Using emacs efficiently at EPITA
and other places of the universe

mefyl <mefyl@lrde.epita.fr>

October 5, 2009

Outline

Outline

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.
- Closes emacs.

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.
- Closes emacs.
- Compiles: `gcc *.c`.

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.
- Closes emacs.
- Compiles: `gcc *.c`.
- Gets 100 lines of errors.

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.
- Closes emacs.
- Compiles: `gcc *.c`.
- Gets 100 lines of errors.
- Scrolls up to the first error.

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.
- Closes emacs.
- Compiles: `gcc *.c`.
- Gets 100 lines of errors.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.
- Closes emacs.
- Compiles: `gcc *.c`.
- Gets 100 lines of errors.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Frack, what line was it again?

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.
- Closes emacs.
- Compiles: `gcc *.c`.
- Gets 100 lines of errors.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Frack, what line was it again?
- Closes emacs.

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.
- Closes emacs.
- Compiles: `gcc *.c`.
- Gets 100 lines of errors.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Frack, what line was it again?
- Closes emacs.
- Scrolls up to the first error.

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.
- Closes emacs.
- Compiles: `gcc *.c`.
- Gets 100 lines of errors.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Frack, what line was it again?
- Closes emacs.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.
- Closes emacs.
- Compiles: `gcc *.c`.
- Gets 100 lines of errors.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Frack, what line was it again?
- Closes emacs.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Press `N` times `down`, `N` being the error line.

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.
- Closes emacs.
- Compiles: `gcc *.c`.
- Gets 100 lines of errors.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Frack, what line was it again?
- Closes emacs.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Press `N` times `down`, `N` being the error line.
- Oh crap, what error message was it?

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.
- Closes emacs.
- Compiles: `gcc *.c`.
- Gets 100 lines of errors.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Frack, what line was it again?
- Closes emacs.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Press `N` times `down`, `N` being the error line.
- Oh crap, what error message was it?
- Anyway... fixes the error.

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.
- Closes emacs.
- Compiles: `gcc *.c`.
- Gets 100 lines of errors.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Frack, what line was it again?
- Closes emacs.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Press `N` times `down`, `N` being the error line.
- Oh crap, what error message was it?
- Anyway... fixes the error.
- Closes emacs.

Work cycle, you're doing it wrong.

Xavier Login (login_x) starts working on his project.

- Launches emacs.
- Edits.
- Closes emacs.
- Compiles: `gcc *.c`.
- Gets 100 lines of errors.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Frack, what line was it again?
- Closes emacs.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Press `N` times `down`, `N` being the error line.
- Oh crap, what error message was it?
- Anyway... fixes the error.
- Closes emacs.
- Press `up` a few times to retrieve the compile command.

Work cycle, you're doing it wrong.

- Launches emacs.
- Edits.
- Closes emacs.
- Compiles: `gcc *.c`.
- Gets 100 lines of errors.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Frack, what line was it again?
- Closes emacs.
- Scrolls up to the first error.
- Launches emacs on the erroneous file.
- Press `N` times `down`, `N` being the error line.
- Oh crap, what error message was it?
- Anyway... fixes the error.
- Closes emacs.
- Press `up` a few times to retrieve the compile command.
- Iterates until it compiles.

Work cycle, you're doing it wrong.

What was wrong?

- Constantly closing and starting emacs.
- Navigating by taping keys like a drummer.
- Compiling in the shell instead of emacs.
- Performing complex repetitive actions.

Outline

Some vocabulary.

- Frame: an emacs window, in the KDE sense of the term.

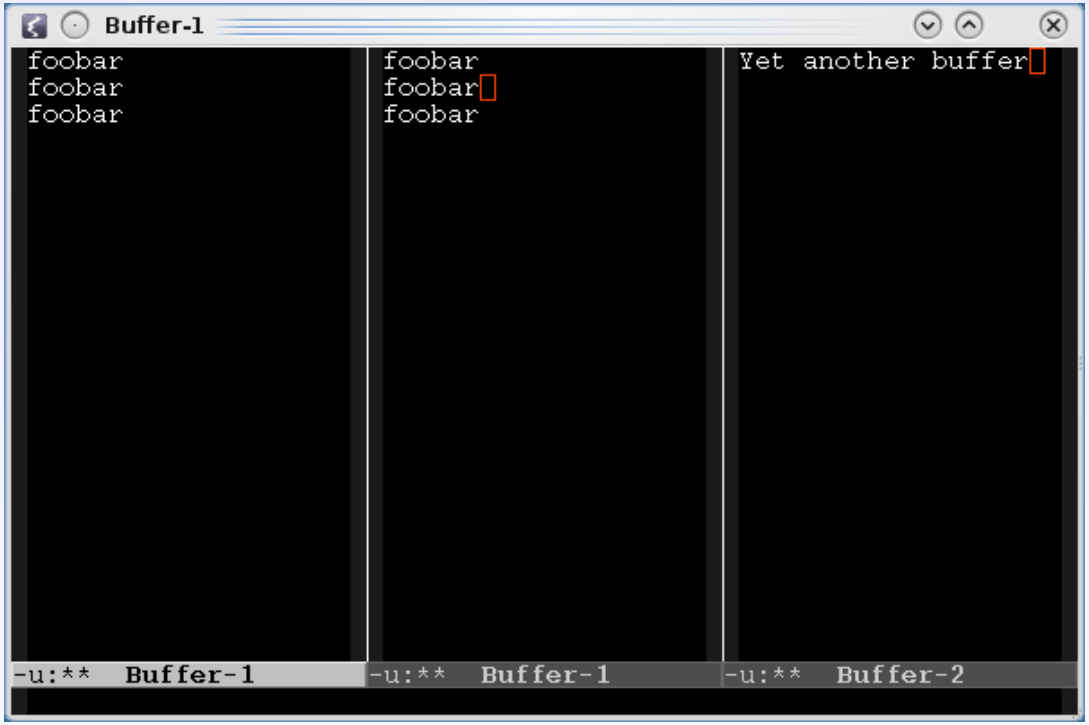

Figure: One emacs *frame*.

Some vocabulary.

- Frame: an emacs window, in the KDE sense of the term.
- Window: a cell in a split frame, with its *modeline*.

Figure: One emacs *frame* with three *windows*.

Some vocabulary.

- Frame: an emacs window, in the KDE sense of the term.
- Window: a cell in a split frame, with its *modeline*.
- Buffer: a text area that can be displayed in windows.
  - Not every buffer is necessarily visible at a given time.
  - A buffer may be displayed in several different windows.

Figure: One emacs *frame* with three *windows* and two *buffers*.

Some basics.

Emacs relies heavily on ELisp, a variety of Lisp.

- You can program virtually any Emacs behavior in ELisp.
- “Behaviors” are ELisp functions that are declared *interactive*.
- You can run any interactive function with: `M-x function-name`.
- Keyboard shortcuts in emacs are in fact just bound to interactive functions.

Use graphical emacs.

Whenever possible, use the graphic version.

Use graphical emacs.

Whenever possible, use the graphic version.

- It's *lighter*.
- It's graphically less limited (borders, fonts, colors, cursors, windows-manager interactions, . . .).
- It handles any kind of input. No conflicts with your shell or terminal.

Use graphical emacs.

Whenever possible, use the graphic version.

- It's *lighter*.
- It's graphically less limited (borders, fonts, colors, cursors, windows-manager interactions, . . .).
- It handles any kind of input. No conflicts with your shell or terminal.

Opposed arguments:

- When I paste with a middle click, it goes where the cursor is, not the mouse pointer.

Use graphical emacs.

Whenever possible, use the graphic version.

- It's *lighter*.
- It's graphically less limited (borders, fonts, colors, cursors, windows-manager interactions, . . .).
- It handles any kind of input. No conflicts with your shell or terminal.

Opposed arguments:

- When I paste with a middle click, it goes where the cursor is, not the mouse pointer.
  $\rightarrow$ Why the fuck are you doing with your mouse? Unplug it.

Use graphical emacs.

Whenever possible, use the graphic version.

- It's *lighter*.
- It's graphically less limited (borders, fonts, colors, cursors, windows-manager interactions, . . .).
- It handles any kind of input. No conflicts with your shell or terminal.

Opposed arguments:

- When I paste with a middle click, it goes where the cursor is, not the mouse pointer.
  $\longrightarrow$ Why the fuck are you doing with your mouse? Unplug it.
- When I'm forced to use the console version (over ssh for instance), I'm already used to it.

Stop closing emacs!

Every time you close and restart emacs for no reason, a kitten dies.

Stop closing emacs!

Every time you close and restart emacs for no reason, a kitten dies.

- It's slow.
- Emacs navigates through files better than your shell.
- You lose your current setup.
- Would you restart a fresh *Visual Studio* or *Eclipse* instance for every file?

Stop closing emacs!

Every time you close and restart emacs for no reason, a kitten dies.

- It's slow.
- Emacs navigates through files better than your shell.
- You lose your current setup.
- Would you restart a fresh *Visual Studio* or *Eclipse* instance for every file?

So:

- Start your emacs, and do not close it!
- Open new files with `C-x C-f` instead.
- Do not re-open files, switch buffers with `C-x b`.
- Use `ido-mode` to greatly improve files and buffer navigation.

Using Ido mode.

Try ido mode: `M-x ido-mode`. Once you're convinced, put it in your config file:

Using ido mode

```
; Turn Ido mode on.
(ido-mode)
; Use it to navigate files and buffers everywhere.
(ido-everywhere)
; Tab only completes and does not open files.
(setq ido-confirm-unique-completion t)
; Do not search files in other directories.
(setq ido-auto-merge-work-directories-length -1)
; Switch buffers with C-b for instance - shorter than C-x b.
(global-set-key [(control b)] 'ido-switch-buffer)
```

Split your Emacs.

Now that we keep our Emacs alive, we can tweak the window layout.

Split your Emacs.

Now that we keep our Emacs alive, we can tweak the window layout.

- `C-x 3` to split horizontally.
- `C-x 2` to split vertically.
- `C-x 1` to close all other windows.
- `C-x 0` to close this window.
- `C-x o` to switch window.

Using windmove.

Using `C-x o` to cycle through windows is a pain in the ass.

Using windmove.

Using `C-x o` to cycle through windows is a pain in the ass.

Using windmove to switch window.

```
; move to left  window
(global-set-key [M-left]  'windmove-left)
; move to right window
(global-set-key [M-right] 'windmove-right)
; move to upper window
(global-set-key [M-up]    'windmove-up)
; move to lower window
(global-set-key [M-down]  'windmove-down)
```

Now press `Meta + direction` to move through windows.
Cool, uh?

Outline

Compiling in emacs.

Compile directly in Emacs for maximum integration.

- Call `M-x compile`.
- Give it your compilation command.
- Later, use `M-x recompile`, unless you want to change the command.
- Make a shortcut for `recompile`, because we're lazy.

Compiling in emacs.

Compile directly in Emacs for maximum integration.

- Call `M-x compile`.
- Give it your compilation command.
- Later, use `M-x recompile`, unless you want to change the command.
- Make a shortcut for `recompile`, because we're lazy.

Emacs shows the compilation output in a separate window.

- Compile with a simple keystroke.
- No use to retype or search your compilation command.
- We have compilers messages right next to the code.
- The output is colored and much more readable
- No more confusion between compilations.

Compiling in emacs.

Compile directly in Emacs for maximum integration.

- Call `M-x compile`.
- Give it your compilation command.
- Later, use `M-x recompile`, unless you want to change the command.
- Make a shortcut for `recompile`, because we're lazy.

Emacs shows the compilation output in a separate window.

- Compile with a simple keystroke.
- No use to retype or search your compilation command.
- We have compilers messages right next to the code.
- The output is colored and much more readable
- No more confusion between compilations.
- `C-x backquote`: Emacs fucking navigates us directly to errors!

Compiling in emacs

Tweaking compilation

```
; Recompile with C-c c
(global-set-key [(control c) (c)] 'recompile)
; Visit errors with C-c e
(global-set-key [(control c) (e)] 'next-error)
; Set the compilation window height
(setq compilation-window-height 14)
; Scroll automatically to follow compilation
(setq compilation-scroll-output t)
```

Compiling in emacs

Tweaking compilation

```
; Recompile with C-c c
(global-set-key [(control c) (c)] 'recompile)
; Visit errors with C-c e
(global-set-key [(control c) (e)] 'next-error)
; Set the compilation window height
(setq compilation-window-height 14)
; Scroll automatically to follow compilation
(setq compilation-scroll-output t)
```

→ Okay, now we look more like engineers and less like monkeys.

Outline

Abuse your computer.

The computer works for you. Not the opposite.If you do any of these, there is a problem:

- Press more than say 4 times the same key or combination consecutively.
- Maintain some key pressed to repeat an action.
- Perform any kind of repetitive action.

Abuse your computer.

The computer works for you. Not the opposite.If you do any of these, there is a problem:

- Press more than say 4 times the same key or combination consecutively.
- Maintain some key pressed to repeat an action.
- Perform any kind of repetitive action.

Don’t forget that:

- The computer is your work slave. Enjoy, it’s legal.
- Leave all the dirty work to it.
- Don’t let your tool bug you: bend it to your will.

Move efficiently.

Learn to move quickly inside a buffer.

Move efficiently.

Learn to move quickly inside a buffer.

- Use control to change the meaning of directions:
  - `C-left` and `C-right`: move a word forward/backward.
  - `C-up` and `C-down`: move a paragraph up/down.

Move efficiently.

Learn to move quickly inside a buffer.

- Use control to change the meaning of directions:
  - `C-left` and `C-right`: move a word forward/backward.
  - `C-up` and `C-down`: move a paragraph up/down.
- Beginning of line: `C-a`.
- End of line: `C-e`.

Move efficiently.

Learn to move quickly inside a buffer.

- Use control to change the meaning of directions:
  - `C-left` and `C-right`: move a word forward/backward.
  - `C-up` and `C-down`: move a paragraph up/down.
- Beginning of line: `C-a`.
- End of line: `C-e`.
- Beginning of buffer: `M-<`.
- End of buffer: `M->`.

Move efficiently.

Learn to move quickly inside a buffer.

- Use control to change the meaning of directions:
  - `C-left` and `C-right`: move a word forward/backward.
  - `C-up` and `C-down`: move a paragraph up/down.
- Beginning of line: `C-a`.
- End of line: `C-e`.
- Beginning of buffer: `M-<`.
- End of buffer: `M->`.
- If you know what you're looking for, search with `C-s`.

Move efficiently.

Learn to move quickly inside a buffer.

- Use control to change the meaning of directions:
  - `C-left` and `C-right`: move a word forward/backward.
  - `C-up` and `C-down`: move a paragraph up/down.
- Beginning of line: `C-a`.
- End of line: `C-e`.
- Beginning of buffer: `M-<`.
- End of buffer: `M->`.
- If you know what you're looking for, search with `C-s`.
- If you know the line number, use `goto-line`. Bind it!

Move efficiently.

Learn to move quickly inside a buffer.

- Use control to change the meaning of directions:
  - `C-left` and `C-right`: move a word forward/backward.
  - `C-up` and `C-down`: move a paragraph up/down.
- Beginning of line: `C-a`.
- End of line: `C-e`.
- Beginning of buffer: `M-<`.
- End of buffer: `M->`.
- If you know what you're looking for, search with `C-s`.
- If you know the line number, use `goto-line`. Bind it!
- If you really wanna move `N` lines down, you can repeat most actions with a numeric `C-u` prefix: `C-u 4 2 down` moves 42 lines down.

Move efficiently.

Learn to move quickly inside a buffer.

- Use control to change the meaning of directions:
  - `C-left` and `C-right`: move a word forward/backward.
  - `C-up` and `C-down`: move a paragraph up/down.
- Beginning of line: `C-a`.
- End of line: `C-e`.
- Beginning of buffer: `M-<`.
- End of buffer: `M->`.
- If you know what you're looking for, search with `C-s`.
- If you know the line number, use `goto-line`. Bind it!
- If you really wanna move `N` lines down, you can repeat most actions with a numeric `C-u` prefix: `C-u 4 2 down` moves 42 lines down.
- Set the mark somewhere with `C-space`. Get back there with `C-u C-space`.

Edit efficiently.

Learn to move text around quickly.

Edit efficiently.

Learn to move text around quickly.

- Use the selection: `C-space` to set the mark.
- Kill and Paste: `C-w` and `C-y`.

Edit efficiently.

Learn to move text around quickly.

- Use the selection: `C-space` to set the mark.
- Kill and Paste: `C-w` and `C-y`.
- Become a professinal killer:
  - Kill entire word backward: `C-backspace`.
  - Kill entire word forward: `C-delete`.
  - Kill lines: `C-k`.

Edit efficiently.

Learn to move text around quickly.

- Use the selection: `C-space` to set the mark.
- Kill and Paste: `C-w` and `C-y`.
- Become a professinal killer:
  - Kill entire word backward: `C-backspace`.
  - Kill entire word forward: `C-delete`.
  - Kill lines: `C-k`.
- Use the kill ring: `M-y`.

Edit efficiently.

Learn to move text around quickly.

- Use the selection: `C-space` to set the mark.
- Kill and Paste: `C-w` and `C-y`.
- Become a professinal killer:
  - Kill entire word backward: `C-backspace`.
  - Kill entire word forward: `C-delete`.
  - Kill lines: `C-k`.
- Use the kill ring: `M-y`.
- Switch characters, perfect for typos: `C-t`
- Switch words: `M-t`.

Edit efficiently.

Learn to perform common modification automatically.

Edit efficiently.

Learn to perform common modification automatically.

- Don’t change case by hand:
  - Uppercase: `M-u`.
  - Lowercase: `M-l`.
  - Capitalize: `M-c`.

Edit efficiently.

Learn to perform common modification automatically.

- Don't change case by hand:
  - Uppercase: `M-u`.
  - Lowercase: `M-l`.
  - Capitalize: `M-c`.
- Squeeze spaces: `M-space`.
- Fuse lines: `M-^`.

Edit efficiently.

Learn to perform common modification automatically.

- Don't change case by hand:
  - Uppercase: `M-u`.
  - Lowercase: `M-l`.
  - Capitalize: `M-c`.
- Squeeze spaces: `M-space`.
- Fuse lines: `M-^`.
- Spell check: `M-$`.

  You need aspell

Edit efficiently.

Learn to perform common modification automatically.

- Don't change case by hand:
  - Uppercase: `M-u`.
  - Lowercase: `M-l`.
  - Capitalize: `M-c`.
- Squeeze spaces: `M-space`.
- Fuse lines: `M-^`.
- Spell check: `M-$`.

  You need aspell
- Sort region alphabetically: `M-x sort-lines`.

Edit C/C++ code efficiently.

Learn to inspect and manipulate C/C++ code.

Edit C/C++ code efficiently.

Learn to inspect and manipulate C/C++ code.

- Comment region: `C-c C-c`.
- Uncomment region: `C-u C-c C-c`.

Edit C/C++ code efficiently.

Learn to inspect and manipulate C/C++ code.

- Comment region: `C-c C-c`.
- Uncomment region: `C-u C-c C-c`.
- Highlight paired delimiters: `M-x show-paren-mode`

  Put it in your configuration file!

Edit C/C++ code efficiently.

Learn to inspect and manipulate C/C++ code.

- Comment region: `C-c C-c`.
- Uncomment region: `C-u C-c C-c`.
- Highlight paired delimiters: `M-x show-paren-mode`

  Put it in your configuration file!
- Count lines in region: `M-=`.

Edit C/C++ code efficiently.

Learn to inspect and manipulate C/C++ code.

- Comment region: `C-c C-c`.
- Uncomment region: `C-u C-c C-c`.
- Highlight paired delimiters: `M-x show-paren-mode`

  Put it in your configuration file!
- Count lines in region: `M-=`.
- Indent: `tab`.
- Indent region: `C-M-\`.

  If I catch anyone tapping `down` and `tab` to indent a piece of code ...

Edit C/C++ code efficiently.

Learn to inspect and manipulate C/C++ code.

- Comment region: `C-c C-c`.
- Uncomment region: `C-u C-c C-c`.
- Highlight paired delimiters: `M-x show-paren-mode`

  Put it in your configuration file!
- Count lines in region: `M-=`.
- Indent: `tab`.
- Indent region: `C-M-\`.

  If I catch anyone tapping `down` and `tab` to indent a piece of code ...
- Lexical completion: `M-/`.

Edit C/C++ code efficiently.

Learn to inspect and manipulate C/C++ code.

- Comment region: `C-c C-c`.
- Uncomment region: `C-u C-c C-c`.
- Highlight paired delimiters: `M-x show-paren-mode`

  Put it in your configuration file!
- Count lines in region: `M-=`.
- Indent: `tab`.
- Indent region: `C-M-\`.

  If I catch anyone tapping `down` and `tab` to indent a piece of code ...
- Lexical completion: `M-/`.
- Delete trailing whitespace: `M-x delete-trailing-whitespace`.

Edit C/C++ code efficiently.

If you're a warrior, you can design macros to insert common structures...

Edit C/C++ code efficiently.

If you're a warrior, you can design macros to insert common structures...
Otherwise, just re-use other's!

Some (very basic) macro to insert braces.

```
(defun c-insert-braces ()
  "Insert curly braces around the current line"
  (interactive)
  (beginning-of-line)
  (setq begin (point))
  (insert b "{\n")
  (end-of-line)
  (insert "\n}" a)
  (indent-region begin (point))
  (line-move -1)
  (end-of-line))))
; Bind it
(define-key c-mode-base-map
  [(control c) (control b)] 'c-insert-braces)
```

Search and replace

Search and replace code with isearch:

- Search: `isearch-forward`.
- Search regexp: `isearch-forward-regexp`.

Search and replace

Search and replace code with isearch:

- Search: `isearch-forward`.
- Search regexp: `isearch-forward-regexp`.
- Next occurrence: `isearch-repeat-forward`.
- Previous occurrence: `isearch-repeat-backward`.

Search and replace

Search and replace code with isearch:

- Search: `isearch-forward`.
- Search regexp: `isearch-forward-regexp`.
- Next occurrence: `isearch-repeat-forward`.
- Previous occurrence: `isearch-repeat-backward`.
- Yank word from buffer: `isearch-yank-word`.

Search and replace

Search and replace code with isearch:

- Search: `isearch-forward`.
- Search regexp: `isearch-forward-regexp`.
- Next occurrence: `isearch-repeat-forward`.
- Previous occurrence: `isearch-repeat-backward`.
- Yank word from buffer: `isearch-yank-word`.
- Switch to replace mode: `isearch-query-replace`.

Search and replace

Search and replace code with isearch:

- Search: `isearch-forward`.
- Search regexp: `isearch-forward-regexp`.
- Next occurrence: `isearch-repeat-forward`.
- Previous occurrence: `isearch-repeat-backward`.
- Yank word from buffer: `isearch-yank-word`.
- Switch to replace mode: `isearch-query-replace`.

You can make your own bindings.

Use shell commands.

Use shell commands to create and transform text.

Use shell commands.

Use shell commands to create and transform text.

- Execute shell command: `M-!`.
- Execute shell command and output in buffer: `C-u M-!`.

Use shell commands.

Use shell commands to create and transform text.

- Execute shell command: `M-!`.
- Execute shell command and output in buffer: `C-u M-!`.
- Pipe region through a shell command: `M-|`.
- Pipe region through a shell command and replace in buffer: `C-u M-|`.

Use shell commands.

Use shell commands to create and transform text.

- Execute shell command: `M-!`.
- Execute shell command and output in buffer: `C-u M-!`.
- Pipe region through a shell command: `M-|`.
- Pipe region through a shell command and replace in buffer: `C-u M-|`.

Unleash all the text-processing power of your Unix environment on you code!

Keyboard macros.

Meet the final boss: keyboard macros.
The idea is extremely simple, using it effectively requires some skill. But when you get it. . . you're unstoppable.

Keyboard macros.

Meet the final boss: keyboard macros.
The idea is extremely simple, using it effectively requires some skill. But when you get it. . . you're unstoppable.

- Hit `C-x (`.

Keyboard macros.

Meet the final boss: keyboard macros.
The idea is extremely simple, using it effectively requires some skill. But when you get it. . . you're unstoppable.

- Hit `C-x (`.
- Do whatever you want.

Keyboard macros.

Meet the final boss: keyboard macros.
The idea is extremely simple, using it effectively requires some skill. But when you get it. . . you're unstoppable.

- Hit `C-x (`.
- Do whatever you want.
- Hit `C-x )`.

Keyboard macros.

Meet the final boss: keyboard macros.
The idea is extremely simple, using it effectively requires some skill. But when you get it. . . you're unstoppable.

- Hit `C-x (`.
- Do whatever you want.
- Hit `C-x )`.
- Repeat with `C-x e`.

Keyboard macros.

Meet the final boss: keyboard macros.
The idea is extremely simple, using it effectively requires some skill. But when you get it. . . you're unstoppable.

- Hit `C-x (`.
- Do whatever you want.
- Hit `C-x )`.
- Repeat with `C-x e`.

Seamlessly automatize any text processing!

Conclusion. . .

Conclusion . . .

- 200.000 years of evolution.
- Two opposable thumbs

Honor evolution, make a decent use of your tools.

Conclusion. . .

Des questions, des choses pas claires?