Содержание

1 DONE Мой конфиг Emacs в формате org-mode
EMACS

1.1 Описание

Этот конфиг использую на всех десктопах и телефонах. Постепенно он пополняется. Актуальную версию всегда можно найти в рабочем репозитории: `https://github.com/johnlepikhin/emacs-public/blob/master/settings.org`.

Из него динамически подгружаются файлы:

- ~/.emacs.d/local/user-info.org : настройки email, имени пользователя и т.д.
- ~/.emacs.d/local/org-agenda.org : личные настройки шаблонов agenda
- ~/.emacs.d/local/org-capture.org : личные шаблоны org-записей
- ~/.emacs.d/local/gnus-accounts.org : почтовые аккаунты
- ~/.emacs.d/local/gnus-templates.org : почтовые шаблоны

1.1.1 Как подключать

В ~/.emacs надо написать:

```
(package-initialize)

(defun my-load-org-config (subpath)
  (let ((ini-file (expand-file-name subpath user-emacs-directory)))
(condition-case errinfo (org-babel-load-file ini-file)
  (error (message "Cannot load settings for file %s: %s" ini-file errinfo)))))

(my-load-org-config "public/settings.org")
```

Здесь public/settings.org будет транслировано в ~/.emacs.d/public/settings.org, согласно значению user-emacs-directory.

1.2 Базовая конфигурация

Все переменные здесь лексически локальны:

```
(setq-local lexical-binding t)
```

Собрать информацию по use-package. После загрузки можно позвать функци use-package-report и посмотреть.

```
(setq use-package-compute-statistics t)
```

1.2.1 Внешний вид

Некоторые оптимизации:

```
(setq auto-revert-interval 1            ; Refresh buffers fast
      custom-file (make-temp-file "")   ; Discard customization's
      default-input-method "TeX"        ; Use TeX when toggling input method
      echo-keystrokes 0.1               ; Show keystrokes asap
      inhibit-startup-message t         ; No splash screen please
      initial-scratch-message nil       ; Clean scratch buffer
      sentence-end-double-space nil)    ; No double space
```

Выключить кнопку Insert (включение overwrite-mode):

```
(define-key global-map [(insert)] nil)
```

Выключить тулбар:

```
(tool-bar-mode -1)
```

Ширина заполнения по умолчанию большая, сейчас широкие мониторы:

```
(setq-default fill-column 140)
```

Не надо мне сообщать, что у строки есть продолжение. Просто показывай продолжение на следующей строке:

```
(setq-default truncate-lines t)
```

1.2.2 Настройка темы

```
(load-theme 'leuven t)
```

1.2.3 Менеджер пакетов и зависимостей

Подключить менеджер пакетов:

```
(require 'package)

(add-to-list 'package-archives '("melpa" . "http://melpa.org/packages/"))
(add-to-list 'package-archives '("melpa-stable" . "https://stable.melpa.org/packages/")
(add-to-list 'package-archives '("org" . "http://orgmode.org/elpa/") t)
```

Прибить гвоздями указанные пакеты к репозиториям:

```
(add-to-list 'package-pinned-packages '(cider . "melpa-stable") t)
```

Поставить перечисленные пакеты, если не установлены:

```
(dolist (package
         '(use-package))
  (if (not (package-installed-p package))
    (progn
      (message "Installing package %s" package)
    (package-refresh-contents)
      (package-install package))))
```

Настраиваем use-package:

```
(require 'use-package)
(setq use-package-always-ensure t)
```

Больше дебага для use-package:

```
(setq use-package-verbose t)
```

Я хочу указывать зависимости к системным пакетам при использовании use-package:

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

Продублируем определение, подгружающее функции из ~/.emacs:

```
(defun my-load-org-config (subpath)
  (let ((ini-file (expand-file-name subpath user-emacs-directory)))
    (condition-case errinfo
        (progn
          (org-babel-load-file ini-file)
          (message "Loaded config: %s" subpath))
      (error (message "Cannot load settings for file %s: %s" ini-file errinfo)))))
```

1.2.4 Информация о пользователе

```
(my-load-org-config "local/user-info.org")
```

1.3 Настройки ввода

У меня переключение раскладки внутри Emacs интегрировано с XMonad.
Нижеследующие функции служат этой интеграции.

```
(defun my-update-cursor ()
  (set-cursor-color
   (if (string= current-input-method "russian-computer") "red" "black")))

(add-hook 'buffer-list-update-hook 'my-update-cursor)

(defun my-update-isearch-input-method ()
  (if isearch-mode
      (progn
        (setq isearch-input-method-function input-method-function
              isearch-input-method-local-p t)
        (isearch-update))))

(defun my-update-input-method (is-ru)
  (if is-ru
      (set-input-method 'russian-computer)
    (inactivate-input-method))
  (my-update-isearch-input-method)
  (my-update-cursor))

(defun my-select-input-eng ()
  (interactive)
  (my-update-input-method nil))
```

```
(defun my-select-input-rus ()
  (interactive)
  (my-update-input-method t))

(global-set-key (kbd "<M-f11>") 'my-select-input-eng)
(global-set-key (kbd "<M-f12>") 'my-select-input-rus)
(define-key isearch-mode-map (kbd "<M-f11>") 'my-select-input-eng)
(define-key isearch-mode-map (kbd "<M-f12>") 'my-select-input-rus)
```

Странный способ добавить ещё одну раскладку:

```
(set-input-method 'russian-computer)
(toggle-input-method)
```

Создаем личную карту аккордов:

```
(defvar my-bindings-map (make-keymap)
  "Мои личные горячие клавиги.")

(define-minor-mode my-bindings-mode
  "Режим для подключения моих кнопок."
  t nil my-bindings-map)

(define-key my-bindings-map (kbd "C-c <up>")    'windmove-up)
(define-key my-bindings-map (kbd "C-c <down>")  'windmove-down)
(define-key my-bindings-map (kbd "C-c <left>")  'windmove-left)
(define-key my-bindings-map (kbd "C-c <right>") 'windmove-right)
(define-key my-bindings-map [C-return] (lambda () (interactive) (point-to-register 'r)))
(define-key my-bindings-map [M-return] (lambda () (interactive) (jump-to-register 'r)))
```

Зачем мне замораживать фреймы?

```
(global-unset-key (kbd "C-z"))
```

1.3.1 Expand region

По нажатию `C-=` выбранный регион семантически/синтаксически расширяется:

```
(use-package
  expand-region
  :bind (:map my-bindings-map
              ("C-=" . er/expand-region)))
```

1.4 Внешний вид

Золотое сечение для расположения окон:

```
(use-package zoom
  :commands zoom-mode
  :config
  (setq zoom-ignored-major-modes '(gnus-summary-mode))
  (setq zoom-size '(0.618 . 0.618)))
(zoom-mode t)
```

Удобное хождение по окнам:

```
(use-package ace-window
  :defer t
  :bind (:map my-bindings-map
              ("M-o" . ace-window)))
```

Подсказка по вводимым кнопкам. Отображать с задержкой в 1 секунду:

```
(use-package
  which-key
  :config
  (setq which-key-idle-delay 1)
  (which-key-mode))
```

Показывать в минибуфере позицию в строке:

```
(column-number-mode)
```

1.4.1 Imenu

```
(use-package
  imenu
  :bind (:map my-bindings-map
              ("M-'" . imenu-list-smart-toggle)))
```

1.4.2 Выключить menu-bar, показывать по F10

```
(menu-bar-mode -1)

(defun my-toggle-menu-bar ()
  (interactive)
  (toggle-menu-bar-mode-from-frame 'toggle))

(define-key my-bindings-map (kbd "<f10>") 'my-toggle-menu-bar)
```

1.5 Поведение

Бэкапы не нужны:

```
(setq backup-inhibited t)
```

Шаг отступа по умолчанию 4 символа:

```
(setq-default tab-width 4)
(setq-default standart-indent 4)
```

Я люблю ограничивать область видимости до выбранного региона, не надо меня ворнить:

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

История открытых файлов:

```
(use-package
  recentf
  :config
  (setq recentf-max-saved-items 100)
  (setq recentf-max-menu-items 25)
  (recentf-mode 1))
```

Помнить позиции в файлах:

```
(save-place-mode 1)
```

Поминить историю команд минибуфера:

```
(savehist-mode 1)
```

Браузер по умолчанию — chromium:

```
(setq browse-url-browser-function 'browse-url-chromium)
```

Использовать ivy в качестве движка автодополнения имен буферов, файлов и т.д.

[2019-09-23 Mon] закомментил т.к. тормозит на больших файлах.

Ищем swiper'ом:

[2019-09-23 Mon] закомментил т.к. тормозит на больших файлах.

1.5.1 Правильный список буферов

[2019-09-23 Mon] закомментил т.к. тормозит на больших файлах.

1.5.2 Настройки isearch

У меня плохо прижились стандартные кнопки, слегка переназначил:

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

1.6 Редактирование

Многострочный режим комментариев по умолчанию

```
(setq comment-style 'multi-line)
```

У нас не принято разделять предложения двойным пробелом:

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

Мы презираем табулялцию в отступах:

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

1.6.1 Yasnippet: разворачивание сниппетов

```
(use-package yasnippet
  :bind
  (:map my-bindings-map
        ("C-<tab>" . yas-expand))
  ;; я не люблю, когда по tab-у мне пытаются что-то развернуть
  (:map yas-minor-mode-map
        ("<tab>" . nil)
        ("TAB" . nil))
  :hook ((cperl-mode org-mode) . yas-minor-mode)
  :commands (yas-minor-mode)
  :after (yasnippet-classic-snippets)
  :config
  ;; подключить мой публичный репозиторий сниппетов
  (add-to-list 'yas-snippet-dirs (expand-file-name "~/.emacs.d/public/yasnippets"))
  ;; теперь надо всё перечитать
  (yas-reload-all))

(use-package yasnippet-classic-snippets)
```

1.6.2 Удаление без пополнения kill-ring

Не люблю, когда удаляемые куски текста автоматом копируются в буфер обмена. Кусочки кода откуда-то спёр.

```
(defun my-delete-word (arg)
  "Delete characters forward until encountering the end of a word.
With argument, do this that many times.
This command does not push text to `kill-ring'."
  (interactive "p")
  (delete-region
   (point)
   (progn
     (forward-word arg)
     (point))))

(defun my-backward-delete-word (arg)
  "Delete characters backward until encountering the beginning of a word.
With argument, do this that many times.
This command does not push text to `kill-ring'."
```

```
  (interactive "p")
  (my-delete-word (- arg)))

(define-key my-bindings-map (kbd "C-S-k") 'my-delete-line-backward)
(define-key my-bindings-map (kbd "M-d") 'my-delete-word)
(define-key my-bindings-map (kbd "<M-backspace>") 'my-backward-delete-word)
(define-key my-bindings-map (kbd "<C-backspace>") 'my-backward-delete-word)
```

1.6.3 Показывать отступы

Отключить режим для файлов больше 100K:

```
(defun my-disable-indent-guide-mode-for-big-buffer ()
  (when (> (buffer-size) 100000)
    (progn
      (message "Buffer is too big, disable guided indent mode")
      (indent-guide-mode -1))))

(use-package
  indent-guide
  :hook ((prog-mode . indent-guide-mode)
         (find-file . my-disable-indent-guide-mode-for-big-buffer))
  :config
  (setq indent-guide-char "|")
  (set-face-foreground 'indent-guide-face "darkgray"))
```

1.6.4 Умные скобочки

```
(use-package
  smartparens
  :config
  ;; Есть баг с electric-parens-mode с cperl, заплатка из https://github.com/syl20bnr/s
  (with-eval-after-load 'cperl-mode
    (add-hook 'smartparens-enabled-hook  (lambda () (define-key cperl-mode-map "{" nil)
    (add-hook 'smartparens-disabled-hook  (lambda () (define-key cperl-mode-map "{" 'cp
  ;; Включаем глобально
  (smartparens-global-mode 1))
```

Подсвечивать парную скобку, переходы по ним:

```
(show-paren-mode 1)
```

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

(defun match-paren (arg)
  "Go to the matching paren."
  (interactive "p")
  (cond ((looking-at "\\s\(") (forward-list 1) (backward-char 1))
        ((looking-at "\\s\)") (forward-char 1) (backward-list 1))
        (t (self-insert-command (or arg 1)))))

(define-key my-bindings-map (kbd "C-‘") 'match-paren)
```

1.6.5 Дерево отмены

```
(use-package
  undo-tree
  :bind (:map my-bindings-map
              ("C-x u" . undo-tree-visualize))
  :config (global-undo-tree-mode 1))
```

1.6.6 Автодополнение

```
 (use-package
   company
   :hook (prog-mode . company-mode)
   :config
   ;; сначала ищем в gtags, а если не нашли - смотрим в abbrev
   (add-to-list 'company-backends '(company-gtags :with company-dabbrev))
   ;; по кнопке TAB происходит и дополнение, и выбор варианта
;;   (company-tng-configure-default)
   (setq company-idle-delay 0.5
         company-echo-delay 0
         company-dabbrev-downcase nil
         company-show-numbers t
         company-minimum-prefix-length 2
         company-selection-wrap-around t
         company-transformers '(company-sort-by-occurrence
                                company-sort-by-backend-importance)))
```

1. Автодополнение с неточным совпадением

   К сожалению, поддерживает не так много бэкендов company. Я использую только для lisp.

```
(use-package company-flx
  :demand t
  :init
  (company-flx-mode +1))
```

2. Документация в company

```
(use-package company-quickhelp
  :ensure t
  :hook (global-company-mode . company-quickhelp-mode)
  :commands (company-quickhelp-mode)
  :defer t
  :init (add-hook 'global-company-mode-hook #'company-quickhelp-mode))
```

1.6.7 Tramp

```
(use-package
  tramp
  :config (setq tramp-use-ssh-controlmaster-options nil))
```

1.6.8 drug-staff

Таскаем выделенное или строчку по буферу

```
(use-package drag-stuff
  :bind (:map my-bindings-map
              ("M-p" . drag-stuff-up)
              ("M-n" . drag-stuff-down)))
```

1.7 Разработка

1.7.1 Управление проектами

Подключаем projectile:

```
(use-package
  projectile
  :bind (:map projectile-mode-map
              ("C-c p" . 'projectile-command-map))
  :config (projectile-mode +1))
```

Подключаем helm к projectile

```
(use-package
  helm-projectile
  :after (projectile helm)
  :config (helm-projectile-on))
```

1.7.2 Сборки

```
(use-package multi-compile
  :commands (multi-compile multi-compile-run)
  :init
  (setq multi-compile-alist
        '((go-mode . (
                      ("go-build" "go build -v"
                       (locate-dominating-file buffer-file-name ".git"))
                      ("go-build-and-run" "go build -v && echo 'build finish' && eval
                       (multi-compile-locate-file-dir ".git"))))
          )))
```

1.7.3 Magit

```
(use-package
  magit
  :bind (:map my-bindings-map
              ("C-x g" . magit-status)))
```

1.7.4 Индекс по тэгам

```
(use-package ggtags
  :hook (cperl-mode . ggtags-mode)
  :config
  (setq ggtags-sort-by-nearness nil
        ggtags-navigation-mode-lighter nil
        ggtags-mode-line-project-name nil
        ggtags-oversize-limit (* 30 1024 1024)))
```

1.7.5 Flycheck

```
(use-package
  flycheck
  :config
  (setq flycheck-idle-change-delay 5)
```

```
;; Багфиксы, согласно проблеме https://github.com/flycheck/flycheck/issues/1278:
;; в :preface нельзя определять т.к. :preface обрабатывается до загрузки модуля
(advice-add
 'flycheck-process-send-buffer
 :override
 (lambda (process)
   (condition-case err
       (save-restriction
         (widen)
         (if flycheck-chunked-process-input
             (flycheck--process-send-buffer-contents-chunked process)
           (process-send-region process (point-min) (point-max)))
         (process-send-eof process))
     (error
      (let* ((checker (process-get process 'flycheck-checker))
             (type (flycheck-checker-get checker 'standard-input)))
        (when (or (not (eq type 'ignore-error)) (process-live-p process))
          (signal (car err) (cdr err))))))))
(flycheck-define-checker perl
  "A Perl syntax checker using the Perl interpreter.

  See URL `https://www.perl.org'."
  :command ("perl" "-w" "-c"
            (option-list "-I" flycheck-perl-include-path)
            (option-list "-M" flycheck-perl-module-list concat))
  :standard-input ignore-error
  :error-patterns
  ((error line-start (minimal-match (message))
          " at - line " line
          (or "." (and ", " (zero-or-more not-newline))) line-end))
  :modes (perl-mode cperl-mode)
  :next-checkers (perl-perlcritic))
(setq flycheck-global-modes '(not org-mode)
      flycheck-display-errors-function nil
      flycheck-display-errors-delay 0
      ;; Иногда приходится разгребать чуланы с граблями, надо видеть тысячи ошибок в
      flycheck-checker-error-threshold 10000)
(global-flycheck-mode))
```

Показывать ошибки прямо на месте, под строкой:

```
(use-package
  flycheck-inline
  :after (flycheck)
  :hook (flycheck-mode . flycheck-inline-mode))
```

1.7.6 Perl

Таблица символьного просмотра:

```
(defun my-cperl-init-prettify-symbols ()
  (setq prettify-symbols-alist
        '(("<=" . ?)
          ("&&" . ?)
          ("||" . ?)
          ("!=" . ?)
          ("for" . ?)
          ("foreach" . ?)
          ("exists" . ?)
          ("undef" . ?)
          ("sub" . ?)
          ("return" . ?)
          ("//" . ?)
          ("my" . ?)
          ("delete" . ?)
          ("defined" . ?)
          ("!" . ?¬)
          ("not" . ?¬)
          ("join" . ?)
          ("grep" . ?)
          ("map" . ?)
          ("sort" . ?)
          (".." . ?)
          ("next" . ?)
          ("last" . ?)
          ("while" . ?)
          ("if" . ?)
          ("elsif" . ?)
          ("else" . ?)
          ("int" . ?)
          ("keys" . ?)
```

```
          ("ne" . ?)
          ("eq" . ?)
          ("->" . ?→)
          ("=>" . ?)
          ("=~" .?)
          ("!~" . ?)
          ("$self" . ?)))
  (prettify-symbols-mode))

(use-package
  cperl-mode
  :ensure-system-package perltidy
  :after (flycheck tramp)
  :bind (:map cperl-mode-map ())
  :mode ("\\.\\([pP][Llm]\\|al\\)\\'" . cperl-mode)
  :interpreter (("perl"     . cperl-mode)
                ("perl5"    . cperl-mode)
                ("miniperl" . cperl-mode))
  :config
  ;; perl := cperl
  (defalias 'perl-mode 'cperl-mode)
  ;; C-h P будет показывать perldoc на слово под курсором
  (define-key 'help-command "P" 'cperl-perldoc-at-point)
  ;; я люблю smartparens. Извини, cperl-mode.
  (setq cperl-electric-parens nil)
  ;; Символьное отображение синтаксиса
  (add-hook 'cperl-mode-hook 'my-cperl-init-prettify-symbols)
  ;; Отступ равен 4, не выпендриваться!
  (setq cperl-indent-level 4
        cperl-close-paren-offset -4
        cperl-continued-statement-offset 4
        cperl-indent-parens-as-block t
        cperl-tab-always-indent t)
  ;; Красные хэши меня всегда раздражали
  (face-spec-set 'cperl-hash-face '((t :foreground "darkblue"))))
```

1. Автодополнение для Perl через PerlySense

```
(use-package
  company-plsense
  :hook (cperl-mode . company-mode))
```

1.7.7 Ocaml

```
(use-package tuareg)

(use-package merlin
  :init
  ;; merlin сам их не объявил в зависимости, небольшой костыль
  (use-package iedit)
  (use-package auto-complete)
  :after (tuareg iedit auto-complete)
  :hook ((tuareg-mode . merlin-mode)
         (tuareg-mode . company-mode))
  :config
  (require 'opam-user-setup "~/.emacs.d/opam-user-setup.el"))

(use-package flycheck-ocaml
  :after (flycheck merlin)
  :config
  ;; ошибки от мерлина нам не нужны, ведь у нас есть flycheck
  (setq merlin-error-after-save nil)
  (flycheck-ocaml-setup))
```

1.7.8 Coq

```
(defun my-coq-mode-setup ()
  (company-coq-mode)
  (setq-local company-minimum-prefix-length 1)
  (setq coq-double-hit-enable t)
  (golden-ratio-mode -1)
  (auto-complete-mode -1))

(use-package proof-general
  :mode ("\\.v$" . coq-mode)
  :hook (coq-mode . my-coq-mode-setup))
```

1.7.9 Golang

```
(defun my-go-mode-setup ()
  (yas-minor-mode)
  (flycheck-mode)
  (add-hook 'before-save-hook 'gofmt-before-save)
```

```
  (add-hook 'completion-at-point-functions 'go-complete-at-point)
  (go-guru-hl-identifier-mode)
  (go-eldoc-setup))

(use-package go-mode
  :ensure t
  :mode ("\\.go\\'" . go-mode)
  :config
  (add-hook 'go-mode-hook 'my-go-mode-setup))
```

Документация

```
(use-package go-eldoc
  :ensure t
  :commands (go-eldoc-setup))
```

Автодополнение

```
(use-package company-go
  :defer t
  :init
  (with-eval-after-load 'company
    (add-to-list 'company-backends 'company-go)))
```

Рефакторинг

```
(use-package go-rename
  :commands (go-rename))
```

Подсветка идентификаторов

```
(use-package go-guru
  :commands (go-guru-hl-identifier-mode))
```

1.7.10 Rust

```
(use-package rust-mode
  :after (flycheck tramp racer compile)
  :mode "\\.rs\\'"
  :bind (:map rust-mode-map
              ("C-c i b" . rust-format-buffer))
  :config
```

```
(setq rust-format-on-save t)
(defun my-rust-buffer-setup ()
  (set (make-local-variable 'compile-command)
       (if (locate-dominating-file (buffer-file-name) "Cargo.toml")
           "cargo run"
         (format "rustc %s && %s" (buffer-file-name)
                 (file-name-sans-extension (buffer-file-name))))))
(add-hook 'rust-mode-hook 'my-rust-buffer-setup))
```

1. Автодополнение

```
(use-package racer
  :hook (rust-mode . racer-activate)
  :config
  (setq racer-rust-src-path "~/.local/share/rust_src/src"))
```

2. Проверка синтаксиса на лету

```
(use-package flycheck-rust
  :after (flycheck)
  :hook (flycheck-mode . flycheck-rust-setup))
```

1.7.11 Puppet

```
(use-package puppet-mode
  :config
  (setq puppet-indent-level 2)
  (setq puppet-include-indent 2))
```

1.7.12 HTML

Подключаем web-mode (HTML, JS, CUSS в одном):

```
(use-package web-mode
  :mode ("\\.html$" . web-mode)
  :init
  (setq web-mode-markup-indent-offset 4)
  (setq web-mode-code-indent-offset 4)
  (setq web-mode-css-indent-offset 4)
  (setq js-indent-level 4)
  (setq web-mode-enable-auto-pairing t)
  (setq web-mode-enable-auto-expanding t)
  (setq web-mode-enable-css-colorization t))
```

1.7.13 Javascript / ECMA / React

```
(defun my-tide-buffer-setup ()
  (add-hook 'before-save-hook 'tide-format-before-save nil 'local))

(use-package tide
  :after (company flycheck)
  :hook (tide-mode . my-tide-buffer-setup)
  :config
  (add-to-list 'company-backends 'company-tide)
  (define-key tide-mode-map (kbd "C-.") 'tide-jump-to-definition)
  (define-key tide-mode-map (kbd "C-,") 'tide-jump-back)
  (flycheck-add-next-checker 'javascript-eslint 'javascript-tide 'append)
  (flycheck-add-next-checker 'javascript-eslint 'jsx-tide 'append)
  (tide-hl-identifier-mode +1))

(defun my-jsx-setup ()
  (flycheck-mode +1))

(use-package rjsx-mode
  :mode ("\\.js" . rjsx-mode)
  :mode ("\\.jsx" . rjsx-mode)
  :hook (rjsx-mode . tide-setup)
  :hook (rjsx-mode . my-jsx-setup)
  :config
  (setq js-indent-level 2)
  (flycheck-add-mode 'javascript-eslint 'rjsx-mode))
```

1.7.14 SQL

```
(use-package sql-indent
  :hook (sql-mode . sqlind-minor-mode)
  :commands sqlind-minor-mode)
```

1.7.15 Форматирование

Perltidy.el, взято отсюда: `https://www.emacswiki.org/emacs/perltidy.el`

```
(defcustom perltidy-program "perltidy"
  "*Program name of perltidy"
```

```
  :type 'string
  :group 'perltidy)

(defcustom perltidy-program-params
  '(;; I/O control
    "--standard-output"
    "--standard-error-output"
    "--force-read-binary"
    "--quiet"

    ;; FORMATTING OPTIONS
    "--no-check-syntax"
    )
  "*perltidy run options"
  :type 'list
  :group 'perltidy)

(defcustom perltidy-rcregex "\\.perltidyrc"
  "perltidyrc file regex"
  :type 'string
  :group 'perltidy)

(defun perltidy-buffer ()
  "Call perltidy for whole buffer."
  (interactive)
  (perltidy-region (point-min) (point-max)))
;;;###autoload
(defun perltidy-region (beg end)
  "Tidy perl code in the region."
  (interactive "r")
  (or (get 'perltidy-program 'has-perltidy)
      (if (executable-find perltidy-program)
          (put 'perltidy-program 'has-perltidy t)
        (error "Seem perltidy is not installed")))
    (let ((old-perltidy-env (getenv "PERLTIDY"))
          (remote? (tramp-tramp-file-p buffer-file-name))
          (perltidyrc (perltidy-find-perltidyrc buffer-file-truename))
          (pertidyrc-remote (expand-file-name "perltidyrc-remote" temporary-file-direct
          (perltidy-run-list perltidy-program-params)
          )
```

```
      (if (and (bound-and-true-p remote?)
               perltidyrc)
          (progn
            (require 'tramp-sh)
            (tramp-sh-handle-copy-file perltidyrc pertidyrc-remote t)
            (setq perltidyrc pertidyrc-remote)
            (setq perltidy-run-list
                  (append perltidy-run-list
                          (list (concat "-pro=" pertidyrc-remote))))))

      (apply #'call-process-region
             (append (list beg end perltidy-program
                           t
                           t
                           nil
                           )
                     perltidy-run-list)))
    t)

(defun perltidy-subroutine ()
  "Call perltidy for subroutine at point."
  (interactive)

  (save-excursion
    (let ((current-point (point))
          b e)
      (setq b (progn (beginning-of-defun) (point)))
      (when (and
             (looking-at "\\s-*sub\\s-+")
             (< b current-point)
             (> (save-excursion
                  (setq e (progn (end-of-defun) (point))))
                current-point))
        (perltidy-region b e)))))

(defun perltidy-find-perltidyrc (&optional dir rcregex)
  (unless dir (setq dir (buffer-file-name)))
  (unless rcregex (setq rcregex perltidy-rcregex))
  (setq dir (file-name-directory dir))
```

```
  (let (rcfile)
    (catch 'my-tag
      (locate-dominating-file
       dir
       (lambda (parent)
         (let ((rc (car (ignore-errors (directory-files parent t rcregex))))
               (pparent (file-name-directory (directory-file-name parent))))
           (setq rcfile rc)
           (cond ((equal parent
                         pparent)
                  (if (= (length rc) 0)
                      (throw 'my-tag rc)
                    (throw 'my-tag nil)))

                 ((and (= (length rc) 0)
                       (file-exists-p    (expand-file-name "lib" pparent))
                       (file-directory-p (expand-file-name "lib" pparent)))
                  (setq rcfile (car (ignore-errors (directory-files pparent t rcregex))
                  (throw 'my-tag rcfile))
                 (t rc))))))
    rcfile))
```

Perltidy для буфера, текущей функции, строчки и региона:

```
(defun perl-mode-perltidy ()
  "Perltidy buffer or region if this is perl file."
  (interactive)
  (let ((saved-line (line-number-at-pos)))
    (save-excursion
      (when (eq major-mode 'cperl-mode)
        (if (use-region-p)
            (perltidy-region (region-beginning) (region-end))
          (perltidy-buffer))))
    (forward-line saved-line)))

(defun my-perltidy-subroutine ()
  "Perltidy current subroutine keeping current position in the buffer as close as possi
  (interactive)
  (let ((saved-line (line-number-at-pos)))
```

```
    (perltidy-subroutine)
    (goto-line saved-line)))

(defun my-perl-tab-indent ()
  (interactive)
  (if (use-region-p)
      (perltidy-region (region-beginning) (region-end))
    (cperl-indent-command)))

(use-package
  cperl-mode
  :after (tramp)
  :bind (:map cperl-mode-map
              ("C-c i d" . my-perltidy-subroutine)
              ("C-c i b" . perl-mode-perltidy)
              ("TAB" . my-perl-tab-indent)))
```

Вставка JSON из буфера как Perl-структуры. Не спрашивайте, нужно для работы.

```
(defun my-perl-insert-json ()
  (interactive)
  (shell-command-on-region (point) (point) "xclip -o | json_to_perl.pl" t))

(use-package
  cperl-mode
  :bind (:map cperl-mode-map
              ("C-c j" . my-perl-insert-json)))
```

1.7.16 Подсветка TODO/FIXME

```
(use-package fic-mode
  :hook cperl-mode emacs-lisp-mode rust-mode rjsx-mode)
```

1.7.17 Цветной compile buffer

```
(defun my-colorize-compilation-buffer ()
  (toggle-read-only)
  (ansi-color-apply-on-region compilation-filter-start (point))
  (toggle-read-only))
```

```
(use-package ansi-color
  :hook (compilation-filter . my-colorize-compilation-buffer))
```

1.8 Org-mode

1.8.1 Основные настройки

```
(defun my-org-mode-basic-config ()
  ;; По умолчанию таски длятся 1 час
  (setq org-agenda-default-appointment-duration 60)
  ;; SRC-блоки должны выглядеть максимально похоже на исходные режимы редактирования дл
  (setq org-src-fontify-natively t)
  ;; Кнопка tab в SRC-блоках имеет то же поведение, что и в исходных режимах языков
  (setq org-src-tab-acts-natively t)
  ;; Не надо никуда смещать SRC-блоки
  (setq org-edit-src-content-indentation 0)
  ;; Ругаться, если пытаемся редактировать невидимый (напр., схлопнутый) текст
  (setq org-catch-invisible-edits 'error)
  ;; Хочу видеть дату+время, когда пункт был закрыт (CLOSED)
  (setq org-log-done 'time)
  ;; Задаем виды статусов для задач
  (setq org-todo-keywords
      '((sequence "TODO(t)" "WAIT(w)" "VERIFY(v)" "DELEGATED(D@)" "|" "DONE(d)" "CANCEL
  ;; Спрятать лог изменения статусов в LOGGER
  (setq org-log-into-drawer t)
  ;; Разшить refile в мои org-файлы, в поддеревья до глубины 2
  (setq org-refile-targets '((org-agenda-files :maxlevel . 2)))
  ;; При refile показывать также имя файла
  (setq org-refile-use-outline-path 'file)
  ;; Люблю выделять по shift-стрелочки, даже в org-mode
  (setq org-support-shift-select t)

  ;; Угадывать mode SRC-блоков по названию режимов
  (add-to-list 'org-src-lang-modes '("conf" . conf))
  (add-to-list 'org-src-lang-modes '("ini" . conf))
  (add-to-list 'org-src-lang-modes '("vim" . vimrc))

  ;; Подключить эти модули
  (add-to-list 'org-modules 'org-id)
  (add-to-list 'org-modules 'org-mouse)
```

```
  (add-to-list 'org-modules 'org-attach-screenshot)

  ;; Разукрашиваем статусы
  (setq org-todo-keyword-faces
        '(("WAIT" . (:foreground "#ff8040" :weight bold))
          ("VERIFY" . (:foreground "#afaf00" :weight bold))
          ("CANCELED" . (:foreground "#006000" :weight bold))))

  ;; Подкрасить слова TODO красным в org-файлах
  (face-spec-set 'org-todo '((t :foreground "red")))

  ;; Мне не нравятся большие заголовки 1-го уровня в теме leuven
  (face-spec-set 'org-level-1 '((t :height 1.1)))

  ;; Мне не нравятся большие заголовки в #TITLE
  (face-spec-set 'org-document-title '((t :height 1.2)))
)
```

1.8.2 Подключение

```
(require 'org-protocol)

(use-package
  org
  :ensure nil
  :hook ((org-mode . turn-on-flyspell)
         ;; автоматом считывать изменения с диска
         (org-mode . turn-on-auto-revert-mode)
         ;; автосохранение для org-буферов
         (auto-save . org-save-all-org-buffers)
         ;; автоперенос строк по умолчанию
         (org-mode . auto-fill-mode))
  :bind (:map my-bindings-map
              ("C-c l" . org-store-link))
  :config
  (my-org-mode-basic-config))
```

1.9 Org agenda

1.9.1 Дополнительные функции

Перечитать известные файлы с диска:

```
(setq my-org-last-reload (current-time))

(defun my-org-reload-from-disk (&optional event)
  (interactive)
  ;; релоадить не чаще раза в 3 секунды
  (if (time-less-p (time-add my-org-last-reload 3) (current-time))
      (progn
        (setq my-org-last-reload (current-time))
        (ignore-errors
          (org-agenda-redo-all)))))
```

Заполнить список файлов автоматом, согласно моим правилам. Это делается по крону, через run-with-timer.

```
(defvar my-org-root-path "~/org" "Path to root directory with org files")
(defvar my-org-files-regexp "[.]org$" "Regexp to match org files")

(defun my-org-fill-files-list (&optional EXHAUSTIVE)
  (setq org-agenda-files
        (seq-remove
         (lambda (file) (string-match "[.]#" file))
         (directory-files-recursively my-org-root-path my-org-files-regexp)))
  ;; после пересоздания списков файлов, неплохо бы перечитать их с диска
  (my-org-reload-from-disk))
  ;; (my-org-fill-inotify-handlers))
```

Обновить буфер агенды:

```
(defun my-org-agenda-redo ()
  (ignore-errors
    (with-current-buffer "*Org Agenda*"
      (org-agenda-maybe-redo))))
```

1.9.2 Базовые настройки agenda

```
(defun my-agenda-mode-setup ()
```

```
  (hl-line-mode))

(defun my-org-agenda-basic-config ()
  ;; Дни рожденния в BBDB брать из поля birthday
  (setq org-bbdb-anniversary-field 'birthday)
  ;; Не показывать DONE в агенде
  (setq org-agenda-skip-scheduled-if-done 't)

  ;; Настройки по умолчанию в теме leuven мне в этом месте не нравятся
  (face-spec-set 'org-agenda-structure '((t :height 1.17)))
  (face-spec-set 'org-agenda-date-today '((t :height 1.1)))
  (face-spec-set 'org-agenda-date '((t :height 1.1)))
  (face-spec-set 'org-agenda-date-weekend '((t :height 1.1))))
```

1.9.3 Отложенные таски

Иногда хочется спрятать таск из агенды на определенное время, чтобы глаза его мои не видели.

Вешаем в agenda-буфере на кнопку C-d функционал откладывания таска:

```
(defun my-agenda-delayed-tasks-setup ()
  (defun my-org-agenda-delay-task ()
    (interactive)
    (org-agenda-check-no-diary)
    (let* ((hdmarker (or (org-get-at-bol 'org-hd-marker)
                         (org-agenda-error)))
           (buffer (marker-buffer hdmarker))
           (pos (marker-position hdmarker))
           (inhibit-read-only t)
           newhead)
      (org-with-remote-undo buffer
        (with-current-buffer buffer
          (widen)
          (goto-char pos)
          (org-show-context 'agenda)
          (let ((delay (org-read-date 't 'nil 'nil "Отложить до" 'nil
                                      (format-time-string "%H:%M" (time-add (current-t
            (org-set-property "DELAYED_TILL" delay))))
      (org-agenda-redo-all)))
```

```
  (defun my-org-agenda-delay-task-setup-hook ()
    (local-set-key (kbd "\C-c d") 'my-org-agenda-delay-task))
  (add-hook 'org-agenda-mode-hook 'my-org-agenda-delay-task-setup-hook)
)
```

Функция, реализующая пропуск отложенных тасков в agenda-буфере:

```
(defun my-org-agenda-skip-delayed ()
  (let ((now (format-time-string "%Y-%m-%d %H:%M" (time-add (current-time) 120)))
        (delayed-till (org-read-date t nil (or (org-entry-get nil "DELAYED_TILL") "") n
        (subtree-end (save-excursion (org-end-of-subtree t))))
      (if (string> delayed-till now) subtree-end nil)))
```

1.9.4 Горячие кнопки

Конфигурация команд agenda:

```
(defun my-org-agenda-commands-config ()
  (setq org-agenda-custom-commands
      '(("d" . "Сегодня")
        ("dd" agenda "Сегодня, все записи"
         ((org-agenda-span 'day)
          (org-agenda-overriding-header "Сегодня, все записи")))
        ("da" agenda "Сегодня, без отложенных"
         ((org-agenda-span 'day)
          (org-agenda-skip-function 'my-org-agenda-skip-delayed)
          (org-agenda-overriding-header "Сегодня, только активные"))))))
```

1.9.5 Подключение

```
(use-package
    org-agenda
    :ensure nil
    :after (org org-element)
    :hook (org-agenda-mode . my-agenda-mode-setup)
    :bind (:map my-bindings-map
                ("C-c a" . org-agenda))
    :config
    (my-org-agenda-basic-config)
    (my-agenda-delayed-tasks-setup)
```

```
(my-org-agenda-commands-config)
(my-org-fill-files-list)
;; раз в 10 минут заново составлять список файлов, на случай появления новых
(run-with-timer 0 600 'my-org-fill-files-list)
;;
(run-with-idle-timer 120 120 'my-org-agenda-redo)
(my-load-org-config "local/org-agenda.org"))
```

1.9.6 Прочее

Автосохранение перед выходом:

```
(advice-add 'org-agenda-quit :before 'org-save-all-org-buffers)
```

1. Архивирование всех DONE тасков в текущем буфере

```
(defun my-org-archive-done-tasks ()
  "Archive all DONE tasks in current buffer"
  (interactive)
  (org-map-entries
   (lambda ()
     (org-archive-subtree)
     (setq org-map-continue-from (outline-previous-heading)))
   "/DONE" 'file))
```

2. Клонировать текущий heading со всеми подзаголовками, со сдвигом всех дат внути

```
(defun my-org-clone-to-date ()
  "Clone current subtree into specified file with all dates shifted to the same pe
  (interactive)
  (let* ((title (nth 4 (org-heading-components)))
         (orig-date (org-time-string-to-absolute (org-entry-get nil "SCHEDULED")))
         (dest-date (org-time-string-to-absolute
                     (org-read-date nil nil nil (format "Дата для '%s'" title))))
         (offset (format "+%id" (- dest-date orig-date))))
    (org-copy-subtree)
    (with-temp-buffer
      (org-mode)
      (org-paste-subtree)
```

```
          (org-clone-subtree-with-time-shift 1 offset)
          (org-forward-element)
          (org-refile))))
```

1.10 Org export

1.10.1 Confluence

```
(use-package
  org
  :defer t
  :config
  (require 'ox-confluence))
```

1.10.2 LaTeX

```
(use-package
  org
  :defer t
  :config
  ;; Какие \usepackage прописывать в LaTeX по умолчанию
  (setq org-latex-default-packages-alist
        '(("utf8" "inputenc" t ("pdflatex"))
          ("T2A" "fontenc" t ("pdflatex"))
          ("russian" "babel" t)
          ("" "cmap" t)
          ("" "graphicx" t)
          ("" "grffile" t)
          ("" "longtable" nil)
          ("" "wrapfig" nil)
          ("" "rotating" nil)
          ("normalem" "ulem" t)
          ("" "amsmath" t)
          ("" "textcomp" t)
          ("" "tabularx" t)
          ("" "amssymb" t)
          ("" "capt-of" nil)
          ("" "hyperref" nil)))
  ;; Файлы с этими расширениями считаются временными при экспорте и будут удалены
  (setq org-latex-logfiles-extensions
        '("aux" "bcf" "blg" "fdb_latexmk" "fls" "figlist" "idx" "log" "nav" "out" "ptc
```

1.10.3 Hugo

1. Базовая настройка

```
(use-package
  ox-hugo
  :after (ox org)
  :hook ((org-export-before-processing . my-org-hugo-add-printable-version)
         (org-export-before-processing . my-org-hugo-add-source-of-article))
  :config
  (setq org-hugo-external-file-extensions-allowed-for-copying
        '("jpg" "jpeg" "tiff" "png" "svg" "gif" "pdf" "odt" "doc" "ppt" "xls" "doc
  (remove-hook 'org-export-before-parsing-hook 'my-org-hugo-add-printable-version)
```

2. Фича: ссылка на версию для печати

```
(defun my-org-hugo-add-printable-version (backend)
  (if (eq backend 'hugo)
      (let ((generate-printable (org-entry-get nil "HUGO_GENERATE_PRINTABLE"))
            (file-org-name (buffer-file-name))
            (file-pdf-name (concat (file-name-sans-extension (buffer-file-name)) '
        (if (and generate-printable (string= generate-printable "t"))
            (if (file-newer-than-file-p file-org-name file-pdf-name)
                (progn
                  (org-latex-export-to-pdf)
                  (find-file file-org-name)
                  (if (not (org-entry-get nil "HUGO_GENERATE_PRINTABLE_ADDED"))
                      (progn
                        (save-excursion
                          (goto-char (point-max))
                          (insert (format "\n** Версия для печати\n\nДля удобства
                        (org-set-property "HUGO_GENERATE_PRINTABLE_ADDED" "t")
                        (save-buffer)))))))))

(use-package
  ox-hugo
  :after (ox org)
  :hook (org-export-before-processing . my-org-hugo-add-printable-version)
  :config
  (remove-hook 'org-export-before-parsing-hook 'my-org-hugo-add-printable-version)
```

3. Фича: ссылка на исходник статьи

```
(defun my-org-hugo-add-source-of-article (backend)
  (if (eq backend 'hugo)
      (let* ((generate-printable (org-entry-get nil "HUGO_ADD_ARTICLE_SOURCE"))
             (file-org-name (buffer-file-name))
             (file-org-shortname (file-name-nondirectory file-org-name)))
        (if (and generate-printable (string= generate-printable "t"))
            (progn
              (find-file file-org-name)
              (if (not (org-entry-get nil "HUGO_ADD_ARTICLE_SOURCE_ADDED"))
                  (progn
                    (save-excursion
                      (goto-char (point-max))
                      (insert (format "\n** Исходник статьи\n\nСсылка для скачиван
                                      file-org-shortname
                                      file-org-shortname)))
                    (org-set-property "HUGO_ADD_ARTICLE_SOURCE_ADDED" "t")
                    (save-buffer)))))))

(use-package
  ox-hugo
  :after (ox org)
  :hook (org-export-before-processing . my-org-hugo-add-printable-version))
```

4. Фича: расстановка переносов

   Использую внешний скриптик, основанный на нейросети, которую обучил переносам. `https://github.com/johnlepikhin/p5-AI-Hyphen`

```
(defun my-hyphenize-russian (input hyphen)
  (interactive)
  (with-temp-buffer
    (progn
      (insert input)
      (call-process-region (point-min) (point-max) "~/bin/hyphen/russian/russian-h
      (buffer-string))))

(defun my-hugo-improvements (text backend info)
  (when (org-export-derived-backend-p backend 'hugo)
```

```
    (my-hyphenize-russian text "&#173;")))

(use-package
  ox-hugo
  :after (ox org)
  :config
  (add-to-list 'org-export-filter-plain-text-functions
               'my-hugo-improvements))
```

5. Автоматический экспорт (batch mode)

```
(defvar my-org-blog-path "~/org/personal" "Root path where to find blog articles")

(defun my-org-hugo-export-file (f)
  (interactive)
  (save-excursion
    (message (concat "Processing file " f))
    (find-file f)
    (my-org-hugo-twits-prepare f)
    (save-buffer)
    (org-hugo-export-wim-to-md :all-subtrees nil nil t)
    (kill-buffer (current-buffer))))

(defun my-org-hugo-export-files-org-personal (&key newer-than)
  (interactive)
  (save-excursion
    (let ((newer-than (seconds-to-time (if (null newer-than) 0 newer-than))))
      (mapc 'my-org-hugo-export-file
            (seq-filter
             (lambda (file)
               (and
                (not (string-match "/[.]#" file))
                (time-less-p newer-than (nth 5 (file-attributes file)))))
             (directory-files-recursively my-org-blog-path "\\.s?org$"))))))
```

6. Экспорт embedded видео в Hugo. Не спрашивайте.

```
(defun org-image-update-overlay (file link &optional data-p refresh)
```

```
"Create image overlay for FILE associtated with org-element LINK.
      If DATA-P is non-nil FILE is not a file name but a string with the image c
      See also ‘create-image’.
      This function is almost a duplicate of a part of ‘org-display-inline-image
(when (or data-p (file-exists-p file))
  (let ((width
         ;; Apply ‘org-image-actual-width’ specifications.
         (cond
          ((not (image-type-available-p 'imagemagick)) nil)
          ((eq org-image-actual-width t) nil)
          ((listp org-image-actual-width)
           (or
            ;; First try to find a width among
            ;; attributes associated to the paragraph
            ;; containing link.
            (let ((paragraph
                   (let ((e link))
                     (while (and (setq e (org-element-property
                                          :parent e))
                                 (not (eq (org-element-type e)
                                          'paragraph))))
                     e)))
              (when paragraph
                (save-excursion
                  (goto-char (org-element-property :begin paragraph))
                  (when
                      (re-search-forward
                       "^[ \t]*#\\+attr_.*?: +.*?:width +\\(\\S-+\\)"
                       (org-element-property
                        :post-affiliated paragraph)
                       t)
                    (string-to-number (match-string 1))))))
            ;; Otherwise, fall-back to provided number.
            (car org-image-actual-width)))
          ((numberp org-image-actual-width)
           org-image-actual-width)))
        (old (get-char-property-and-overlay
              (org-element-property :begin link)
              'org-image-overlay)))
    (if (and (car-safe old) refresh)
```

```
          (image-refresh (overlay-get (cdr old) 'display))
        (let ((image (create-image file
                                   (and width 'imagemagick)
                                   data-p
                                   :width width)))
          (when image
            (let* ((link
                    ;; If inline image is the description
                    ;; of another link, be sure to
                    ;; consider the latter as the one to
                    ;; apply the overlay on.
                    (let ((parent
                           (org-element-property :parent link)))
                      (if (eq (org-element-type parent) 'link)
                          parent
                        link)))
                   (ov (make-overlay
                        (org-element-property :begin link)
                        (progn
                          (goto-char
                           (org-element-property :end link))
                          (skip-chars-backward " \t")
                          (point)))))
              (overlay-put ov 'display image)
              (overlay-put ov 'face 'default)
              (overlay-put ov 'org-image-overlay t)
              (overlay-put
               ov 'modification-hooks
               (list 'org-display-inline-remove-overlay))
              (push ov org-inline-image-overlays)))))))

;; youtube

(defvar yt-iframe-format
  (concat "<div class=\"yt-container\"><iframe src=\"https://www.youtube.com/embed
          " frameborder=\"0\""
          " allowfullscreen>%s</iframe></div>"))

(defvar yt-hugo-format "{{< youtube id=\"%s\" >}}")
```

```
(org-link-set-parameters
 "yt"
 :follow (lambda (handle)
           (browse-url
            (concat "https://www.youtube.com/embed/"
                    handle)))
 :export (lambda (path desc backend)
           (cl-case backend
             (md (format yt-hugo-format path))
             (html (format yt-iframe-format path (or desc "")))
             (latex (format "\href{%s}{%s}"
                            path (or desc "video"))))))

(defun org-yt-get-image (url)
  "Retrieve image from url."
  (let ((image-buf (url-retrieve-synchronously url)))
    (when image-buf
      (with-current-buffer image-buf
        (goto-char (point-min))
        (when (looking-at "HTTP/")
          (delete-region (point-min)
                         (progn (re-search-forward "\n[\n]+")
                                (point))))
        (setq image-data (buffer-substring-no-properties (point-min) (point-max)))

(defconst org-yt-video-id-regexp "[-_[:alnum:]]\\{10\\}[AEIMQUYcgkosw048]"
  "Regexp matching youtube video id's taken from 'https://webapps.stackexchange.co

(defun org-yt-display-inline-images (&optional include-linked refresh beg end)
  "Like 'org-display-inline-images' but for yt-links."
  (when (display-graphic-p)
    (org-with-wide-buffer
     (goto-char (or beg (point-min)))
     (let ((re (format "\\[\\[%s:\\(%s\\)\\]\\]" "yt" org-yt-video-id-regexp)))
       (while (re-search-forward re end t)
         (let ((video-id (match-string 1))
               (el (save-excursion (goto-char (match-beginning 1)) (org-element-co
               image-data)
           (when el
             (setq image-data
```

```
                    (or (let ((old (get-char-property-and-overlay
                                    (org-element-property :begin el)
                                    'org-image-overlay)))
                          (and old
                               (car-safe old)
                               (overlay-get (cdr old) 'display)))
                        (org-yt-get-image (format "http://img.youtube.com/vi/%s/0.j
                  (when image-data
                    (org-image-update-overlay image-data el t t)))))))))

    (use-package
      org
      :ensure nil
      :config (advice-add #'org-display-inline-images :after #'org-yt-display-inline-i
```

1.10.4 Mind map

```
(use-package org-mind-map
  :init
  (require 'ox-org)
  :ensure t
  :config
  ;; Никогда не хочу получать маленькие картинки. У меня большой мозг и он генерирует 6
  (setq org-mind-map-default-graph-attribs '()))
```

1.11 Org babel (исполняемые кусочки кода в org-файлах)

Org babel режимые, которые надо поддержать

```
(defun my-org-confirm-babel-evaluate (lang body)
  (not (or
        (string= lang "latex")
        (string= lang "dot")
        (string= lang "graphviz")
        (string= lang "gnuplot")
        (string= lang "plantuml"))))

(use-package
  org-babel
  :ensure nil
  :after
```

```
  (org ob-ruby ob-perl ob-shell ob-sql ob-plantuml ob-gnuplot ob-coq ob-python ob-ocaml
  :config
  ;; Не просить подтверждение для запуска SRC-блоков
  (setq org-confirm-babel-evaluate 'my-org-confirm-babel-evaluate)
  ;; Загрузить языки
  (org-babel-do-load-languages
   'org-babel-load-languages
   '((perl . t)
     (ruby . t)
     (shell . t)
     (latex . t)
     (org . t)
     (dot . t)
     (http . t)
     (sql . t)
     (coq . t)
     (ocaml . t)
     (plantuml . t)
     (gnuplot 't)
     (emacs-lisp . t))))
```

1.12 Org capture (шаблоны записей)

```
(defvar my-org-file-main-inbox "~/org/personal/general-TODO.org" "Главный файл для инбо
(defvar my-org-file-web-bookmarks "~/org/personal/web-bookmarks.org" "Путь до файла, гд

(use-package
  org-capture
  :ensure nil
  :after (org)
  :bind (:map my-bindings-map
              ("C-c c" . org-capture))
  :config
  ;; Главная кнопка добавления задачи
  (add-to-list
   'org-capture-templates
   '("g" "Общий TODO" entry (file my-org-file-main-inbox)
     "* TODO %?\nSCHEDULED: %t"))
  ;; Заметки из браузера тоже сохраняем в главный инбокс
  (add-to-list
```

```
 'org-capture-templates
 '("Pm" "(Protocol bookmark)" entry (file+headline my-org-file-main-inbox "Сохранено
   "* TODO Взято из веба: [[%:link][%:description]]\n  SCHEDULED: %^T\n\nДобавлено: %
;; Закладки браузера храним в указанном org-файле, в дереве годов/месяцев/дат
(add-to-list
 'org-capture-templates
 '("Pb" "(Protocol bookmark)" entry (file+olp+datetree my-org-file-web-bookmarks)
   "* Закладка %U : [[%:link][%:description]]\n%?\n"))
;; Подгрузить приватный локальный конфиг для конкретного хоста
(my-load-org-config "local/org-capture.org"))
```

1.12.1 Супер-шаблоны

Refile всего поддерева под курсором с подстановкой дат по заданному смещению. Для корневого поддерева должно быть задано SCHEDULED.

```
(defun my-org-clone-to-date ()
  "Clone current subtree into specified file with all dates shifted to the same period
  (interactive)
  (let* ((title (nth 4 (org-heading-components)))
         (orig-date (org-time-string-to-absolute (org-entry-get nil "SCHEDULED")))
         (dest-date (org-time-string-to-absolute
                     (org-read-date nil nil nil (format "Дата для '%s'" title))))
         (offset (format "+%id" (- dest-date orig-date))))
    (org-copy-subtree)
    (with-temp-buffer
      (org-mode)
      (org-paste-subtree)
      (org-clone-subtree-with-time-shift 1 offset)
      (org-forward-element)
      (org-refile))))
```

1. Как использую

   У меня есть шаблоны мероприятий, где описана вся последовательность действий, которые надо совершить при подготовке. В верхнем уровне указан SCHEDULED, указывающий, когда мероприятие должно состояться. При выполнении внутри корневого элемента этой функции, создается копия всей структуры в указанный файл со смещением дат относительно указанной. Например:

```
* Слёт Сплава [0%]                                                    :Сплав:
  DEADLINE: <2019-05-25 Сб -3d> SCHEDULED: <2019-05-25 Сб>
  :PROPERTIES:
  :COOKIE_DATA: todo recursive
  :END:

Ехать до станции Подосинки с Казанского вокзала
Электричка отходит в ..., стоит ... руб

Расписание автобусов от Подосинок: ...

Расписание автобусов от лагеря: ...

** TODO Список вещей
   SCHEDULED: <2019-05-22 Ср>

2 баллона газа
Продукты
Усы
Каска
...

** TODO Узнать расписание электричек, запланировать
   SCHEDULED: <2019-05-22 Ср>
** TODO Узнать расписание автобусов в [[https://vk.com/splavclub][группе]].
   SCHEDULED: <2019-05-22 Ср>
** TODO Карта на GPS
   SCHEDULED: <2019-05-21 Вт>

В архиве карт на GPS лежит splav.img

** TODO Снять деньги
   SCHEDULED: <2019-05-23 Чт>

** TODO Продумать раскладку по еде
   SCHEDULED: <2019-05-21 Вт>
```

Важно: фича реализована через функцию `org-clone-subtree-with-time-shift`, и она слишком умно интерпретирует repeated tasks. Поэтому если в

плане подготовки какой-то таск надо повторять (пример: перед поездкой в горы трижды в неделю запланировать бег), то таск надо раскопировать вручную, заранее расставив даты со смещением.

1.13 Прочие Org-фичи

1.13.1 Менеджер паролей

```
(use-package
  org-password-manager
  :after (org)
  :hook (org-mode . org-password-manager-key-bindings))
```

1.13.2 Таблицы

При сохранении org-файлов автоматом форматировать таблицы в нём:

```
(defun my-org-mode-on-save-buffer-setup ()
  (add-hook 'before-save-hook #'org-table-iterate-buffer-tables nil 'make-it-local))

(use-package
  org-table
  :ensure nil
  :after (org)
  :hook (org-mode . my-org-mode-on-save-buffer-setup))
```

1.13.3 Заархивировать все завершенные задачи в буфере

```
(defun my-org-archive-done-tasks ()
  "Archive all DONE tasks in current buffer"
  (interactive)
  (org-map-entries
   (lambda ()
     (org-archive-subtree)
     (setq org-map-continue-from (outline-previous-heading)))
   "/DONE" 'file))
```

1.14 Spell-checking

```
(use-package
  flyspell
 ;  :ensure-system-package (ispell aspell-en aspell-ru)
```

```
  :hook ((text-mode . turn-on-flyspell)
         (prog-mode . flyspell-prog-mode))
  :commands (flyspell-buffer turn-on-flyspell)
  :config
  ;; Переключаем язык проверки по переключению раскладки
  (defadvice my-select-input-eng (after ispell-american activate) (ispell-change-dictio
  (defadvice my-select-input-rus (after ispell-russian activate) (ispell-change-diction
```

Включаем ленивую проверку, когда ничего не происходит:

```
(use-package
  flyspell-lazy
  :config
  (setq flyspell-lazy-idle-seconds 1)
  ;; Проверять все буферы, включая временные типа *scratch* и буферов Gnus
  (setq flyspell-lazy-disallow-buffers nil)   (flyspell-lazy-mode 1))
```

1.15 Helm

```
(use-package helm-org-rifle)

(use-package
  helm
  :after (helm-org-rifle recentf)
  :bind (:map my-bindings-map
              ("M-s M-s" . my-helm-search-all))
  :preface
  (defvar my-helm-sources)
  ;; Функция ищет по всем подключенным источникам
  (defun my-helm-search-all ()
    (interactive)
    (let ((sources (append (helm-org-rifle-get-sources-for-open-buffers) my-helm-source
      (unless helm-source-buffers-list
        (setq helm-source-buffers-list
              (helm-make-source "Buffers" 'helm-source-buffers)))
      (helm :sources sources
            :buffer "*helm completions*")))
  :config
  (require 'helm-for-files)
  (require 'helm-elisp)
  (setq helm-split-window-in-side-p t
```

```
        helm-M-x-fuzzy-match t
        helm-buffers-fuzzy-matching t
        helm-recentf-fuzzy-match t
        projectile-completion-system 'helm)
  (setq helm-input-idle-delay 0.1)
  (setq my-helm-sources
        '(helm-source-buffers-list
          helm-source-recentf
          helm-source-info-pages
          helm-source-complex-command-history
          helm-source-etags-select
          helm-source-grep-ag
          helm-source-bookmarks))
  (helm-adaptive-mode 1))

(use-package helm-descbinds
  :after helm
  :config
  (helm-descbinds-mode))
```

1.16 Gnus

Функция для обрезания сообщения до сигнатуры:

```
(defun my-gnus-zap-to-signature ()
  (interactive)
  (let ((curpos (point))
        (endpos (re-search-forward "\n-- " nil t 1)))
    (if (not (eq endpos nil))
        (progn
          (kill-region curpos (match-beginning 0))
          (goto-char curpos)
          (open-line 1)))))
```

Проверить почту через offlineimap:

```
(defun my-gnus-update-news-external ()
  (interactive)
  (progn
    (message "Checking new news from external sources...")
    (shell-command "(offlineimap -o -1; notmuch new) >/dev/null 2>&1")
```

```
    (message "Checking new news from external sources... DONE")
    (gnus-group-get-new-news)))
```

Настройка буфера написания сообщения:

```
(defun my-message-mode-setup ()
  (when message-this-is-mail
    ;; Использовать логические строчки, не вставлять ньюлайны
    (turn-off-auto-fill)
    (setq fill-column 140
          visual-line-fringe-indicators '(left-curly-arrow right-curly-arrow))
    (visual-line-mode)
    (visual-fill-column-mode)))

(use-package
  gnus
  :commands (gnus)
  :hook ((message-mode . turn-on-flyspell)
         (message-mode . my-message-mode-setup)
         (message-send . ispell-message)
         (gnus-summary-mode . hl-line-mode)
         (gnus-group-mode . hl-line-mode)
         (gnus-message-setup . mml-secure-message-sign-pgpmime))
  :bind (:map message-mode-map
              ("M-z" . my-gnus-zap-to-signature)
              :map gnus-group-mode-map
              ("M-g" . my-gnus-update-news-external))
  :config
  ;; Надо определить переменные
  (require 'gnus-msg)

  ;; Выставить таймаут на коннект к NNTP
  (setq nntp-connection-timeout 10)

  ;; Периодический полл источников
  (gnus-demon-add-handler 'gnus-demon-scan-news 1200 300)
  (gnus-demon-init)

  ;; Какой браузер использовать
  (setq gnus-button-url 'browse-url-generic
```

```
      browse-url-generic-program "chromium"
      browse-url-browser-function gnus-button-url)

;; По умолчанию во всех группах делать копию отправляемого письма себе
(setq gnus-parameters
      '((".*"
         (gcc-self . t))))

;; Перед сохранением аттача имеет смысл нажать [A C], чтобы вытянуть сообщение целико
(setq nnimap-fetch-partial-articles "text/")

;; Обнуляем основной источник писем
(setq gnus-select-method '(nnnil ""))

;; Подключить дополнительные источники
(setq gnus-secondary-select-methods
      ;; Получение через IMAP с локалхоста
      '((nnimap "Mail"
                (nnimap-stream shell)
                (nnimap-shell-program "/usr/lib/dovecot/imap"))
        ;; UNIX mailbox
        (nnmbox "LocalMBOX")
        ;; Читаем RSS/Atom через ньюсгруппы
        (nntp "news.gwene.org")))

;; Подгрузить приватный локальный конфиг для конкретного хоста
(my-load-org-config "local/gnus-accounts.org")

;; Подгрузить шаблоны писем
(my-load-org-config "local/gnus-templates.org")

;; Способ оформления цитаты
(setq message-citation-line-function 'message-insert-formatted-citation-line)

;; Отправленные сообщения метить прочитанными (TODO работает ли?)
(setq gnus-gcc-mark-as-read t)

;; Архивировать будем в IMAP
(setq gnus-message-archive-method
      '(nnimap "Mail"
```

```
                (nnimap-stream shell)
                (nnimap-shell-program "/usr/lib/dovecot/imap"))
      ;; ... в папку sent
      gnus-message-archive-group "sent")

;; В наши трудные времена надо хотя бы сделать вид, что ты обеспечиваешь безопасност
(setq mml2015-use 'epg
      ;; немного дебага
      mml2015-verbose t
      ;; шифровать и для себя
      mml-secure-openpgp-encrypt-to-self t
      ;; проверять подпись и у зашифрованных сообщений(?)
      mml-secure-openpgp-always-trust nil
      ;; кэшировать пароль от хранилища ключей
      mml-secure-cache-passphrase t
      ;; кэшировать пароль от хранилища на 10 часов
      mml-secure-passphrase-cache-expiry '36000
      ;; определять используемый ключ при подписи по адресу отправителя
      mml-secure-openpgp-sign-with-sender t
      ;; всегда спрашивать, надо ли расшифровать зашифрованный парт письма
      mm-decrypt-option nil
      ;; всегда проверять достоверность подписанного парта
      mm-verify-option 'always)

;; зашифрованные парты и подписи надо показывать кнопкой
(add-to-list 'gnus-buttonized-mime-types "multipart/signed")
(add-to-list 'gnus-buttonized-mime-types "multipart/encrypted")

;; Наводим красоты
(setq gnus-group-line-format "%M%S%5y%6t: %(%g%)\n"
      gnus-user-date-format-alist '((t . "%Y-%m-%d %H:%M"))
      gnus-summary-line-format "%U%R %&user-date; %B %[%-23,23a%] %s\n")

(when window-system
  (setq gnus-sum-thread-tree-indent "  "
        gnus-sum-thread-tree-root " "
        gnus-sum-thread-tree-false-root "◯ "
        gnus-sum-thread-tree-single-indent " "
        gnus-sum-thread-tree-vertical        ""
        gnus-sum-thread-tree-leaf-with-other " "
```

```
          gnus-sum-thread-tree-single-leaf     " "))
  )
```

1.17 Dired

Обновлять буферы dired автоматом при изменении директории:

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

1.18 Спортивные утилиты

```
(defun sport/equipment-report-weights (src)
  "Return summary weights grouped by buggage type"
  (let
      ((vals
        (let ((cols (- (length (first src)) 2)))
                (seq-reduce '(lambda (sums row)
                               (mapcar*
                                '(lambda (v s)
                                   (+ (* (number-or-v v 0) (lst-number-or-v row 1 0)) s
                                (nthcdr 2 row)
                                sums))
                            (cdr src)
                            (make-list cols 0)))))
    (list
     (cons "ИТОГО" (nthcdr 2 (first src)))
     'hline
     (cons (seq-reduce #'+ vals 0) vals))))

(defun sport/equipment-report-baggage (src pos)
  "Return list for specified buggage type (column position in src table)"
  (cons
   '("" "Вес" "Кол-во")
   (cons
    'hline
    (sort
     (remove-if-not
      '(lambda (row) (> (nth 1 row) 0))
      (map
       #'list
       '(lambda (row)
```

```
          (let ((name (first row))
                (cnt (lst-number-or-v row pos 0)))
            (list name (* (lst-number-or-v row 1 0) cnt) cnt)))
        (cdr src)))
     (lambda (a b) (> (nth 1 a) (nth 1 b)))))))

(defun sport/equipment-report-shared (src)
  "Return list for specified buggage type (column position in src table)"
  (cons
   '("" "Вес" "Кол-во")
   (cons
    'hline
    (sort
      (map
       #'list
       '(lambda (row)
          (list
           (first row)
           (lst-number-or-v row 1 0)
           (lst-number-or-v row 6 0)))
       (remove-if
        '(lambda (row) (= (lst-number-or-v row 6 0) 0))
        (cdr src)))
      (lambda (a b) (string< (nth 0 a) (nth 0 b)))))))

(defun my-sport-journal-add (type value notes)
  (interactive
   (list
    (completing-read
     "Тип: "
     (with-current-buffer (find-file-noselect "~/org/personal/sport/sports-periodic-TOD
       (outline-show-all)
       (goto-char (point-min))
       (setq case-fold-search nil)
       (re-search-forward "^#\\+NAME: sports-journal")
       (next-line)
       (mapcar
        (lambda (row) (car (cdr row)))
        (seq-filter (lambda (row) (not (eq row 'hline))) (org-table-to-lisp)))))
    (read-string "Значение: ")
```

```
    (read-string "Заметки: ")))
  ;; TODO: improve
  (with-current-buffer (find-file-noselect "~/org/personal/sport/sports-periodic-TODO.o
    (goto-char (point-min))
    (setq case-fold-search nil)
    (re-search-forward "^#\\+NAME: sports-journal")
    (next-line)
    (goto-char (org-table-end))
    (insert
     (format "| [%s] | %s | | %s | %s |\n" (format-time-string "%F %R") type value note
```

1.19 Прочее

1.19.1 Календарь

```
(use-package
  calendar
  :ensure nil
  :config
  (setq calendar-location-name "Moscow"
        calendar-latitude 55.5
        calendar-longitude 37.4)
  ;; Хочу русский календарь!
  (setq calendar-week-start-day 1
        calendar-day-abbrev-array ["Вс" "Пн" "Вт" "Ср" "Чт" "Пт" "Сб"]
        calendar-day-header-array calendar-day-abbrev-array
        calendar-day-name-array ["Воскресенье" "Понедельник" "Вторник" "Среда" "Четверг
        calendar-month-name-array ["Январь" "Февраль" "Март" "Апрель" "Май"
                                   "Июнь" "Июль" "Август" "Сентябрь"
                                   "Октябрь" "Ноябрь" "Декабрь"]))
```

Русские праздники:

```
(use-package
  russian-holidays
  :after (calendar)
  :config
  (setq calendar-holidays russian-holidays))
```

1.19.2 Финансы

```
(use-package
  org-expenses
  :load-path "~/.emacs.d/public/org-expenses"
  :bind (:map my-bindings-map
              ("C-c f" . org-expenses/expense-view))
  :config
  (setq org-expenses/files "~/org/personal"))
```

1.19.3 Вспомогательные функции

```
(defun number-or-v (n v)
  "Return V if N is not a number, or N otherwise."
  (if (not (numberp n)) v n))

(defun lst-number-or-v (lst pos v)
  "Return V if list element at POS is not a number, or (nth POS LST) otherwise."
  (number-or-v (nth pos lst) v))
```