# Keyboard Macros

| Description | Keystroke | Function | Note |
|---|---|---|---|
| **Keyboard Macros** | | Keyboard macros allow you to quickly record a sequence of keys and then re-execute that same sequence later. Any of the keys can be Emacs bound commands.  This is a very useful tool. | |
| **PEL Customization** | | PEL provides the pel-kbmacro customization group.<br>• Use the **`<f11> <f1> k`** key sequence to quickly access the customization buffer for the group.<br>⚏ This holds the following user options:<br>  • **pel-kbmacro-prompts**: if **t** the keyboard macro recorder will prompt before overriding an existing keyboard macro. Off (nil) by default.<br><br>PEL supports the following external packages that extend keyboard macro features:<br>  • 📦 The **centimacro** external package ☑ PEL activates it when the **pel-use-centimacro** user option is set to **t**.<br>    • See the extra commands in the centimacro section below. | |
| **Customize PEL Keyboard Macros support**<br>See also: ∑ **Customize** | **`<f11> <f1> k`** | **(pel-cfg-kbmacro** &optional OTHER-WINDOW**)** | Customize the PEL keyboard macro  support.<br>• If OTHER-WINDOW is non-nil (use **C–u**), display in another window. |
| **Common C-x C-k prefix** | | Several of the keyboard macros commands share the same **C-x C-k prefix**.<br>• Once you have typed the **C-x C-k prefix** of one of these commands, you can type just the last part for executing the others.<br>  • For example you could type "C-x C-k C-p C-k C-n C-k" to exit the previously defined keyboard macro and then execute the last defined keyboard macro.<br>  • Also since you can bind macros to single characters in the range [0-9] and [A-Z]  these can also be executed in side that string of characters. | |
| **Record & Play** | | Use **`<f3>`** and **`<f4>`** to record and play keyboard macros, instead of the older **C-x (** and **C-x )** bindings. These are easier to type and easier to use since **`<f4>`** is used both to stop recording and to execute the macro. | |
| **Start Recording** | • **`<f3>`**<br>• **`C–x (`** | • **(kmacro-start-macro-or-insert-counter** ARG**)**<br>• **(pel-kmacro-start-macro-or-insert-counter** ARG**)** | Record subsequent keyboard input, defining a keyboard macro.<br>The commands are recorded even as they are executed.While already defining a macro (with a previous F3), typing F3 inserts the current value of the keyboard macro counter into the buffer, and increments the counter by 1).  See The Keyboard Macro Counter.<br><br>• **C-u <f3>** executes the last macro then appends the keystrokes to its definition.<br>• **C-u C-u <f3>** appends keys to the last defined macro without executing it.<br>☝ By default , the PEL version of the command prompts if a macro already exists, before allowing overwriting it.<br>• Use a negative argument (**M- -**  or **C-_**) argument or numeric 0 to prevent this prompt and allow overwriting already defined macro.<br>• ☑ This behaviour is customizable.  Customize the **pel-kbmacro-prompts** variable in the **Pel/Pel Kbmacro** subgroup to change it and prevent the prompting. |
| **End Recording or call last macro** | • **`<f4>`**<br>• **`C–x e`** | **(kmacro-end-or-call-macro** ARG &optional NO-REPEAT**)** | Ends macro recording done with **`<f3>`**.  Typing **`<f4>`** again runs the last recorded macro.  This is the most convenient way to replay a recently recorded macro. Typing **C-u <f4>** runs the *second* macro in the ring.<br><br>• A prefix argument number N specified the number of times to execute the macro.<br>⚠ If N is 0 the macro will run forever until it exits with an error (such as encountering the end of the buffer) or it is manually stopped with **C-g** (or **C-<BREAK>** on DOS/Windows)! During that time the display may not even be updated!! |
| **Execute macro at the head of the macro ring** | **`C–x C–k C–k`** | **(kmacro-end-or-call-macro-repeat** ARG**)** | You can use this instead of <f4> to end a macro definition or to execute the *current* macro  (the one at the top of the keyboard macro ring).<br>It's advantageous if you want to execute another command with the **C-x C-k prefix** right after: you won't have to repeat the prefix, so you could type **C-k** again, **C-n**  or **C-p** or one of the other commands with the same prefix. |
| **Allow overwrite of recorded macro** | **`<f11> k k`** | **(pel-forget-recorded-keyboard-macro)** | Forget that a keyboard macro was recorded by F3.<br>Does not delete the macro from the keyboard macro ring. |
| **Apply macro to selected area (region)** | **`C–x C–k r`** | **(apply-macro-to-region-lines** TOP BOTTOM &optional MACRO**)** | Apply last defined keyboard macro to each line of a region. It does it line by line: by moving point to the beginning of the line and then executing the macro. |
| **Start Recording** | **`C–x (`** | **(kmacro-start-macro** ARG**)** | Record subsequent keyboard input, defining a keyboard macro.<br>An older command and binding.<br>• **C-u C-x (** appends keys to the last defined macro without executing it. |
| **End Recording** | **`C–x )`** | **(kmacro-end-macro** ARG**)** | Ends macro recording done with **C-x (**.<br>To execute the macro use **C-x e**.<br>An older  command and key binding.<br>Use if the macro recording was started with **C-x (** |
| **Naming & Saving** | | Assign a name or bind to a single key ([0-9] or [A-Z]).  Later you can use the name (with **M-x**) of the single key (with **C-x C-k** prefix) to use the macro. | |
| **Bind the most recent defined macro** | **`C–x C–k b`** | **(kmacro-bind-to-key** ARG**)** | Emacs will prompt.  Use keys [0-9A-Z].<br>Emacs then binds the last defined keystroke macro to the corresponding command **C-x C-k 0** through **9** and capital **A** to **Z**.<br>To re-run the macro: type **C-x C-k** followed by the character that identifies the macro. For example: **C-x C-k 0**  would execute macro bound to 0.<br>⚠ These bindings **do not** persist after Emacs closes. |
| **Name last defined macro** | **`C–x C–k n`**<br>*`<name>`* | **(kmacro-name-last-macro** SYMBOL**)** | The name can be any string as long as it does not conflict with the name of an existing function (in which case it won't be accepted).<br>A good convention is to use underscores or period in the names since most of the emacs functions do not use them.  For example,  "km_" or "km." as prefix.<br>To execute a name keyboard macro, use **M-x** *<name>*<br><br>⚠ The names **do not** persist after Emacs closes. If you want to retain the named keyboard macro convert it into Lisp code with **insert-kbd-macro** in a file and save that file. See below. |
| **Insert Lisp definition in buffer** | **`<f11> k i`** | **(insert-kbd-macro** MACRONAME &optional KEYS**)** | Insert in buffer the definition of kbd macro MACRONAME, as Lisp code. MACRONAME should be a symbol.<br>• Optional second arg KEYS means also record the keys it is on (this is the prefix argument, when calling interactively).<br>• This Lisp code will, when executed, define the kbd macro with the same definition it has now. If you say to record the keys, the Lisp code will also rebind those keys to the macro.  Only global key bindings are recorded since executing this Lisp code always makes global bindings.<br>• To save a kbd macro, visit a file of Lisp code such as your '~/.emacs', use this command, and then save the file. |
| **Name last defined macro** | | **(name-last-kbd-macro** SYMBOL**)** | An older implementation, similar to **kmacro-name-last-macro** but which does not put the 'kmacro property to the symbol.  I suspect this will eventually go away or become an alias for the other one.<br>Use **kmacro-name-last-macro** instead. |

| Description | Keystroke | Function | Note |
|---|---|---|---|
| **Keyboard Macro Ring** | The macro at the head of the macro ring can be executed with **\<f4>** and **C-x C-k n** will name it. The maximum number of macros in the keyboard macro ring is determined by the customizable variable '***kmacro-ring-max***'. | | |
| **Rotate the macro ring to the next (defined earlier) macro** | `C-x C-k C-n` | **(kmacro-cycle-ring-next** &optional ARG) | Move to next keyboard macro in keyboard macro ring.<br>• Displays the selected macro in the echo area.<br>• The ARG parameter is unused.<br>• You can continue to rotate the ring with a single **C-n** or **C-p** until the desired macro is at the head of the ring, and then execute it with a single **C-k** |
| **Rotate the macro ring to the previous (defined later ) macro** | `C-x C-k C-p` | **(kmacro-cycle-ring-previous** &optional ARG) | Move to previous keyboard macro in keyboard macro ring.<br>• Displays the selected macro in the echo area.<br>• The ARG parameter is unused.<br>• You can continue to rotate the ring with a single **C-n** or **C-p** until the desired macro is at the head of the ring, and then execute it with a single **C-k** |
| **Delete current macro from Keyboard macro ring** | `C-x C-k C-d` | **(kmacro-delete-ring-head** &optional ARG) | Delete current macro from keyboard macro ring.<br>• The ARG parameter is unused. |
| **Keyboard macro counter** | A counter is associated with each keyboard macro. A macro can be defined to insert the integer value of the counter in the text. While defining the macro, you can type either \<f3> or C-x C-k C-i to insert the counter. You can also define a way to format the counter : use **C-x C-k C-f** before defining the macro. With these facilities you can easily create a macro to number lines. For example, consider the following commands:<br>　• `C-x C-k C-f %02d`<br>　• `<f3> C-a <f3> . SPC <f4>`<br><br>• That will create lines with: "00.", "01. ", "02. ",  etc…  By default the counter start at 0.  If you want another value,  initialize it to another value specify it as a numeric argument to \<**f3**>. | | |
| **Insert current counter & increment by 1** | `<f3>` | • **(kmacro-start-macro-or-insert-counter** ARG)<br>• **(pel-kmacro-start-macro-or-insert-counter** ARG) | While defining a macro typing **\<f3>** inserts the macro counter in the buffer. |
| **Insert keyboard macro counter value in the buffer, then increment it by 1 (or ARG)** | `C-x C-k C-i` | **(kmacro-insert-counter** ARG) | Insert current value of 'kmacro-counter', then increment it by ARG.<br>• Interactively, ARG defaults to 1.<br>• With **C-u,** insert the previous value of '***kmacro-counter***', and do not increment the current value. The previous value of the counter is the one it had before the last increment.<br>• Can be typed while defining a macro, but also outside. |
| **Set keyboard macro counter** | `C-x C-k C-c` | **(kmacro-set-counter** ARG) | Set the value of '***kmacro-counter'*** to ARG, or prompt for value if no argument. With **C-u** prefix, reset counter to its value prior to this iteration of the macro. |
| **Add prefix arg to the keyboard macro counter** | `C-x C-k C-a` | **(kmacro-add-counter** ARG) | Add the value of numeric prefix arg (prompt if missing) to '***kmacro-counter***'. With **C-u**, restore previous counter value. |
| **Specify the format for inserting the keyboard macro counter** | `C-x C-k C-f` | **(kmacro-set-format** FORMAT) | Set the format of '***kmacro-counter***' to FORMAT.  See formatting strings.<br>This allows controlling the string inserted when the macro counter is inserted.  The text provided should contain a format entry for **one** integer.  The enclosing double quotes are ignored.<br>☞ The format string must be defined **before** defining the macro. |
| **Macros with variations** | You can force macro execution to stop, query for action. Use C-x q during macro definition to identify that point.  When the macro runs, it will prompt at that point with C-l, C-r, y, n, q. | | |
| **Insert a query in a macro** | `C-x q` | **(kbd-macro-query** FLAG) | Used when defining a macro to force a query when the macro will be executed.<br>• With prefix argument (**C-u**), enters recursive edit, reading keyboard commands even within a kbd macro.  You can give different commands each time the macro executes.<br>• Without prefix argument, asks whether to continue running the macro. Your options are:<br>　• y　　Finish this iteration normally and continue with the next.<br>　• n　　Skip the rest of this iteration, and start the next.<br>　• RET　Stop the macro entirely right now.<br>　• C-l　Redisplay the screen, then ask again.<br>　• C-r　Enter recursive edit; ask again when you exit from that.<br><br>Later during execution of that macro, use **C-M-c** to exit the recursive edit. |
| **Editing macros** | You can edit the content of a keyboard macro with the following commands.  They format the macro definition in a buffer and enter a specialized mode to edit it.  Type **C-c C-c** to complete the editing.<br>☞These work better in **graphics** mode; in terminal mode the cursor and meta keys are recorded as escape sequences and that's what is shown in the key lossage.  It makes the macro actions more difficult to read. | | |
| **Edit the last defined keyboard macro** | `C-x C-k C-e` | **(kmacro-edit-macro-repeat** &optional ARG) | Edit last keyboard macro. |
| **Edit the last defined keyboard macro** | `C-x C-k <RET>` | **(kmacro-edit-macro** &optional ARG) | As edit last keyboard macro, but without kmacro-repeat property. |
| **Edit a previously defined macro name** | `C-x C-k e`<br>`<name>` | **(edit-kbd-macro** KEYS &optional PREFIX FINISH-HOOK STORE-HOOK) | Edit a keyboard macro.<br>• At the prompt, type any key sequence which is bound to a keyboard macro.<br>• Or, type **C-x e** or **\<RET>** to edit the last keyboard macro, **C-h l** to edit the last 300 keystrokes as a keyboard macro, or **M-x** to edit a macro by its command name.<br>• With a prefix argument, format the macro in a more concise way. |
| **Edit the last 300 keystrokes as a keyboard macro** | `C-x C-k l` | **(kmacro-edit-lossage)** | Edit most recent 300 keystrokes as a keyboard macro.  No mouse click allowed in the last 300 events for this to work. |
| **Stepwise macros editing** | You can also interactively execute and edit a keyboard macro with the following command. | | |
| **Stepwise Replay/Edit** | `C-x C-k SPC` | **(kmacro-step-edit-macro)** | Step edit and execute last keyboard macro.<br><br>Executes the keyboard macro but before each command prompts the user for action, including:<br>• y　　　execute current command, go to next one<br>• n, d　skip & delete current command<br>• f　　　skip current command (don't delete)<br>• \<tab> execute current commands and all similar in sequence<br>• c　　　continue execution without further editing<br>• C-k　 skip and deletes rest of macro, terminate editing and replace<br>• q, C-g　cancel editing, ignore any editing changes<br>• i　　　insert/execute following keys in macro up until **C-j** in macro<br>• I　　　insert/execute one key sequence in macro<br>• r　　　replace current command with read/executed keys in macro up until **C-j**, advance over the inserted key sequence.<br>• R　　 read & execute key sequence, replace current command with it, advance over the inserted key sequence.<br>• a　　　append & execute key sequence up until **C-j**.  Append to end of macro and advance overt the inserted key sequence.<br>• A　　　Same as 'a' but after appending stops editing & replace original macro. |

| Description | Keystroke | Function | Note |
|---|---|---|---|
| **centimacro** | | 📦 The **centimacro** external package 🔳 PEL activates it when the **pel-use-centimacro** user option is set to **t**.<br>• The centimacro package provides the ability to create keyboard macros that are bound to keys other than **<f4>**.<br>• To create a keyboard macro binding you use the **centi-assign** command, identify the key that will playback the keyboard macro, type the keys that constitute the keyboard macro and terminate it by the same playback key.<br>• The bindings can override another key binding and the centimacro bindings can be deleted, restoring the original bindings by executing the **centi-restore-all** command.<br>• The centimacro bindings are not persistent: they are not kept once Emacs is closed.<br>• By default, centimacro maps the **<f5>** key to centi-assign. It's a problem for PEL as PEL uses that key as the repeat key.<br>  • 🔳 To solve that PEL issue, PEL provides the **pel-centi-assign-key** user option to identify the default key for centi-assign and reassigns the key on startup. By default it uses the **<C-f5>** key instead. But also not the other key binding below, which is always available with PEL.<br>  • Therefore, when using PEL, just leave the centi-assign-key user option unchanged to **<f5>** and control the key binding with pel-centi-assign-key instead. PEL code restores the binding of **<f5>** to repeat and uses pel-centi-assign-key to specify the key binding used for centi-assign.<br>☝ With PEL, the **<C-f5>** key binding becomes active only once you invoked a centimacro command through one of the the **<f11> k** keys. | |
| **Record a keyboard macro bound to a specified key** | • **<C-f5>**<br>• **<f11> k =** | **(centi-assign)** | Record a keyboard macro for a specified KEY.<br>• Read a KEY and start recording a keyboard macro for it.<br>• Pressing KEY again stops recording and assigns the macro to KEY.<br>• Aborts if KEY belongs to a minor mode.<br>• Use 'centi-summary' to list bound macros.<br>• Use 'centi-restore-all' to un-bind macros and restore the old key bindings. |
| **Show all keyboard macros created by centi-assign** | **<f11> k ?** | **(centi-summary)** | Show a summary of bound macros created by centi-assign. |
| **Unbind all keyboard macros created by centi-assign** | **<f11> k DEL** | **(centi-restore-all)** | Unbind all bound macros created by centi-assign, restoring the previous key bindings. |

## Keyboard Macros — References

| Topic & Link | Description |
|---|---|
| **GNU Emacs manual - Keyboard Macros** | |
| **GNU Emacs manual - Keyboard Macros - Basic Usage** | |
| **GNU Emacs - naming, saving macros** | |
| **GNU Emacs - macros with variations** | |
| **GNU Emacs - keyboard macro counter** | |
| **GNU Emacs- editing keyboard macro** | |
| **Emacs Wiki - Keyboard Macros** | |
| **Emacs Wiki - Keyboard Macros Tricks** | |
| **Code example 2 - adding more keys to run macros** | |
| **Introduction - very basic** | |
| **Example @ Ergoemacs** | |
| **Stepwise Editing a Keyboard Macro** | |
| **Building macros with pause, prompts, etc..** | |
| **Define, name and run keystroke macros** | |
| | |
| **Extra Notes** | 2 elisp files implement these functions: macros.el and kmacro.el . The latter appears to be newer and provides more functionality. For example: it provides the (**kmacro-name-last-macro** SYMBOL) similar to the (**name-last-kbd-macro** SYMBOL). It's almost the same code except that it puts the macro property to the symbol, which the older function does not do. I would think that these 2 files should be merged in future versions of Emacs to reduce code bloat. |