Configuration

Kalelzar

August 1, 2020

Contents

```
;; config.el
```

1 Personal Information

```
(setq user-full-name "Borislav Atanasov"
      user-mail-address "natomanofglory@gmail.com")
```

2 User Interface

I do enjoy myself a good looking user interface. In fact customization of graphical elements is one of the reason I use Emacs.

2.1 Unicode

Enable unicode fonts using the suprisingly named package unicode-fonts
This does require that some Unicode fonts exists.

```
yay -S ttf-symbola quivira ttf-dejavu noto-fonts noto-fonts-emoji noto-fonts-extra
```

Setting up this package for the first time may take upwards of 5 minutes the first time you start Emacs. There is a lot of unicode characters.

```
(use-package unicode-fonts
  :ensure t
  :init (unicode-fonts-setup))
```

2.2 Themes

2.2.1 Pywal

Currently I use pywal to dynamically generate a colour scheme on the fly from my current background, which I change automatically every 5 minutes

Ideally I would apply that colour scheme to my Emacs theme as well.

```
t
```

2.2.2 ewal

Thankfully ewal exist so I can just use that.

Now we need a theme that knows how to apply the scheme colours. ewal-doom-themes looks pretty nice.

We do need to configure some things so that the ewal theme is reapplied every time the background changes, since apparently that is not a common circumstance.

1. **TODO** Emacs Client background bug Oddly ewal themes look horrid in the terminal when executed with emacsclient. The background colour makes the buffers basically unreadable.

2.2.3 Safe themes

Mark all themes as safe for simplicity.

```
(setq custom-safe-themes t)
```

2.3 Org Appearance

Org mode is something I use quite often (case in point) so I would prefer it would look fairly decent.

```
(setq org-fontify-done-headline t
      org-fontify-whole-heading-line t
      org-src-fontify-natively t
      org-src-window-setup 'current-window
      org-src-strip-leading-and-trailing-blank-lines t
      org-src-preserve-indentation t)
```

I set headlines to fontify the whole line as well as change the face when marked DONE. Also fontify code blocks.

Obviously we *want* to `display` emphasis markers as what they do rather than some ~~random characters~~.

```
(setq org-hide-emphasis-markers t)
```

And we want some fancy UTF8 characters for entries

```
(setq org-pretty-entities t)
```

Since I write in LaTeX a lot I would prefer if LaTeX things were being highlighted.

```
(setq org-highlight-latex-and-related (quote (native script entities)))
```

Finally replace the default ... when a heading is collapsed with a fancy unicode arrow

```
(setq org-ellipsis "")
```

2.4 Icons

Enable icons in various buffers with all-the-icons.

```
(use-package all-the-icons
  :ensure t)

(use-package all-the-icons-ibuffer
  :ensure t
  :init (all-the-icons-ibuffer-mode 1))

(use-package all-the-icons-gnus
  :ensure t
  :init (all-the-icons-gnus-setup))
```

```
(use-package all-the-icons-dired
  :ensure t
  :init (add-hook 'dired-mode-hook 'all-the-icons-dired-mode))
```

2.5 Font Ligatures

Font ligatures sure are nice.

I happen to know that the Fira Code ones are doubly so.

First we need to set the default font to Fire Code.

That requires that it is installed on the system of course.

Thankfully I happen to know that a nice Fira Code package exists in the AUR.

```
yay -S "otf-fira-code-symbol" "ttf-fira-code"
```

We also need to set the Fira Code as the actual font for emacs.

```
(add-to-list 'default-frame-alist
             (cond
              ((string-equal system-type "gnu/linux") '(font . "Fira Code-12"))))
```

```
(use-package fira-code-mode
    :ensure t
    :init (define-globalized-minor-mode global-fira-code-mode fira-code-mode
  (lambda () (fira-code-mode 1))))
```

Make a global minor mode for Fira code font ligatures.

2.6 Modeline

Enable doom-modeline.

```
(use-package doom-modeline
  :ensure t
  :init (doom-modeline-mode 1))
```

Enable icons in the modeline

```
(setq doom-modeline-icon t)
```

Don't show time in the Emacs modeline. I have the Stumpwm modeline for that.

```
  (display-time-mode 0)
```

Diplay the column number in the modeline

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

The following function for occur-dwim is taken from Oleh Krehel from his blog post at (or emacs. It takes the current region or the symbol at point as the default value for occur.

```
(defun occur-dwim ()
  "Call `occur' with a sane default."
  (interactive)
  (push (if (region-active-p)
            (buffer-substring-no-properties
             (region-beginning)
             (region-end))
          (thing-at-point 'symbol))
        regexp-history)
  (call-interactively 'occur))

(bind-key "M-s o" 'occur-dwim)
```

Make page breaks pretty instead of `^L`. See also this article.

```
  (use-package page-break-lines :ensure t)
```

2.7 Ibuffer

Use Ibuffer by default.

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

  (add-hook 'dired-mode-hook 'auto-revert-mode)

  ;; Also auto refresh dired, but be quiet about it
  (setq global-auto-revert-non-file-buffers t)
  (setq auto-revert-verbose nil)
```

Save recent files.

```
  (use-package recentf
    :config
    (recentf-mode t)
    (setq recentf-max-saved-items 500))
```

2.8 Misc

Disable fringes.

```
  (fringe-mode 0)
```

Disable the blinking cursor.

```
(blink-cursor-mode -1)
```

Show matching parenthesis.

```
  (show-paren-mode t)
```

Wrap lines properly

```
(global-visual-line-mode)
(diminish 'visual-line-mode)
```

Smooth scrolling

```
(use-package smooth-scrolling
  :ensure t
  :config
  (smooth-scrolling-mode))
```

3 Misc

Set the cusomize file to a separate file.

```
 (setq custom-file (expand-file-name "custom.el" user-emacs-directory))
 (load custom-file)
```

Set up Language Server Protocol

```
  (use-package lsp-mode :commands lsp)
  (use-package lsp-ui :commands lsp-ui-mode)
  (use-package company-lsp :commands company-lsp)
```

Kill. Whole. Lines. This should be the default.

```
(setq kill-whole-line t)
```

I don't actually know what this does but the EmacsWiki told me to put it in.

```
(autoload 'wl "wl" "Wanderlust" t)
```

Watch my emacs activity with ActivityWatch

```
(use-package activity-watch-mode
 :ensure t)
(global-activity-watch-mode)
(diminish 'activity-watch-mode)
```

Global semantic mode for my programming needs.

```
(semantic-mode 1)
```

Set zathura as the default pdf viewer

```
sudo pacman -S zathura zathura-pdf-mupdf
```

```
(with-eval-after-load 'tex
  (add-to-list 'TeX-view-program-selection
               '(output-pdf "Zathura")))
```

More convenient `*scratch*` buffers

```
(use-package scratch
  :ensure t
  :commands scratch)
```

Undo trees

```
(use-package undo-tree
  :ensure t)
```

Crux

```
(use-package crux
  :ensure t
  :bind (("C-c o o" . crux-open-with)
         ("C-c o u" . crux-view-url)))
```

Revert buffer automatically when underlying file is changed outside of Emacs.

```
(global-auto-revert-mode t)
```

Smarter tab.

```
(setq tab-always-indent 'complete)
```

Switch between visible buffers with shift + arrow keys

```
(use-package windmove :ensure t)
(windmove-default-keybindings)
```

Save buffers on buffer switch

```
(use-package super-save :ensure t)
;; add integration with ace-window
(add-to-list 'super-save-triggers 'ace-window)
(super-save-mode +1)
```

Highlighting

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

;(use-package volatile-highlights :ensure t)
;(volatile-highlights-mode 0)
;(diminish 'volatile-highlights-mode)
```

Add the ability to kill the current line without marking it

```
;(use-package rect :ensure t)
;(require 'rect)
;(crux-with-region-or-line kill-region)
```

Automatically clean up unused buffers.

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

Do not activate mark if there is no active region when `exchange-point-and-mark` (C-x C-x by default) is called.

```
(defadvice exchange-point-and-mark (before deactivate-mark activate compile)
  "When called with no active region, do not activate mark."
  (interactive
   (list (not (region-active-p)))))
```

Create macro for calling functions on region or buffer

```
;  (use-package tabify :ensure t)
  (defmacro with-region-or-buffer (func)
    "When called with no active region, call FUNC on current buffer."
    `(defadvice ,func (before with-region-or-buffer activate compile)
       (interactive
        (if mark-active
            (list (region-beginning) (region-end))
          (list (point-min) (point-max))))))

  (with-region-or-buffer indent-region)
;  (with-region-or-buffer untabify)
```

Show available keybindings after you start typing

```
;; show available keybindings after you start typing
(use-package which-key :ensure t)
(which-key-mode +1)
```

4 Sane Defaults

Some are sourced from Daniel Mai's config.

For some reason these functions are disabled.

```
(put 'downcase-region 'disabled nil)
(put 'upcase-region 'disabled nil)
(put 'narrow-to-region 'disabled nil)
(put 'dired-find-alternate-file 'disabled nil)
```

Yes is two letters too long for me.

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

Clean up back-ups / autosaves.

```
(setq backup-directory-alist '(("." . "~/.emacs.d/backups")))
(setq auto-save-file-name-transforms '((".*" "~/.emacs.d/auto-save-list/" t)))
```

UTF-8.

```
(setq locale-coding-system 'utf-8)
(set-terminal-coding-system 'utf-8)
(set-keyboard-coding-system 'utf-8)
(set-selection-coding-system 'utf-8)
(prefer-coding-system 'utf-8)
```

Don't indent with TABS please.

```
(setq-default indent-tabs-mode nil)
```

Indicate empty lines.

```
(setq-default indicate-empty-lines t)
```

Don't count two spaces after a period as the end of a sentence. Just one space is needed.

```
(setq sentence-end-double-space nil)
```

Delete the region when typing.

```
(delete-selection-mode t)
```

Logical buffer names.

```
(setq uniquify-buffer-name-style 'forward)
```

Load aliases from .bash$_{\text{profile}}$

```
(setq shell-command-switch "-ic")
```

Silence!

```
(setq visible-bell t)
(setq ring-bell-function 'ignore)
```

5 Modes

5.1 Yasnippet

```
(use-package yasnippet
  :ensure t
  :diminish yas-minor-mode
  :init
  ;(setq yas-snippet-dirs (concat user-emacs-directory "snippets"))
  (setq yas-indent-line 'fixed)
  (yas-global-mode))
```

5.2 Org

Fetch the latest version of org mode as per this instructions.

```
(use-package org :ensure org-plus-contrib)
```

5.2.1 Babel

Don't ask for confirmation when evaluating code blocks. It's annoying.

```
(setq org-confirm-babel-evaluate nil)
```

Enable some languages for evaluation in Org code blocks.

```
(org-babel-do-load-languages
 'org-babel-load-languages
 '((python . t)
   (C . t)
   (shell . t)))
```

5.2.2 Export

I mainly export to LaTeX but that comes installed by default.
I also occasionally have need to export to epub.

```
(use-package ox-epub
  :ensure t)

(setq org-export-backends
   (quote
    (ascii beamer html latex epub)))
```

Enable linting of source code blocks when exported to LaTeX This requires minted.

On Archlinux:

```
sudo pacman -S minted
```

You also might need to install some of the (La)TeX libraries included by your distribution.

```
(require 'ox-latex)
(add-to-list 'org-latex-packages-alist '("" "minted"))
(add-to-list 'org-latex-packages-alist '("" "color"))
(add-to-list 'org-latex-packages-alist '("" "xcolor"))
(setq org-latex-listings 'minted)

(setq org-latex-pdf-process
      '("pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"
        "pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"
        "pdflatex -shell-escape -interaction nonstopmode -output-directory %o %f"))
```

5.2.3 Templates

Source blocks

```
(add-to-list 'org-structure-template-alist '("el" . "src emacs-lisp"))
(add-to-list 'org-structure-template-alist '("py" . "src python"))
(add-to-list 'org-structure-template-alist '("sh" . "src sh"))
(add-to-list 'org-structure-template-alist '("bh" . "src bash"))
(add-to-list 'org-structure-template-alist '("sc" . "src scala"))
```

5.2.4 Misc

Set up emphasis symbols

```
(setq org-emphasis-alist
      (quote
       (("*" bold)
        ("/" italic)
        ("_" underline)
        ("=" org-verbatim verbatim)
        ("~" org-code verbatim)
        ("+"
         (:strike-through t)))))
```

Enable some good minor modes for working with org-mode when *in* org-mode.

```
(use-package org-superstar :ensure t)
(use-package org-sticky-header :ensure t)
(use-package cdlatex :ensure t)

(defun org-mode-enable-minor-modes-hook ()
  "Enable some good minor-modes for fancier 'org-mode' experience."
  (org-superstar-mode 1)
  (org-sticky-header-mode 1)
  (org-indent-mode 1)
  (org-cdlatex-mode 1)
  (yas-minor-mode 1)
  (fira-code-mode 1)
  )

(add-hook 'org-mode-hook 'org-mode-enable-minor-modes-hook)
```

Let TAB behave as expected when inside code block.

```
(setq org-src-tab-acts-natively t)
```

Set the default notes file.

```
(setq org-default-notes-file "~/Documents/notes.org")
```

Enable speed commands.

```
(setq org-use-speed-commands t)
```

```
(setq org-tags-column 45)
```

5.3 C/C++

Set LSP for C/C++ using ccls.

```
sudo pacman -S ccls
```

We also need the emacs package.

```
(use-package ccls :ensure t
  :hook ((c-mode c++-mode objc-mode cuda-mode) .
         (lambda () (require 'ccls) (lsp))))

(setq ccls-executable "/usr/bin/ccls")
```

Enable some refactoring with srefactor.

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

This package displays function signatures in the mode line.

```
(use-package c-eldoc
 :commands c-turn-on-eldoc-mode
 :ensure t
 :init (add-hook 'c-mode-hook #'c-turn-on-eldoc-mode))
```

5.4 Lisp

5.4.1 SLIME

Install the Superior Lisp Interaction Mode for Emacs.

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

Set the inferior Lisp program for SLIME to sbcl.

```
(setq inferior-lisp-program "sbcl")
```

Set up company for SLIME

```
(use-package slime-company :ensure t)
(slime-setup '(slime-company))
```

5.5 Edit Server

Enable editing of browser text fields in Emacs. Just because it's possible.

```
(use-package edit-server
  :ensure t
  :commands edit-server-start
  :init (if after-init-time
            (edit-server-start)
```

```
          (add-hook 'after-init-hook
                    #'(lambda() (edit-server-start))))
    :config (setq edit-server-new-frame-alist
                  '((name . "Edit with Emacs")
                    (minibuffer . t)
                    (menu-bar-lines . t)
                    (window-system . x))))
```

5.6 Reading

5.6.1 TODO Convert to major mode

Make reading stuff in Emacs easier.

```
(defun reading-mode ()
  "Enable a major mode and some minor modes useful for reading."
  (interactive)
  (fundamental-mode)
  (text-scale-set 1)
  (visual-line-mode 1)
  (set-frame-font "Roboto")
  (set-fill-column 65)
  (set-justification-full (point-min) (point-max))
  (set-left-margin (point-min) (point-max) 7)
  (split-window-horizontally)
  (follow-mode 1)
  (read-only-mode 1))
```

5.7 IDO

```
(use-package ido
  :disabled t
  :init
  (setq ido-enable-flex-matching t)
  (setq ido-everywhere t)
  (ido-mode t)
  (use-package ido-vertical-mode
    :ensure t
    :defer t
    :init (ido-vertical-mode 1)
    (setq ido-vertical-define-keys 'C-n-and-C-p-only)))
```

5.8 Whitespace

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

  (setq whitespace-line-column 80) ;; limit line length
  (setq whitespace-style '(face tabs empty trailing lines-tail))
  (global-whitespace-mode)

    (defun cleanup-on-save ()
     "Call `whitespace-cleanup' on save"
       (whitespace-cleanup))

    (add-hook 'before-save-hook 'cleanup-on-save)
```

5.9 Helm

```
(use-package helm
  :disabled t
  :ensure t
  :diminish helm-mode
  :bind (("C-c h" . helm-command-prefix)
         ("C-x b" . helm-mini)
         ("C-`" . helm-resume)
         ("M-x" . helm-M-x)
         ("C-x C-f" . helm-find-files)
         ("C-x C-r" . helm-recentf))
  :init
  (require 'helm-config)
  :config
  (setq helm-locate-command "mdfind -interpret -name %s %s"
        helm-ff-newfile-prompt-p nil
        helm-M-x-fuzzy-match t)
  (helm-mode))
(use-package helm-projectile
  :ensure t
  :after helm-mode
  :commands helm-projectile
  :bind ("C-c p h" . helm-projectile))
(use-package helm-ag
```

```
  :ensure t
  :after helm-mode)
(use-package helm-swoop
  :ensure t
  :after helm-mode
  :bind ("H-w" . helm-swoop))
```

5.10 Ivy

```
(use-package ivy
  :ensure t
  :diminish (ivy-mode . "")
  :bind
  (:map ivy-mode-map
   ("C-'" . ivy-avy))
  :config
  (ivy-mode 1)
  ;; add 'recentf-mode' and bookmarks to 'ivy-switch-buffer'.
  (setq ivy-use-virtual-buffers t)
  ;; number of result lines to display
  (setq ivy-height 10)
  ;; Show candidate index and total count
  (setq ivy-count-format "(%d/%d) ")
  ;; no regexp by default
  (setq ivy-initial-inputs-alist nil)
  ;; configure regexp engine.
  (setq ivy-re-builders-alist
        ;; allow input not in order
        '((t   . ivy--regex-ignore-order))))
(use-package avy
  :ensure t
  :bind ("C-S-s" . avy-goto-char))
(use-package counsel
  :ensure t
  :bind (("M-x" . counsel-M-x)
         ("C-x C-r" . counsel-recentf)
         ("C-c h i" . counsel-imenu)
         ("C-h v" . counsel-describe-variable)
         ("C-h f" . counsel-describe-function)))
(use-package counsel-projectile
```

```
  :ensure t
  :config
  (define-key projectile-mode-map (kbd "C-c p") 'projectile-command-map)
  (counsel-projectile-mode)
  (setq counsel-projectile-switch-project-action 'dired))
(use-package swiper
  :ensure t
  :bind (("C-s" . swiper)))
(ivy-mode 1)
```

5.11 Magit

A great interface for git projects. It's much more pleasant to use than the git interface on the command line. Use an easy keybinding to access magit.

```
(use-package magit
  :ensure t
  :defer t
  :bind (("C-c g" . magit-status)
         ("C-c G" . magit-dispatch)
         ("C-c m l" . magit-log-buffer-file)
         ("C-c m b" . magit-blame))
  :config
  (setq magit-display-buffer-function 'magit-display-buffer-same-window-except-diff-v1)
  (setq magit-diff-refine-hunk t))
```

> The following code makes magit-status run alone in the frame, and then restores the old window configuration when you quit out of magit.
>
> No more juggling windows after commiting. It's magit bliss.

From: Magnar Sveen

```
;; full screen magit-status
(defadvice magit-status (around magit-fullscreen activate)
  (window-configuration-to-register :magit-fullscreen)
  ad-do-it
  (delete-other-windows))

(defun magit-quit-session ()
```

```
  "Restores the previous window configuration and kills the magit buffer"
  (interactive)
  (kill-buffer)
  (jump-to-register :magit-fullscreen))
```

Magit extension for GitHub/GitLab

```
(use-package forge
  :ensure t
  :after magit)
```

5.12 Flycheck

```
(use-package flycheck
  :ensure t
  :defer 10
  :init (global-flycheck-mode)
  :config (setq flycheck-html-tidy-executable "tidy5"))
```

Enable flyspell

```
  (use-package flyspell
    :ensure t
    )
  (setq ispell-program-name "aspell"
                      ispell-extra-args "--sug-mode=ultra")
                (flyspell-mode 1)
```

5.13 Emacs Lisp

```
  (use-package macrostep
    :ensure t
    :bind (("H-`" . macrostep-expand)
           ("H-C-`" . macrostep-collapse)))
```

5.14 Projectile

Project management and navigation.

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

```
  :diminish projectile-mode
  :commands (projectile-mode projectile-switch-project)
  :bind (("C-c p p" . projectile-switch-project)
         ("C-c p s s" . projectile-ag)
         ("C-c p s r" . projectile-ripgrep))
  :config
  (setq projectile-keymap-prefix (kbd "C-c p"))
  (projectile-global-mode t)
  (setq projectile-enable-caching t)
  (setq projectile-switch-project-action 'projectile-dired))
```

5.15 Python

```
(use-package python-mode
  :defer t
  :ensure t)
```

5.16 Company

Auto completion

```
  (use-package company
    :ensure t
    :config
    (setq company-tooltip-limit 20)
    (setq company-idle-delay .15)
    (setq company-echo-delay 0)
    (setq company-begin-commands '(self-insert-command))
    (define-key company-active-map (kbd "C-n") #'company-select-next)
    (define-key company-active-map (kbd "C-p") #'company-select-previous))
```

5.17 Nov

Install nov so I can read epub files in Emacs.

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

Enable nov-mode for epub files.

```
  (add-to-list 'auto-mode-alist '("\\.epub\\'" . nov-mode))
```

Properly justify text. This requires justify-kp which is unfortunately not in MELPA, since it hasn't been updated since *<2019-11-19 Tue>*. It is still up on github though so we can do a quick clone.

```
cd "$HOME/.emacs.d/elpa/"
git clone "https://github.com/Fuco1/justify-kp"
```

It should also be available as a git submodule of my .emacs.d repo

```
(add-to-list 'load-path "~/.emacs.d/elpa/justify-kp/")
(require 'justify-kp)
(setq nov-text-width t)

(defun my-nov-window-configuration-change-hook ()
  (nov-justify-hook)
  (remove-hook 'window-configuration-change-hook
               'my-nov-window-configuration-change-hook
               t))

(defun nov-justify-hook ()
  (if (get-buffer-window)
      (
       let ((max-width (pj-line-width))
            buffer-read-only)
       (save-excursion
         (goto-char (point-min))
         (while (not (eobp))
           (when (not (looking-at "^[[:space:]]*$"))
             (goto-char (line-end-position))
             (when (> (shr-pixel-column) max-width)
               (goto-char (line-beginning-position))
               (pj-justify)))
           (forward-line 1)))
       (toggle-word-wrap 1)
       )
    (add-hook 'window-configuration-change-hook
              'my-nov-window-configuration-change-hook
              nil t)))

(add-hook 'nov-post-html-render-hook 'nov-justify-hook)
```

5.18 Tramp

```
(use-package tramp
  :ensure t
  :config (setq tramp-default-method "ssh"))
```

6 Keybindings

6.1 C

```
(define-key c-mode-map (kbd "C-c r") 'srefactor-refactor-at-point)
```

6.2 C++

```
(define-key c++-mode-map (kbd "C-c r") 'srefactor-refactor-at-point)
```

6.3 Global

```
(global-set-key (kbd "C-d") 'crux-duplicate-current-line-or-region)

(global-set-key (kbd "<delete>") 'delete-char)

(global-set-key (kbd "C-x c c") 'replace-regexp)
(global-set-key (kbd "C-x c C-c") 'replace-string)

(global-set-key (kbd "C-+") 'text-scale-increase)
(global-set-key (kbd "C--") 'text-scale-decrease)

(global-set-key (kbd "C-x O") (lambda ()
                                (interactive)
                                (other-window -1))) ;; back one
(bind-key "C-k" 'crux-kill-whole-line)

(bind-key "C-c c" 'org-capture)

(defcustom after-save-interactively-hook nil
  "Normal hook that is run after a buffer is saved interactively to its file.
See `run-hooks'."
  :group 'files
  :type 'hook)
```

```
(defun save-buffer-and-call-interactive-hooks (&optional arg)
  "Save the buffer and call hooks if called interactively.
ARG is passed to 'save-buffer'"
  (interactive "p")
  (save-buffer arg)
  (when (called-interactively-p 'all)  ;; run post-hooks only if called interactively
    (run-hooks 'after-save-interactively-hook)))

(global-set-key (kbd "C-x s") 'save-buffer-and-call-interactive-hooks)

(use-package expand-region
  :ensure t
  :bind ("C-=" . er/expand-region))
```