Emacs

Edition de texte

Introduction

- est un logiciel permettant de saisir et de modifier du texte
- n'est pas un traitement de texte
- permet de manipuler des fichiers contenant du texte au sens large
- possède les fonctions classiques : déplacement dans les fichiers, recherche, positionnement, copier/couper/coller, ...
- s'interface avec le système : parcours de répertoires, lecture de courrier, compilation, etc
- possède de nombreuse bibliothèques et modes spécialisés
- environnement très extensible et hautement configurable ; programmation en lisp
- est un logiciel libre

Principales façons de lancer emacs en ligne de commande

- `emacs &` lance emacs seul en tâche de fond
- `emacs file &` emacs ouvre le fichier (si le fichier n'existe pas il le crée)
- `emacs dir/ &` emacs ouvre le répertoire et montre son contenu.

Lexique (1)

- **buffer** (tampon) : zone mémoire dans laquelle emacs copie le fichier
- **fenêtre** : écran dans lequel emacs affiche le fichier. Emacs est capable de diviser l'écran en plusieurs fenêtres
- chaque fenêtre est munie d'une **ligne de mode** : position du curseur, nom du buffer, buffer modifié, ...
- en bas de l'écran, se trouve le **mini-buffer** utilisé pour effectuer des opérations d'entrée/sortie, affiche des messages, ...
- le cycle d'édition classique : ouverture d'un fichier dans un buffer, édition (modification) du fichier, sauvegarde (il faut sauvegarder le plus souvent possible)

Dans la suite :

- `C` : Contrôle, touche 'CRTL'
- `M` : Méta, touche 'ESC'
- `C-x` : Appuyé sur 'CRTL' et 'x' simultanément

Lexique (2)

- **clé** : suite de caractères interprétée comme un tout. Certains caractères sont des **préfixes** et ne constituent pas des clés : `C-c, C-x, C-h, ESC`, i.e., ils attendent une suite dans la séquence de caractères.
- **commande** : toute action effectuée dans emacs est le résultat d'une commande (même l'insertion de caractère). De nombreuses commandes sont liées à des clés.
  Ex. : `forward-char C-f` ; `backward-char C-b` ; `save-buffers-kill-emacs C-x C-c`
- l'exécution d'une commande se fait via :
  - la clé associée (ex. : `C-f`),
  - le `mini-buffer` en tapant le nom de la commande (ex. `M-x forward-char`
- chaque commande, des plus simples aux plus complexes est définie par une fonction en `lisp`

Lexique (3)

- **liaison** : on peut modifier dynamiquement une liaison entre une clé et une commande (valables pour tous les buffers) via `global-set-key`.
  - `global-unset-key` pour annuler,
  - `C-h k: describe-key` pour vérifier l'existence,
  - `C-h b: describe-binding` pour avoir l'ensemble des clés.
- **macro** : suite de commandes clavier définie par l'utilisateur.
  - `C-x (` / `C-x )` : début/fin de définition.
  - `C-x e` : exécute la dernière macro définie.
  - `M-x name-last-kdb-macro` : permet de nommer la macro.

Quelques commandes/clés de base à connaître (1)

Commandes d'édition

- aller début/fin de ligne `C-a` / `C-e`
- aller à la ligne précédente/suivante/par numéro `C-p` / `C-n` / `M-g g`
- avancer/reculer d'un caractère `C-f` / `C-b`
- aller au début/à la fin d'un mot `M-b` / `M-f`
- sélectionner tout le buffer : `C-x h`
- copier/couper/coller/couper à partir du curseur : une sélection : `M-w` / `C-w` / `C-y` / `C-k`
- coller depuis le kill ring : `C-y`, puis `M-y` autant de fois que désiré pour parcourir l'anneau de ce qui a été coupé
- annuler l'action précédente : `C-_`
- `C-u n char` répète n fois char (n=4 par défaut, char peut être une clé)

Quelques commandes/clés de base à connaître (2)

Commandes sur les fichiers/buffers

- ouvrir un fichier : `C-x C-f`
- enregistrer un fichier : `C-x C-s`
- enregistrer sous : `C-x C-w`

Recherche/remplacement de texte

- recherche incrémentale en avant `C-s` et en arrière `C-r`
- remplacer un texte interactif `M-x query-replace`, `M-%`
- remplacer toute les occurrences `M-x replace-string`
- complétion d'un mot existant dans les buffers ayant le même préfixe `M-/`

Correction orthographique

- `M-x ispell-change-dictionary` pour modifier le dictionnaire
- `M-x flyspell-mode` analyse à la volée
- `M-x ispell-buffer` analyse le buffer

Personnalisation `.emacs`

- si un fichier de configuration `~/.emacs` se trouve à la racine du `home` alors à chaque lancement, le fichier de configuration est chargé et permet de configurer emacs.
- ce fichier contient une liste de commandes écrites en lisp.

Quelques exemples :

```
;; a comment
(set-background-color "blue")
(global-set-key (kbd "C-l") 'goto-line)
(global-set-key (kbd "C-c c") 'comment-region) ;; comment region
(global-set-key (kbd "C-c u") 'uncomment-region) ;; uncomment
(global-set-key (kbd "C-c i") 'indent-region) ;; indent
(transient-mark-mode t) ;; set region hilighted
(setq column-number-mode t) ;; column number
(setq line-number-mode t) ;; line number
(global-font-lock-mode t) ;; font-lock-mode
(setq font-lock-maximum-decoration t) ;; max decoration
(show-paren-mode t) ;; show parenthesis mode
(add-hook 'find-file-hook (lambda () (linum-mode 1))) ;; line numbering
(message "ok") ;; print ok
```

Emacs

Exercices

Tous les exercices sont à faire **SANS** la souris

Exercice 1 : Chargement d'un fichier

- exécuter la commande `find-file` à l'aide de sa clé. Emacs affiche dans le mini-buffer le message Find file : ~/
- essayer `esc esc esc`. Commenter
- relancer find-file
- essayer `tab tab`. Commenter.
- à l'aide de la complétion, allez chercher le fichier `/etc/passwd` et l'ouvrir
- remarquer dans la ligne de mode les symboles `%%`. Cela signifie quoi ?
- Fermer le buffer. Quelle est la commande à exécuter ? Quelle est sa clé ?

Exercice 2 : Manipulation de fenêtres

- ouvrir emacs en plein écran
- diviser l'écran verticalement en 2. Revenir à un seul écran
- diviser l'écran horizontalement en 2
- activer un shell dans emacs et lancer la commande `date`
- changer de buffer et créer un fichier dans votre `home` intituler `~/test-emacs.txt`
- changer de buffer et se mettre à la fin de la ligne de résultat de `date`
- venir en début de ligne et la couper
- changer de buffer et coller la ligne dans le fichier
- Remarquer les `**` dans la ligne de mode. Sauver Pourquoi les `**` ont disparues ? Sauver à nouveau. Commenter
- Quitter emacs. Répondre yes pour arrêter le `shell`

Exercice 3

Clés et couper/coller

- Ouvrir le fichier `test-emacs.txt` avec emacs : `emacs test-emacs.txt .`
- Se mettre en début de buffer.
- Lancer les commandes `kill-line` et `yank` en utilisant la complétion
- Essayer `C-h b`, une nouvelle fenêtre apparaît
- Se placer dans le nouveau buffer
- Rechercher la clé associée à `'kill-line'` et celle à `'yank'`
- recommencer le couper/coller à l'aide des clés ainsi trouvées

Enregistrer sous

- grâce à `C-h k` trouver la fonction associée à `C-x C-w`
- Sauver le fichier sous un autre nom, `test-emacs.txt.bak`, grâce à cette fonction
- Vérifier son existence dans le répertoire grâce à un `shell` dans emacs
- Supprimer le fichier depuis ce `shell` et vérifier que le fichier n'est plus là. Quitter le `shell`
- Ouvrir de nouveau l'ancien fichier `test-emacs.txt`

Exercice 4 : Damier

- effacer tout le contenu du fichier `test-emacs.txt`
- en un minimum de manipulation réaliser le damier de gauche puis celui de droite

```
XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX
    XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX
XXXX    XXXX    XXXX    XXXX
```

```
0000    0000    0000    0000
0000    0000    0000    0000
    0000    0000    0000    0000
    0000    0000    0000    0000
0000    0000    0000    0000
0000    0000    0000    0000
    0000    0000    0000    0000
    0000    0000    0000    0000
0000    0000    0000    0000
0000    0000    0000    0000
    0000    0000    0000    0000
    0000    0000    0000    0000
0000    0000    0000    0000
0000    0000    0000    0000
    0000    0000    0000    0000
    0000    0000    0000    0000
0000    0000    0000    0000
0000    0000    0000    0000
```

Exercice 4 : Macros

Macro

- Définir une macro permettant de placer les guillemets anglais (") autour du mot sur lequel est positionné le curseur.
- Tester avec le fichier `test-emacs.txt` et `C-x e`
- Nommer la macro en `insert-quote`. Tester avec `M-x insert-quote`
- Vérifier que la clé `M-"` est libre et la liée globalement à la macro `insert-quote`. Tester la liaison
- La commande n'est valable que pour la session en cours. Nous allons la sauvegarder dans le fichier `~/my-macros.el`
- Ouvrir un fichier nommé `my-macros.el` dans une nouvelle fenêtre
- Dans le fichier `my-macros.el` exécuter la commande
  `M-x insert-kbd-macro <return> insert-quote <return>`
  qui permet d'avoir la définition en lisp de la macro. Sauver le fichier.
- Terminer la session. Relancer emacs sur le fichier de test
- Charger le fichier de macro et tester la macro

GNU Emacs Reference Card

(for version 22)

Starting Emacs

To enter GNU Emacs 22, just type its name: `emacs`

Leaving Emacs

suspend Emacs (or iconify it under X)	`C-z`
exit Emacs permanently	`C-x C-c`

Files

read a file into Emacs	`C-x C-f`
save a file back to disk	`C-x C-s`
save **all** files	`C-x s`
insert contents of another file into this buffer	`C-x i`
replace this file with the file you really want	`C-x C-v`
write buffer to a specified file	`C-x C-w`
toggle read-only status of buffer	`C-x C-q`

Getting Help

The help system is simple. Type `C-h` (or `F1`) and follow the directions. If you are a first-time user, type `C-h t` for a **tutorial**.

remove help window	`C-x 1`
scroll help window	`C-M-v`
apropos: show commands matching a string	`C-h a`
describe the function a key runs	`C-h k`
describe a function	`C-h f`
get mode-specific information	`C-h m`

Error Recovery

abort partially typed or executing command	`C-g`
recover files lost by a system crash	`M-x recover-session`
undo an unwanted change	`C-x u`, `C-_` or `C-/`
restore a buffer to its original contents	`M-x revert-buffer`
redraw garbaged screen	`C-l`

Incremental Search

search forward	`C-s`
search backward	`C-r`
regular expression search	`C-M-s`
reverse regular expression search	`C-M-r`
select previous search string	`M-p`
select next later search string	`M-n`
exit incremental search	`RET`
undo effect of last character	`DEL`
abort current search	`C-g`

Use `C-s` or `C-r` again to repeat the search in either direction. If Emacs is still searching, `C-g` cancels only the part not done.

Motion

entity to move over	**backward**	**forward**
character	`C-b`	`C-f`
word	`M-b`	`M-f`
line	`C-p`	`C-n`
go to line beginning (or end)	`C-a`	`C-e`
sentence	`M-a`	`M-e`
paragraph	`M-{`	`M-}`
page	`C-x [`	`C-x ]`
sexp	`C-M-b`	`C-M-f`
function	`C-M-a`	`C-M-e`
go to buffer beginning (or end)	`M-<`	`M->`

scroll to next screen	`C-v`
scroll to previous screen	`M-v`
scroll left	`C-x <`
scroll right	`C-x >`
scroll current line to center of screen	`C-u C-l`

Killing and Deleting

entity to kill	**backward**	**forward**
character (delete, not kill)	`DEL`	`C-d`
word	`M-DEL`	`M-d`
line (to end of)	`M-0 C-k`	`C-k`
sentence	`C-x DEL`	`M-k`
sexp	`M-- C-M-k`	`C-M-k`

kill **region**	`C-w`
copy region to kill ring	`M-w`
kill through next occurrence of *char*	`M-z` *char*
yank back last thing killed	`C-y`
replace last yank with previous kill	`M-y`

Marking

set mark here	`C-@` or `C-SPC`
exchange point and mark	`C-x C-x`
set mark *arg* **words** away	`M-@`
mark **paragraph**	`M-h`
mark **page**	`C-x C-p`
mark **sexp**	`C-M-@`
mark **function**	`C-M-h`
mark entire **buffer**	`C-x h`

Query Replace

interactively replace a text string	`M-%`
using regular expressions	`M-x query-replace-regexp`

Valid responses in query-replace mode are

replace this one, go on to next	`SPC`
replace this one, don't move	`,`
skip to next without replacing	`DEL`
replace all remaining matches	`!`
back up to the previous match	`^`
exit query-replace	`RET`
enter recursive edit (`C-M-c` to exit)	`C-r`

Multiple Windows

When two commands are shown, the second is a similar command for a frame instead of a window.

delete all other windows	`C-x 1`	`C-x 5 1`
split window, above and below	`C-x 2`	`C-x 5 2`
delete this window	`C-x 0`	`C-x 5 0`
split window, side by side	`C-x 3`	
scroll other window	`C-M-v`	
switch cursor to another window	`C-x o`	`C-x 5 o`
select buffer in other window	`C-x 4 b`	`C-x 5 b`
display buffer in other window	`C-x 4 C-o`	`C-x 5 C-o`
find file in other window	`C-x 4 f`	`C-x 5 f`
find file read-only in other window	`C-x 4 r`	`C-x 5 r`
run Dired in other window	`C-x 4 d`	`C-x 5 d`
find tag in other window	`C-x 4 .`	`C-x 5 .`
grow window taller	`C-x ^`	
shrink window narrower	`C-x {`	
grow window wider	`C-x }`	

Formatting

indent current **line** (mode-dependent)	`TAB`
indent **region** (mode-dependent)	`C-M-\`
indent **sexp** (mode-dependent)	`C-M-q`
indent region rigidly *arg* columns	`C-x TAB`
insert newline after point	`C-o`
move rest of line vertically down	`C-M-o`
delete blank lines around point	`C-x C-o`
join line with previous (with arg, next)	`M-^`
delete all white space around point	`M-\`
put exactly one space at point	`M-SPC`
fill paragraph	`M-q`
set fill column	`C-x f`
set prefix each line starts with	`C-x .`
set face	`M-o`

Case Change

uppercase word	`M-u`
lowercase word	`M-l`
capitalize word	`M-c`
uppercase region	`C-x C-u`
lowercase region	`C-x C-l`

The Minibuffer

The following keys are defined in the minibuffer.

complete as much as possible	`TAB`
complete up to one word	`SPC`
complete and execute	`RET`
show possible completions	`?`
fetch previous minibuffer input	`M-p`
fetch later minibuffer input or default	`M-n`
regexp search backward through history	`M-r`
regexp search forward through history	`M-s`
abort command	`C-g`

Type `C-x ESC ESC` to edit and repeat the last command that used the minibuffer. Type `F10` to activate the menu bar using the minibuffer.

GNU Emacs Reference Card

Buffers

select another buffer	`C-x b`
list all buffers	`C-x C-b`
kill a buffer	`C-x k`

Transposing

transpose **characters**	`C-t`
transpose **words**	`M-t`
transpose **lines**	`C-x C-t`
transpose **sexps**	`C-M-t`

Spelling Check

check spelling of current word	`M-$`
check spelling of all words in region	`M-x ispell-region`
check spelling of entire buffer	`M-x ispell-buffer`

Tags

find a tag (a definition)	`M-.`
find next occurrence of tag	`C-u M-.`
specify a new tags file	`M-x visit-tags-table`
regexp search on all files in tags table	`M-x tags-search`
run query-replace on all the files	`M-x tags-query-replace`
continue last tags search or query-replace	`M-,`

Shells

execute a shell command	`M-!`
run a shell command on the region	`M-|`
filter region through a shell command	`C-u M-|`
start a shell in window `*shell*`	`M-x shell`

Rectangles

copy rectangle to register	`C-x r r`
kill rectangle	`C-x r k`
yank rectangle	`C-x r y`
open rectangle, shifting text right	`C-x r o`
blank out rectangle	`C-x r c`
prefix each line with a string	`C-x r t`

Abbrevs

add global abbrev	`C-x a g`
add mode-local abbrev	`C-x a l`
add global expansion for this abbrev	`C-x a i g`
add mode-local expansion for this abbrev	`C-x a i l`
explicitly expand abbrev	`C-x a e`
expand previous word dynamically	`M-/`

Regular Expressions

any single character except a newline	`.` (dot)
zero or more repeats	`*`
one or more repeats	`+`
zero or one repeat	`?`
quote regular expression special character *c*	`\`*c*
alternative (“or”)	`|`
grouping	`\( ... \)`
same text as *n*th group	`\`*n*
at word break	`\b`
not at word break	`\B`

entity	**match start**	**match end**
line	`^`	`$`
word	`\<`	`\>`
buffer	`\‘`	`\’`
class of characters	**match these**	**match others**
explicit set	`[ ... ]`	`[^ ... ]`
word-syntax character	`\w`	`\W`
character with syntax *c*	`\s`*c*	`\S`*c*

International Character Sets

specify principal language	`C-x RET l`
show all input methods	`M-x list-input-methods`
enable or disable input method	`C-\`
set coding system for next command	`C-x RET c`
show all coding systems	`M-x list-coding-systems`
choose preferred coding system	`M-x prefer-coding-system`

Info

enter the Info documentation reader	`C-h i`
find specified function or variable in Info	`C-h S`

Moving within a node:

scroll forward	`SPC`
scroll reverse	`DEL`
beginning of node	`.` (dot)

Moving between nodes:

next node	`n`
previous node	`p`
move **up**	`u`
select menu item by name	`m`
select *n*th menu item by number (1–9)	*n*
follow cross reference (return with `l`)	`f`
return to last node you saw	`l`
return to directory node	`d`
go to top node of Info file	`t`
go to any node by name	`g`

Other:

run Info **tutorial**	`h`
look up a subject in the indices	`i`
search nodes for regexp	`s`
quit Info	`q`

Registers

save region in register	`C-x r s`
insert register contents into buffer	`C-x r i`
save value of point in register	`C-x r SPC`
jump to point saved in register	`C-x r j`

Keyboard Macros

start defining a keyboard macro	`C-x (`
end keyboard macro definition	`C-x )`
execute last-defined keyboard macro	`C-x e`
append to last keyboard macro	`C-u C-x (`
name last keyboard macro	`M-x name-last-kbd-macro`
insert Lisp definition in buffer	`M-x insert-kbd-macro`

Commands Dealing with Emacs Lisp

eval **sexp** before point	`C-x C-e`
eval current **defun**	`C-M-x`
eval **region**	`M-x eval-region`
read and eval minibuffer	`M-:`
load from standard system directory	`M-x load-library`

Simple Customization

customize variables and faces	`M-x customize`

Making global key bindings in Emacs Lisp (examples):

```
(global-set-key "\C-cg" 'goto-line)
(global-set-key "\M-#" 'query-replace-regexp)
```

Writing Commands

```
(defun command-name (args)
  "documentation" (interactive "template")
  body)
```

An example:

```
(defun this-line-to-top-of-window (line)
  "Reposition line point is on to top of window.
With ARG, put point on line ARG."
  (interactive "P")
  (recenter (if (null line)
                0
              (prefix-numeric-value line))))
```

The `interactive` spec says how to read arguments interactively. Type `C-h f interactive` for more details.