 'css)
(setq org-src-fontify-natively t)
(setq org-export-default-language "pt_BR")
(add-hook 'org-mode-hook #'visual-line-mode)
(add-hook 'org-mode-hook #'toggle-word-wrap)
(add-to-list 'org-structure-template-alist '("t" "#+begin_tip ?
\n\n#+end_tip"))
(add-to-list 'org-structure-template-alist '("n" "#+begin_note ?
\n\n#+end_note"))
;; linguagens utilizadas por org-babel
(org-babel-do-load-languages
 'org-babel-load-languages
 '((emacs-lisp . t)
   (red . t)
   (python . t)
   (ruby . t)
   (R . t)
   (C . t)
   (ditaa . t)
   (shell . t)))
(defvar my-term-shell "/usr/bin/zsh")
(defadvice ansi-term (before force-bash)
  (interactive (list my-term-shell)))
(ad-activate 'ansi-term)
```

Definição de funções

Agora a definição de algumas funções que achei úteis (YMMV)

Abrir lista de arquivos recente (C-x C-r))

```
(require 'recentf)
(recentf-mode t)
(setq recentf-max-menu-items 25)
(defun recentf-ido-find-file ()
  "Find a recent file using Ido."
  (interactive)
  (let ((file (ido-completing-read "Choose recent file: "
recentf-list nil t)))
    (when file
      (find-file file))))
(global-set-key (kbd "C-x C-r") 'recentf-ido-find-file)
;; ------  fim do arquivo de configuração (ref:sc-fim)
```

Pronto. E essas são as (sc-fim) linhas de configuração.

Adaptações <>

Como disse antes, serão necessárias algumas alterações. Se você não deseja algum pacote, você pode apenas excluir toda a parte existente no arquivo .org. Se você não quer no momento ou não sabe se será preciso, inclua `:disabled` antes de `:ensure t`.

Leia a documentação para ver o resumo de cada pacote e/ou seguir o link do pacote para maiores informações.

Requisitos

- Ter o Emacs (versão 24.4 ou maior) instalado (instale pelo gerenciador de pacotes da sua distribuição).
- Não possuir o arquivo "~/.emacs" (se existir, faça backup e apague o existente pois será **destruído**).

Iniciando

- Digite `git clone https://github.com/guaracy/.emacs.d ~/.emacs.d` e pressione enter para baixar este repositório.
- Digite no prompt `emacs ~/.emacs.d/README.org` para editar o arquivo README.org e siga os passos abaixo.

Alterando

- Faça as alterações desejadas, incluindo, eliminando ou desabilitando os pacotes desejados.
- Ajuste os parâmetros dos pacotes para as suas necessidades/configurações.

Gerando

- Salve o arquivo alterado digitando `C-x C-s`
- Gere a documentação digitando `C-c e h h`
- Gere o arquivo ~/.emacs digitando `C-c C-v t`
- Encerre o Emacs digitando `C-x C-c`

Agora é só iniciar o Emacs que os pacotes necessários serão baixados e as novas configurações serão aplicadas.

Problemas

Você pode tirar dúvidas consultando o Google o DuckDuckGo, por exemplo, no stackoverflow e reddit entre outros.

Como qualquer programa, o Emacs pode apresentar problemas.

Fim