

Emacs Tipps & Tricks

Emacs is the Program I live in.

I don't use Emacs for everything, because it is not best for everything, but any tool that cannot be integrated with Emacs requires considerably more context switches than if it can be integrated.

Also see the [org-mode tips](#). Org-mode is what binds Emacs together for me.

Other larger topics have their own articles:

- [Emacs for Javascript development](#) — it keeps up on the raw coding part, and the integration turns it into the best tool. Jumping from the org-mode planning document into the code project and back knows no equal.
- [What I need from IntelliJ and what I miss when not using Emacs](#) — why I don't (yet?) use Emacs for Java development.

Contents

Simple EXWM setup

[EXWM](#) is a window manager running in Emacs. It's actually awesome to work in it, though it still has some rough edges.

I started using `exwm` seriously when Gnome Shell killed my keyboard layout once too often while switching users and Chromium corrupted its shortcuts when I set my keyboard layout again. Now `exwm` is my regular work environment.

I'm typing this from my Emacs running `exwm`.

Setup:

```
(use-package exwm :ensure t
  :config
  ;; Allow resizing with mouse, of non-floating windows.
  (setq window-divider-default-bottom-width 2)
  ;; do not open new frame for ediff, that would break
  (setq ediff-window-setup-function 'ediff-setup-windows-plain)
  (setq window-divider-default-right-width 2))
```

```

(window-divider-mode)
;; show the time in the modeline (also see the customization in the variables)
(display-time-mode)
;; Make buffer name more meaningful
(add-hook 'exwm-update-class-hook
          (lambda ()
            (exwm-workspace-rename-buffer exwm-class-name)))
;; disable CTRL-z (pause), because it breaks my workflows and exwm
(global-unset-key (kbd "C-z"))

(use-package pinentry
  :ensure t
  :config
  (pinentry-start))

(use-package type-break
  :custom
  (type-break-demo-boring-stats t)
  (type-break-mode-line-message-mode t)
  :config
  ;; ignore errors for type-break mode. A missing type-break stats
  ;; file can break startup otherwise.
  (ignore-errors (type-break-mode t))
  ;; the regular y-or-n-p or yes-or-no-p break exwm when in an X11 window.
  (add-hook 'exmw-init
            (lambda ()
              (setq type-break-query-function
                    (lambda (msg)
                      (ding)
                      (alert msg
                            :severity 'high
                            :title "M-x type-break"
                            :icon "warning"))))))))

```

I currently start my session as normal in Gnome Shell (to get all the monitor setup and tweaks), then start Emacs and just run

```
M-x exwm-init
```

(answer yes: replace existing window manager)

To be able to switch back, you can get the current window manager with

```
wmctrl -m
```

in guix you can install and run it in one step:

```
# guix shell wmctrl -- wmctrl -m
```

Simple EXWM setup for two screens

The tutorials I found to setup EXWM for two screens are much too complex.

In addition to the changes above, you just need `exwm-randr`.

I have two screens: HDMI-1, the primary screen, and DP-1, the secondary that is somewhat smaller than the primary.

Use `arandr` to find the correct position for the secondary screen. For me it is 1920x120: The width of the primary screen and the difference in height (resolution) between both.

```
;; Enable exwm-randr before exwm-init gets called
(use-package exwm-randr
  ;; :if dw/exwm-enabled
  :after (exwm)
  :custom
  ;; two workspaces
  (exwm-workspace-number 2)
  ;; move the second workspace to the secondary monitor,
  ;; the others are on the primary display
  (exwm-randr-workspace-monitor-plist '(1 "DP-1"))
  ;; enable switching to any buffer from any workspace
  (exwm-workspace-show-all-buffers t)
  (exwm-layout-show-all-buffers t)
  :config
  (exwm-randr-enable)
  ;; update the screen setup when a screen is turned on or off
  (add-hook 'exwm-randr-screen-change-hook
    (lambda ()
      (start-process-shell-command
        "xrandr" nil
        "xrandr --output HDMI-3 --primary --auto --pos 0x0 --output DP-1 --auto --pos 1920x120"))))
```

Controlling pulseaudio from EXWM

I need mouse-controlled volume. The `desktop-environment` package provides it, but out of the box only for `amixer`. I adjusted it for `pulseaudio`.

Also I added the volume to the `modline`, with scrollwheel control: Scroll up or down to increase or decrease volume.

```

(use-package desktop-environment :ensure t
  :custom
  (desktop-environment-screenshot-directory "~/Bilder")
  (desktop-environment-volume-normal-decrement "-5%")
  (desktop-environment-volume-normal-increment "+5%")
  (desktop-environment-volume-get-command
   "LC_ALL=C pacmd list-sinks|grep -A 15 '* index'| awk '/volume: front/{ print $5 }'")
  (desktop-environment-volume-set-command "pactl set-sink-volume 1 %s")
  (desktop-environment-volume-small-decrement "-1%")
  (desktop-environment-volume-small-increment "+1%")
  :config
  (desktop-environment-mode)
  (global-set-key (kbd "M-σ") 'desktop-environment-volume-decrement-slowly)
  (global-set-key (kbd "M-π") 'desktop-environment-volume-increment-slowly)
  (global-set-key (kbd "M--") 'desktop-environment-volume-decrement)
  (global-set-key (kbd "M-+") 'desktop-environment-volume-increment))

```

```

(define-minor-mode display-volume-mode
  "Toggle display of volume in mode lines.

```

Depends on desktop-environment.

When Display Time mode is enabled, it queries the volume from your system on every displaying using desktop-environment-volume-get (which you can customize)."

```

:global t :group 'desktop-environment
(or global-mode-string (setq global-mode-string '(""))))
(setq display-volume-keymap (make-sparse-keymap))
(define-key display-volume-keymap (kbd "<mode-line> <mouse-4>") 'desktop-environment-volume-decrement)
(define-key display-volume-keymap (kbd "<mode-line> <mouse-5>") 'desktop-environment-volume-increment)
(setq display-volume-modeline-entry
  '(:eval (propertize (format " V:%s%% "(desktop-environment-volume-get))
    'keymap display-volume-keymap)))
(if display-volume-mode
  (progn
    (or (memq display-volume-modeline-entry global-mode-string)
      (setq global-mode-string
        (append global-mode-string (list display-volume-modeline-entry))))))
  (display-volume-mode))

```

elfeed and emms: listen to podcast with convenience

[Elfeed](#) enables the nicest podcast integration I know: Treat podcast entries as regular RSS entries, but hit **A** to enqueue them in [emms](#).

Usage:

- Prepare emms with `M-x emms-playlist-mode-go`
- Update your feeds with `M-x elfeed-update`
- Read your feeds with `M-x elfeed`; hit `enter` to open an entry, then `n` and `p` to switch between them *quickly* (that's the speed I wish to have everywhere).
- Hit **A** to enqueue the file in emms.
- Start listening to media with `M-x emms-start`.
- Keep queueing up further interesting entries from elfeed.
- Do something else while listening :-)
- Pause emms with `M-x emms-pause` when needed.
- Setup

```
(use-package elfeed :ensure t)
(elfeed-add-feed "https://www.deutschlandfunk.de/informationen-am-morgen-102.xml")
(require 'emms-setup)
(emms-all)
(emms-default-players)
(setq emms-source-file-default-directory "/path/to/your/media/files/")
```

[2022-02-25 Fr]

Save the frame configuration to a register

With `C-x r f LETTER` (`frame-configuration-to-register`) and `C-x r j LETTER` (`jump-to-register`) you can store and restore the the exact layout of buffers in Emacs.

When you use `exwm` that's almost a super-power, because you can prepare several exact window setups for different tasks, but it is also useful in regular emacs usage

This is similar to switching between perspectives in Eclipse, but if you use `exwm`, this encompasses *all programs*.

Use the regular window splitting in Emacs (`C-x 2` for each vertical split, `C-x 3` for each horizontal split), then hit `C-x r f LETTER` to store the exact setup in the letter `LETTER`. Then hack away, move around, split and unify buffers, whatever you need.

Then hit `C-x r j` LETTER to restore the saved setup, down to the positions of the cursors in all the files.

When I have the files in front of me that I need right now and a colleague asks me something, `C-x r f a` stores the window, `C-x 1` switches to single-window layout and we look together. When we're done, `C-x r j a` restores my context and I continue hacking.

I have not used more than 3 frame-configurations at the same time yet, but even like that this has already been a huge asset.

Aside: You can also use the registers to store content, position in the file, and even the position of all Emacs windows. For details see <https://www.emacswiki.org/emacs/Registers>

[2022-06-18 Sa]

select pgp key to use for mu4e

If recipients cannot decrypt your emails, check the Mml Secure Key Preferences:

```
M-x customize-variable mml-secure-key-preferences
```

There you can add their key.

[2022-07-21 Do]

make-window-dedicated: prevent replacing of buffers

When you use commands that open new buffers or can split windows — or if emms opens videos in exwm buffers — it is annoying when emacs replaces some content you always want to see. To fix that, you can use `make-window-dedicated`:

```
(defun make-window-dedicated (arg)
  "Set the current window as dedicated,
so it will not be chosen as target for other buffers.
With C-u (ARG != 1; some prefix argument)
set window as non-dedicated."
  (interactive "p")
  (set-window-dedicated-p nil (if (= 1 arg) t nil)))
```

Usage:

- Enter the window with the buffer you want to keep visible.
- M-x `make-window-dedicated` to prevent replacing the buffer.

- C-u M-x `make-window-dedicated` in a dedicated window to make it non-dedicated.

[2022-08-18 Do]

Create simple packages (tutorial)

Here's a complete and quick to read description for creating and submitting your own Emacs packages. It covers the whole process and shows how to customize the compilation mode:

<https://spin.atomicobject.com/2016/05/27/write-emacs-package/>

LSP and DAP mode (language server and debug adapter)

With language servers and debug adapters, Emacs can be used for languages that require complex IDE support to work efficiently.

See <https://emacs-lsp.github.io/dap-mode/page/configuration/> and <https://github.com/emacs-lsp/dap-mode>

For Java this does not scale to the source repository at work, but for Typescript it is good enough.

For Javascript `js2-mode` is better.

[2022-10-02 So]

Use ido everywhere, except in specific functions

Ido is great to select existing entries, with `ido-cr+` even more so, but if I want to create a new one, or just want the last matching folder, it can be annoying. To precisely disable ido (and `ido-cr+`) there, use M-x `customize-variable` to change `ido-read-file-name-non-ido` and `ido-cr+-disable-list`. I do that in my ido-preview setup:

```
(use-package ido-completing-read+ :ensure t :defer 1
  :config
  ;; disable ido-cr+ where it does not fit.
  (add-to-list 'ido-cr+-disable-list 'journal t)
  (add-to-list 'ido-cr+-disable-list 'gnus-mime-save-part t)
  (add-to-list 'ido-cr+-disable-list 'bbdb-read-string)
  :custom
```

```

(ido-cr+-auto-update-blacklist t)
(ido-cr+-function-whitelist nil)
(ido-cr+-max-items 30000)
(ido-cr+-replace-completely nil)
(ido-ubiquitous-mode t))

(use-package ido-preview ;; no need to ensure: it is part of emacs
:defer 2
:config
(add-hook 'ido-setup-hook
  (lambda()
    (define-key ido-completion-map (kbd "C-M-p") (lookup-key ido-completion-map
      (define-key ido-completion-map (kbd "C-M-n") (lookup-key ido-completion-map
      (define-key ido-completion-map (kbd "C-p") 'ido-preview-backward)
      (define-key ido-completion-map (kbd "C-n") 'ido-preview-forward)))
;; disable ido where it does not fit.
(add-to-list 'ido-read-file-name-non-ido 'journal t)
(add-to-list 'ido-read-file-name-non-ido 'gnus-mime-save-part t)
:custom
(ido-all-frames t)
(ido-buffer-disable-smart-matches nil)
(ido-enable-dot-prefix t)
(ido-enable-flex-matching t)
(ido-everywhere t)
(ido-max-work-file-list 10)
(ido-max-dir-file-cache 20)
;; reduce number of prospects to reduce window size changes
(ido-max-prospects 10)
(ido-mode (quote both) nil (ido))
(ido-read-file-name-non-ido
  '(tramp-rename-these-files tramp-rename-files
    journal gnus-mime-save-part))
(ido-use-filename-at-point (quote guess))
(ido-use-url-at-point t))

;; Ignore ido save errors; they can block emacs shutdown on disk-full,
;; and the history is not important enough for that.
(defun ido-kill-emacs-hook ()
  (ignore-errors (ido-save-history)))

;; my disable list for ido-cr+:
;; (ido-cr+-disable-list
;; '(read-file-name-internal read-buffer
;;   internal-complete-buffer todo-add-category

```



```
;; gnus-emacs-completing-read gnus-iswitchb-completing-read
;; grep-read-files magit-builtin-completing-read ess-completing-read
;; Info-read-node-name tmm-prompt org-tags-completion-function
;; ffap-read-file-or-url ffap-read-file-or-url-internal
;; sly-read-symbol-name org-olpath-completing-read bbdb-read-string
;; journal gnus-mime-save-part)
```

[2022-10-27 Do]

type-break with image-slideshow from workrave using slideview

Show workrave exercises during typing breaks. Requires [workrave](#) installed. This replaces workrave in exwm for me, because workrave tends to lock up the screen (I do not know why).

```
;; add demo images to type-break
(use-package slideview :ensure t
  :config
  (add-hook 'image-mode-hook 'slideview-mode)
  (defun type-break-demo-slideview ()
    "Take a typing break with a training slideshow."
    (and (get-buffer "*Slideview*")
         (kill-buffer "*Slideview*")))
  (condition-case ()
    (progn
      (with-temp-buffer
        (rename-buffer "*Slideview*")
        (let* ((files (directory-files
                      "~/guix-profile/share/workrave/exercises/" t))
              (len-files (length files))
              ;; compensate that random selection of a range
              ;; start favors later entries.
              (compensate 5)
              (random-exercise-index (- (random len-files) compensate))
              (starting-exercise-index
               (if (< random-exercise-index compensate)
                   (random (min compensate len-files))
                   random-exercise-index))
              (excercise (nth starting-exercise-index files)))
              (find-file excercise))
          (let ((timer (run-with-timer 30 10 'slideview-next-file)))
            ;; Wait for user to come back.
```

```

        (read-event)
        (type-break-catch-up-event)
        (cancel-timer timer)))
    (kill-buffer "*Slideview*"))
  (quit
   (read-event)
   (type-break-catch-up-event)
   (and (get-buffer "*Slideview*")
        (kill-buffer "*Slideview*"))))
  (add-to-list 'type-break-demo-functions 'type-break-demo-slideview)
)

```

[2022-12-12 Mo]

find-current-as-root

I often find myself needing to edit a file in `/etc/`.

To avoid having to remember the tramp syntax for this, I have the command `M-x find-current-as-root`. This simply re-opens the current file in the current buffer with root permissions.

To get it, just add the following to your `~/.emacs.d/init.el` or `~/.emacs`:

```

;;; Open files as root - quickly
(defcustom find-file-root-prefix "/sudoedit::"
  "Tramp root prefix to use.")

(defun find-file-as-root ()
  "Like `ido-find-file', but automatically edit the file with
root-privileges (using tramp/sudo), if the file is not writable by
user."
  (interactive)
  (let ((file (ido-read-file-name "Edit as root: ")))
    (unless (file-writable-p file)
      (setq file (concat find-file-root-prefix file)))
    (find-file file)))
  ;; or some other keybinding...
  ;; (global-set-key (kbd "C-x F") 'djcb-find-file-as-root)

(defun find-current-as-root ()
  "Reopen current file as root"
  (interactive)

```

```
(set-visited-file-name (concat find-file-root-prefix (buffer-file-name)))  
(setq buffer-read-only nil))
```

[2022-12-15 Do]

see complete changes with M-x diff-hl-set-reference-rev

When you're working on a patch series you may want to see not just the lines changed compared to the last commit, but the change by your patch series. You can do that in by selecting the last commit before your changes as reference revision:

M-x diff-hl-mode

M-x diff-hl-set-reference-rev RET <revision>

Now your buffers fringes will highlight added, removed and changed lines compared to this reference revision.

That can come in very handy when you're creating atomic commits.

Or when you step back into a project after a 2 years break to finish a pull-request.

[2023-01-17 Di]

Org-Agenda with desktop notifications and Outlook calendar ics

This is my emacs OWA calendar setup. Replace `example.com` and `SECRET` by the right values.

```
(use-package org-agenda  
  :defer 6  
  :custom  
  (org-agenda-diary-file "~/emacs.d/diaries/calexport")  
  (calendar-mark-diary-entries-flag t)  
  (calendar-view-diary-initially-flag t)  
  (alert-default-style 'libnotify)  
  (appt-disp-window-function 'alert-for-appt)  
  (org-agenda-include-diary t)  
  (appt-delete-window-function (lambda ()))  
  (org-agenda-clock-consistency-checks  
   (quote  
    (:max-duration "12:00" :min-duration 0 :max-gap "0:05" :gap-ok-around
```

```

("4:00" "12:00")
:default-face
((:background "DarkRed")
 (:foreground "white"))
:overlap-face nil :gap-face nil :no-end-time-face nil :long-face nil :short-face nil
(org-agenda-clockreport-parameter-plist (quote (:link t :maxlevel 2 :properties ("E
(org-agenda-start-with-clockreport-mode nil)
:config
;; Rebuild the reminders everytime the agenda is displayed
(add-hook 'org-agenda-finalize-hook (lambda () (org-agenda-to-appt t)))
;; Run once when Emacs starts
(org-agenda-to-appt t)
;; Activate appointments so we get notifications
(appt-activate t)
;; Add calendar, option 3 https://www.ict4g.net/adolfo/notes/emacs/emacs-caldav.htm
(setq diary-location "~/.emacs.d/diaries/")
(add-hook 'diary-list-entries-hook 'diary-include-other-diary-files)
(add-hook 'diary-mark-entries-hook 'diary-mark-included-diary-files)
(setq calendars
  '(("calexport" . "https://example.com/owa/SECRET@example.com/SECRET/calendar.
(defun getcal (url file)
  "Download ics file and add it to file"
  (let ((tmpfile (url-file-local-copy url)))
    (icalendar-import-file tmpfile file)
    (kill-buffer (car (last (split-string tmpfile "/" ))))))))
(defun getcals ()
  "Load a set of ICS calendars into Emacs diary files"
  (interactive)
  (save-mark-and-excursion
    (save-window-excursion
      (with-silent-modifications
        (mapcar #'(lambda (x)
                    (let ((file (concat diary-location (car x)))
                          (url (cdr x)))
                      (message (concat "Loading " url " into " file))
                      (find-file file)
                      ;; (flush-lines "^[& ]") ;; if you import ical as non marking
                      (erase-buffer) ;; to avoid duplicating events
                      (getcal url file)))
                  calendars))))))
(defun appt-reparse-diary-file ()
  "force reparsing the diary file"
  (appt-check t))
(add-to-list 'midnight-hook 'getcals)

```

```
(add-to-list 'midnight-hook 'appt-reparse-diary-file)
)
```

[2023-01-17 Di]