Windows

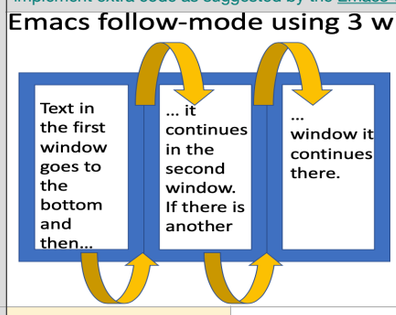
Operation	Keystroke	Function	Notes
Move to window (and window into another frame in graphics mode)	Note PEL does not use the default windmove key bindings (which use the standard cursor keys with Shift) in order to preserve the ability to move the cursor while marking. Instead PEL does the following configurations: • it configures new bindings with the <f11> key, making the operation available in terminal mode. • On macOS, in graphics mode, the ⌘ key is mapped to the super prefix key (s-). This does not work in terminal mode. • ❖ On Windows, the Menu key is mapped to the hyper key. Below the ❖ icon is used to represent the Menu key under Windows. • In graphics mode, the commands also move to frames, either by window number inside another frame, or using the cursor commands when moving away from the edge window toward a frame present at the edge identified by the cursor's direction. In graphics mode, one can always use the mouse for these operations. In terminal mode, the mouse operation is also available if the xterm-mouse-mode is enabled. The PEL package binds `<f11><f12>` to toggle the xterm-mouse-mode.		
PEL Window Management Hydra	• With PEL user option **pel-use-hydra** set to **t**, PEL activates the **hydra external package** and also creates a Hydra set of keys to help speed up navigation and management of windows. These keys are identified in the table below. • To start this Hydra, hit the `<f7>` key, then hit one of the following keys once or several times. • The keys that are in the PEL window hydra are all identified below with a `<f7>` prefix, but after typing `<f7>` once, you can hit several other window hydra keys without typing the `<f7>` prefix again. • While active the Hydra Hint is normally shown in the minibuffer (like the one shown below). While the Hydra is active use the `?` key to toggle it off or back on. To have the Hydra hint off when the Hydra activates set the hydra-is-helpful user option to nil (but then you can still toggle it on/off with `?`. • To cancel the Hydra hit the `<f7>` or `q` key or use a command not available in the windows management hydra. • The name of the PEL window hydra commands are not listed below. They all have a name that begins with **pel-∑wnd/** and ends with the same name as the command function listed in the Function column. For example, **pel-∑wnd/windmove-up** is bound to `<f7> <up>`. A snapshot of the window management hydra hint menu that shows up in the minibuffer area as soon as one of its keys is pressed is shown below.		

```
Move            | Resize      | Split            | Split to              | Layout            | Close            | Close 1                | End
----------------| ------------| -----------------| ----------------------| ------------------| -----------------| -----------------------| ------------
   <up>: up     | =: balance  | |: vertically    |   C-<up>: above       |   n: next layout  | d: this window   |   C-S-<up>: above      |   ?: hint
 <down>: down   | V: taller   | 3: vertically    |  C-<down>: below      |   p: last layout  | 0: this window   |  C-S-<down>: below     |   q: cancel
 <left>: left   | v: shorter  | _: horizontally  |  C-<left>: left       |   x: swap with..  | K: &kill buffer  |  C-S-<left>: left      | <f7>: cancel
<right>: right  | H: wider    | 2: horizontally  | C-<right>: right      | M-v: flip vert.   | .: all others    | C-S-<right>: right     |
                | h: narrower |                  |                       | M-h: flip horiz.  | 1: all others    |                        |
Showing Hydra Hint
```

Move to window above	• `<f11> <up>` • ⌘-`<up>` • ❖ `-<up>` • `<f7><up>`	(**windmove-up** &optional ARG)	Select the window above the current one. • With no prefix argument, or with prefix argument equal to zero, "up" is relative to the position of point in the window; otherwise it is relative to the left edge (for positive ARG) or the right edge (for negative ARG) of the current window. • If no window is at the desired location, an error is signaled.
Move to window below	• `<f11><down>` • ⌘-`<down>` • ❖ `-<down>` • `<f7><down>`	(**windmove-down** &optional ARG)	Select the window below the current one. • With no prefix argument, or with prefix argument equal to zero, "down" is relative to the position of point in the window; otherwise it is relative to the left edge (for positive ARG) or the right edge (for negative ARG) of the current window. • If no window is at the desired location, an error is signaled.
Move to window at right	• `<f11><right>` • ⌘-`<right>` • ❖ `-<right>` • `<f7><right>`	(**windmove-right** &optional ARG)	Select the window to the right of the current one. • With no prefix argument, or with prefix argument equal to zero, "right" is relative to the position of point in the window; otherwise it is relative to the top edge (for positive ARG) or the bottom edge (for negative ARG) of the current window. • If no window is at the desired location, an error is signaled.
Move to window at left	• `<f11><left>` • ⌘-`<left>` • ❖ `-<left>` • `<f7><left>`	(**windmove-left** &optional ARG)	Select the window to the left of the current one. • With no prefix argument, or with prefix argument equal to zero, "left" is relative to the position of point in the window; otherwise it is relative to the top edge (for positive ARG) or the bottom edge (for negative ARG) of the current window. • If no window is at the desired location, an error is signaled.
Move point to other window **- C-u: swap** **- C-u C-u: delete**	`C-x o`	(**ace-window** ARG) — — — — — — — — — — — — — (**other-window** COUNT &optional ALL-FRAMES)	Select (move point) to other window. • If there are several windows, prompt for a window number (using `ace-window' mechanism). • If there is only 2 windows in the frame and no argument : select other window without prompting. This command allows selecting any window, even one that is inside another Emacs frame. • In graphics mode the other Emacs frames are in other OS window. • In text terminal mode, other Emacs frames are hidden (as they occupy the exact same OS window): just one Emacs frame is displayed. An argument can be used to perform more operations: • To force a window number prompt, use any negative prefix (including just typing `C- -` alone). This is useful when there's several frames and the current frame has 1 or 2 windows active. • Prefixed with one `C-u`, does a **swap** between the selected window and the current window, so that the selected buffer moves to current window (and current buffer moves to selected window). The PEL `<f11> w x` key does the same (but does not prompt when there are only 2 windows.) • Prefixed with two `C-u`'s, **deletes** the window identified by the window number. Demo: C'est la Z, video 5 Depends on the **ace-window** external package. • On PEL, customize ***pel-use-ace-window*** to **t** to activate the feature. • When *pel-use-ace-window* is **nil** (the default), `C-x o` is bound to Emacs standard behaviour function (**other-window**).
Move point to next window	`<f11> w o`	(**pel-other-window**)	Execute (other-window 1). • Useful when '**other-window**' has been remapped to something like 'ace-window' and want to see where the *next* window is.
Move point to previous window	`<f11> w O`	(**pel-other-window-backward** &optional N)	Select Nth previous window. • n defaults to 1 : meaning direct previous window. • This is the inverse of what does the standard (other-window). • This command might be useful when ace-window is not used.
Exchange windows	• `<f11> w x` • `<f7> x`	(**ace-swap-windows**)	Swap buffers of the current window with another. If 3 windows or more, a single digit shows up in the top-left corner identifying the number to type to swap to this window. Depends on the **ace-window** external package. • On PEL, customize ***pel-use-ace-window*** to **t** to activate the feature.
Close/Create Windows	The following commands are used to create and remove windows. The last 2 rows correspond to two sets of four PEL commands bound to cursor keys.		
Close this windows	• `C-x 0` • `<f7> 0` • `<f7> d`	(**delete-window** &optional WINDOW)	This just closes the window and moves the cursor to the next window.
Kill current buffer and close window	• `C-x 4 0` • `<f7> K`	(**kill-buffer-and-window**)	Kill the current buffer and delete the selected window.

Operation	Keystroke	Function	Notes
Close a window identified by number	`<f11> w k`	(**ace-delete-window**)	Delete a window selected by a number, a number shown in the top-left corner of the window. 📦 Depends on the **ace-window** external package. • On PEL, customize ***pel-use-ace-window*** to **t** to activate the feature.
Close all other windows	• `C-x 1` • `<f7> 1` • `<f7> .`	(**delete-other-windows** &optional WINDOW)	Make current window fill its frame.
Maximize one window, identified by number	`<f11> w m`	(**ace-maximize-window**) — — — — — — — — — — — — — (**ace-delete-other-windows**)	Maximize a window. Close all windows except the window selected by number, a number shown in the top-left corner of the window. 📦 Depends on the **ace-window** external package. The old versions of ace-window used ace-window-maximize, but newer versions use ace-delete-maximize-windows. PEL uses the one that is available. • On PEL, customize ***pel-use-ace-window*** to **t** to activate the feature.
Create new window below	• `C-x 2` • `<f7> 2` • `<f7> -`	(**split-window-below** &optional SIZE)	Split the selected window into two windows, one above the other. • The selected window is above. The newly split-off window is below and displays the same buffer. ☛ Note that Emacs default behaviour attempts to maximize the view into the current buffer when splitting the buffer into 2 windows. This means that the cursor will not be located in the same position in the new window. To change this behaviour and keep the same point in both windows, execute *(setq split-window-keep-point nil).* The PEL packages does that.
Create new window at right	• `C-x 3` • `<f7> 3` • `<f7> |`	(**split-window-right** &optional SIZE)	Split the selected window into two side-by-side windows. • The selected window is on the left. The newly split-off window is on the right and displays the same buffer.
Create window at cursor direction	• `<f11> C-<right>` • `<f11> C-<left>` • `<f11> C-<down>` • `<f11> C-<up>` • `<f7> C-<right>` • `<f7> C-<left>` • `<f7> C-<down>` • `<f7> C-<up>`	• (**pel-create-window-right**) • (**pel-create-window-left**) • (**pel-create-window-down**) • (**pel-create-window-up**)	Create a window at the location pointed by the cursor's direction, and keep point inside the new window. ☛These are 4 different commands, one for each of the available cursors: **`<right>`**, **`<left>`**, **`<down>`** and **`<up>`**. ☛The **`<f11>`** commands are stand alone commands, while the **`<f7>`** are part of the PEL Hydra for Windows Management.
Close a window at cursor direction	• `<f11> C-S-<right>` • `<f11> C-S-<left>` • `<f11> C-S-<down>` • `<f11> C-S-<up>` • `<f7> C-S-<right>` • `<f7> C-S-<left>` • `<f7> C-S-<down>` • `<f7> C-S-<up>`	• (**pel-close-window-right**) • (**pel-close-window-left**) • (**pel-close-window-down**) • (**pel-close-window-up**)	Kill window pointed by the cursor's direction. ☛These are 4 different commands, one for each of the available cursors: **`<right>`**, **`<left>`**, **`<down>`** and **`<up>`**. ☛The **`<f11>`** commands are stand alone commands, while the **`<f7>`** are part of the PEL Hydra for Windows Management.
Resize Window	The following commands are used to change the current window size. None of these commands are easy to re-type quickly. The best way to use them is to type them once and then use the repeat key **`C-x z`** once and then repeat more by only typing '**`z`**'. The PEL package also binds the **`<f5>`** key to repeat.		
Grow window taller	• `C-x ^` • `<f11> w s V` • `<f7> x`	(**enlarge-window** DELTA &optional HORIZONTAL)	Grow window taller by DELTA lines (defaults to 1), specify more with **`C-u`** n (or **`M-`** n) argument prefix.
Shrink window smaller	• `<f11> w s v` • `<f7> v`	(**shrink-window** DELTA &optional HORIZONTAL)	Shrink height of window by DELTA lines (defaults to 1), specify more with **`C-u`** n (or **`M-`** n) argument prefix.
Grow windows wider	• `C-x }` • `<f11> w s H` • `<f7> H`	(**enlarge-window-horizontally** DELTA)	Enlarge the current window horizontally.
Shrink window narrower	• `C-x {` • `<f11> w s h` • `<f7> h`	(**shrink-window-horizontally** DELTA)	Reduce the width of the current window.
Reduce current window size if buffer is smaller than window	• `C-x -` • `<f11> w s -`	(**shrink-window-if-larger-than-buffer** &optional WINDOW)	Shrink height of current window if its buffer doesn't need so many lines. • More precisely, shrink window vertically to be as small as possible, while still showing the full contents of its buffer. • Do not shrink window to less than 'window-min-height' lines. Do nothing if the buffer contains more lines than the present window height, or if some of the window's contents are scrolled out of view, or if shrinking this window would also shrink another window, or if the window is the only window of its frame.
Make all windows the same size	• `C-x +` • `<f11> w s =` • `<f7> =`	(**balance-windows** &optional WINDOW-OR-FRAME)	Balance the sizes of windows of WINDOW-OR-FRAME. • WINDOW-OR-FRAME is optional and defaults to the selected frame. • If WINDOW-OR-FRAME denotes a frame, balance the sizes of all windows of that frame. If WINDOW-OR-FRAME denotes a window, recursively balance the sizes of all child windows of that window.
Quick Window Layout Change	The following commands flip the layout of 2 windows: the current and *next* window between 2 horizontal windows to 2 vertical windows and vice versa.		
Flip 2 horizontal windows to 2 vertical ones	• `<f11> w v` • `<f7> M-v`	(**pel-2-vertical-windows**)	Convert 2 horizontal windows into 2 vertical windows. • Flip the orientation of the current window and its next one.
Flip 2 vertical windows to 2 horizontal ones	• `<f11> w h` • `<f7> M-h`	(**pel-2-horizontal-windows**)	Convert 2 horizontal windows into 2 horizontal windows. • Flip the orientation of the current window and its next one.
Window Layout History	The following commands allow you to restore a previously used window layout. They depend on the **winner** package, a package that is part of the standard Emacs. PEL activates them when *pel-use-winner* customize variable is **t**.		
Restore an earlier window configuration	• `C-c <left>` • `<f11> w p` • `<f7> p`	(**winner-undo**)	Switch back to an earlier window configuration saved by Winner mode. In other words, "undo" changes in window configuration.
Restore a more recent window configuration	• `C-c <right>` • `<f11> w n` • `<f7> n`	(**winner-redo**)	Restore a more recent window configuration saved by Winner mode.
Open Buffer in another window	With the following commands you can show a different buffer inside another window. One command select that other window (move point to that window) and the other does not. Under PEL both commands are bound to the IDO version of the command when the pel-use-ido customization variable is set to **t**, otherwise they retain the Emacs default binding. The IDO binding provides more information at the prompt.		
Select buffer in other window	• `C-x 4 b` • `<f11> w B`	(**ido-switch-buffer-other-window**) — — — — — — — — — — — — (**switch-to-buffer-other-window** BUFFER-OR-NAME &optional NORECORD)	Select buffer bufname in another window (switch-to-buffer-other-window). See Select Buffer.

Operation	Keystroke	Function	Notes
Display buffer in other window, don't select the other window.	• `C-x 4 C-o` • `<f11> w b`	(**ido-display-buffer**) — — — — — — — — — — — — (**display-buffer** BUFFER-OR-NAME &optional ACTION FRAME)	Display a buffer in other window but don't select it. When *pel-use-ido* is customized to **t**, (ido-display-buffer) is used, which prompts and provides easy to select list of available buffer names. Otherwise the standard Emacs (display-buffer) is used prompting without showing the available buffers.
Dedicated Windows	Emacs windows can be dedicated to specific buffers in such a way that future windows operations do not affect the dedicated windows. The following commands help you manage dedicated windows.		
Show dedicated status of current window	`<f11> w d ?`	(**pel-show-window-dedicated-status**)	Display the dedicated status of the current window in the echo area (the minibuffer).
Toggle dedicated status of current window	`<f11> w d d`	(**pel-toggle-window-dedicated**)	Toggle the dedicated status of the current window. Use with care.

Follow Mode
 	
(See Also: ∑ Scrolling)
	Emacs has a scroll all windows mode which applies all scroll commands to all visible windows. To support mouse wheel or scroll bar you need to implement extra code as suggested by the Emacs Wiki Scroll All Mode page.
	
	 
	
			When Emacs follow-mode is used on 2 or more windows, these windows show the text of the same buffer spread across these windows that act as a one continuous stream.
			
			• Follow mode is a minor mode that combines windows into one tall virtual window. This is accomplished by two main techniques:
			  • The windows always displays adjacent sections of the buffer. This means that whenever one window is moved, all the others will follow. (Hence the name Follow mode.)
			  • Should point (cursor) end up outside a window, another window displaying that point is selected, if possible. This makes it possible to walk between windows using normal cursor movement commands.
			• Follow mode comes to its prime when used on a large screen and two or more side-by-side windows are used. The user can, with the help of Follow mode, use these full-height windows as though they were one.

Toggle follow-mode (See Also: ∑ Scrolling)	• `<f11> w f` • `<f11> | f`	(**follow-mode** &optional ARG)	Toggle Follow mode. With a prefix argument ARG, enable Follow mode if ARG is positive, and disable it otherwise.
Scrolling Window	☛ For all other commands to scroll the window text, see the ∑ **Scrolling** page.		

Windows — Reference

Topic/URL	Comment
GNU Emacs — Displaying a Buffer in a Window	Describes the Emacs features related to displaying buffers inside windows.
GNU Emacs Lisp — Displaying Buffers — The Zen of Buffer Display	Describes the rules Emacs tries to use to control the creation of new windows when they are created dynamically from commands.