

Environnement de développement sous Linux

Module 2I012-2018fev

Dominique Béréziat (Dominique.Bereziat@lip6.fr),
Valérie Ménissier-Morain



Première partie I

Introduction générale

Plan

Présentation et organisation de l'UE

Environnement de travail

Finalités de la Licence d'Informatique à Sorbonne Université

- ▶ c'est quoi un informaticien ?
 - ▶ connaissance du fonctionnement d'un ordinateur (archi, système, réseau)
 - ▶ maintenir le bon fonctionnement de la machine
 - ▶ créer de nouvelles fonctionnalités, automatiser les tâches, traiter une grande quantité de données
- ▶ devenir un informaticien compétent et autonome
 - ▶ utiliser les bons outils, les bons langages, les bonnes bibliothèques, ne pas réinventer la roue
 - ▶ s'auto-former
 - ▶ se tenir au courant des technologies
 - ▶ tester et débbugger
 - ▶ écrire des codes lisibles et documentés (pour soi-même et les autres)

Finalités de 2I012

- ▶ une part très importante des tâches de l'informaticien : écrire des fichiers textes \Rightarrow maîtrise d'un éditeur de texte puissant et **programmable** (~90% du temps de travail)
- ▶ maîtriser son environnement (ici Linux/Debian) :
 - ▶ savoir utiliser le terminal texte et son interpréteur de commandes
 - ▶ savoir utiliser les commandes unix principales et fondamentales (`grep`, `sed`, `cut`, `uniq`, `sort`, `head`, ...), les chaîner, les paramétrer, accéder à leur documentation
- ▶ répéter deux fois la même tâche, mais pas une troisième fois : les ordinateurs ont été créés dans ce but ! \Rightarrow savoir faire des scripts
- ▶ pour résoudre un problème posé, savoir le découper en tâches élémentaires, qui, si possible, peuvent être traitées par une commande unix ou une séquence simple de script
- ▶ savoir trouver et utiliser cette commande
- ▶ savoir écrire un *Makefile* pour automatiser la production de fichiers (compilation ou autre)
- ▶ faire la nique aux étudiants de 42 !

Ce que 2I012 n'apporte pas

- ▶ programmation (C ou autre) *malgré certains exemples en C ou autres*
- ▶ programmation système *malgré certains exemples en C*
- ▶ administration système *malgré certains exemples en bash*
- ▶ compilation (savoir écrire un compilateur),
- ▶ programmation réseau
- ▶ ...

Plan des 11 semaines d'enseignement

- ▶ Éditeur de textes : une 1/2 semaine
- ▶ Interprète de commandes : 7 1/2 semaines
 - ▶ commandes de base du système,
 - ▶ les scripts pour les combiner
 - ▶ les expressions régulières et les outils qui les manipulent
 - ▶ script avancés (programmation, structures de contrôle)
- ▶ Production automatisée, cohérente et optimale de fichiers : 3 semaines
 - ▶ la commande `make`
 - ▶ une introduction à la compilation séparée

Les deux premières semaines de TD/TME sont spéciales !

- ▶ TD+TME remplacé par TME+TME
- ▶ commandes de base d'Unix + éditeur de textes + documentation du système
- ▶ uniquement sur machines, ce sont les briques élémentaires de l'UE
- ▶ supposé être acquises pour la suite des TD/TME

Supports

- ▶ Starter Kit sur le site de la PPTI (notamment ssh et connexion distante)
- ▶ Site web de l'UE (sur Moodle !) : <https://tinyurl.com/y9ymvc5n>
 - ▶ version électronique des documents, dont le cours
 - ▶ Mémento `bash` écrit par Valérie Menissier-Morain (utile en TME et examen)
 - ▶ Refcard `emacs`
 - ▶ documentations diverses
 - ▶ les micro-sujets (correction automatique d'exercices)
 - ▶ les annales : elles sont complètes, corrigées, en ligne ou en version livre !
 - ▶ dépôt des devoirs et exercices

Calendrier

- ▶ 129 inscrits (au 19 janvier), 4 groupes quasi pleins (responsables de groupes : Valérie Menissier Morain, Christoph Lauter, Audrey Wilmet et Xavier Bonnetain)
- ▶ 11 cours (mercredi, 14h00-15h45, amphi 15/P1)
- ▶ 2 premières semaines de TME+TME (donc 2 fois 3h30 sur machine)
- ▶ 9 semaines de TD+TME (une modif à confirmer dans le calendrier)
- ▶ un TME solo avant le partiel pour la découverte
- ▶ un partiel sur machine la semaine du 12 mars (créneau du cours, à confirmer)
- ▶ un examen sur machine la semaine du 21 mai
- ▶ une session de rattrapage sur machine la semaine du 26 juin
- ▶ voir le planning officiel en ligne :
<http://www-licence.ufr-info-p6.jussieu.fr/lmd/licence/2017/edt/pdf/organisation-l2-s2-info.pdf>

Évaluation

- ▶ spécificité : essentiellement sur machine, attention aux personnes passives en binôme !
 - ▶ examens finaux sur machine : 60%
 - ▶ contrôle continu : 40%
- ▶ Contrôle continu :
 - ▶ interrogation sur les commandes de base en semaine 3 : 10%
 - ▶ TME solo (sur machine dans les conditions du partiel et examen) en semaine 5 : 10%
 - ▶ partiel en mars sur machine : 40%
 - ▶ exercices à rendre chaque semaine : 20%
 - ▶ devoir à la maison sur Makefile : 20%
- ▶ La règle du max ne s'applique qu'à la deuxième session !

Condition des examens sur machine

- ▶ seul sur une machine
- ▶ *homedir* vierge
- ▶ pas de /Vrac
- ▶ **pas d'internet**, ni même de communications locales
- ▶ téléphones rangés et éteints
- ▶ les logs des machines sont scrutés
- ▶ documents autorisés
- ▶ montage des clefs USB autorisée en **lecture seule** : pas de limitation sur le contenu des clés ni leurs nombres
- ▶ pas d'échange d'information entre étudiants

Stratégie gagnante

- ▶ venir en cours, et poser des questions !
- ▶ venir en TD et TME, travailler en binôme et être actif !
- ▶ venir en TD et TME connaissant son cours !
- ▶ refaire seul chez soi, et s'entraîner (lire ou copier-coller n'est pas comprendre)
- ▶ faire les micro-sujets !
- ▶ faire les devoirs
- ▶ faire les annales **DANS LES CONDITIONS DE L'EXAMEN** : temps imparti, pas d'internet, mais tout document autorisé.
- ▶ beaucoup de notions nouvelles : un travail **régulier** est nécessaire
- ▶ se tenir au courant (nouvelles fraîches) et utiliser et lire son webmail étudiant

Plan

Présentation et organisation de l'UE

Environnement de travail

Unix/Linux

Un système révolution, et une belle descendance

- ▶ **UNIX** : système d'exploitation multi-tâches et multi-utilisateurs créé en 1969, philosophie :
 - ▶ un interpréteur de commandes
 - ▶ de nombreux petit outils (commandes) commutables entre eux
 - ▶ un compilateur (language B), pour créer de nouvelles commandes
- ▶ Réellement diffusé (en dehors de Bell) en 1976 avec le système VI (le système a été réécrit en C, fraîchement inventé pour cette tâche)
- ▶ Base d'Internet avec le protocole TCP/IP (1982)
- ▶ En 2012 : famille d'environ 25 systèmes¹ dont les plus populaires sont **GNU/Linux**, BSD, OSX
- ▶ Depuis 1998 : **norme POSIX** pour certifier le fonctionnement et fonctionnalité d'un système d'exploitation (norme payante)
- ▶ Single UNIX specification : version libre (et augmentée) de **POSIX**

1. <https://www.youtube.com/watch?v=cN00SbyoHiQ>

La distribution Debian

- ▶ Linux : 1991, système d'exploitation compatible POSIX, réécriture opensource d'Unix + logiciels de la GNU Free Software Foundation
- ▶ **Distribution Linux au cours du temps**
- ▶ Debian : distribution communautaire fondée en 1993 par Ian Murdock. Elle est soutenue par la FSF
- ▶ Le chef de projet est élu par les membres de l'association
- ▶ Dernière version stable : 9.0 depuis juillet 2017
- ▶ Sert de support à de nombreuses autres distributions dont Ubuntu

Le système à la PPTI

Debian Jessie (8.8)

- Votre seul ami à partir d'aujourd'hui : le terminal texte !!!



FIGURE – Zone activités (cliquer en haut à gauche)

Trouver l'interpréteur



FIGURE – Zone de saisie (en haut à droite)

Trouver l'interpréteur

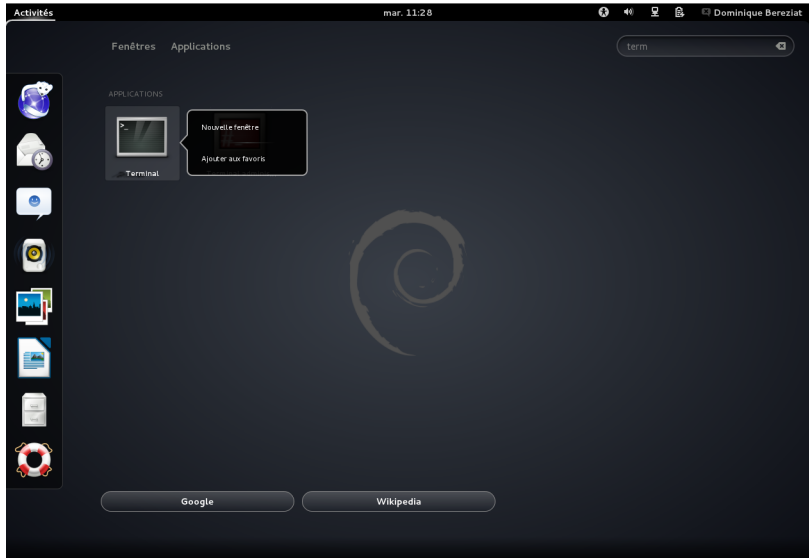


FIGURE – Clic droit (menu contextuel) ou tiré-lâché

Trouver l'interpréteur

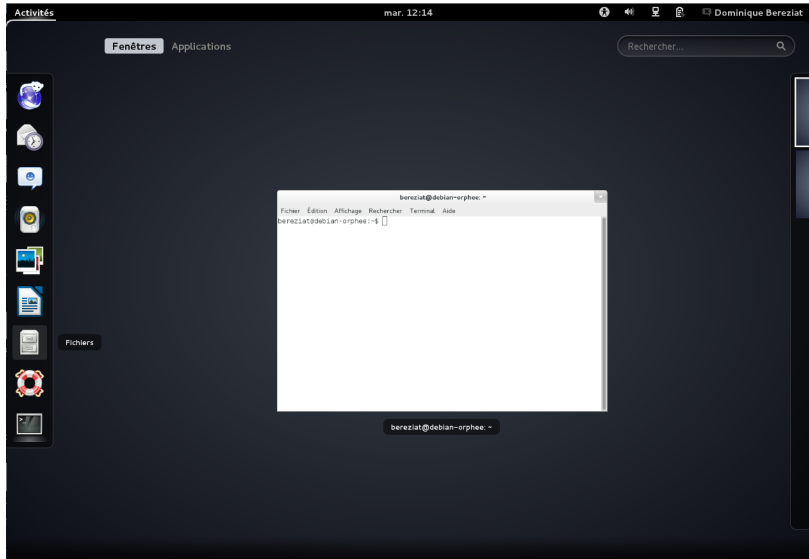


FIGURE – Lancement de l'interprète de commandes

Sur sa machine

- ▶ Avoir le même système que la PPTI (un PC-Linux), la AEIP6 organise des *install party* bonne solution pour les petits ordinateurs (Eeepc et autres)
- ▶ Autres systèmes (Windows, Mac OSX, BSD, ...) : une machine virtuelle (Virtual Box), et une image du système de la PPTI :
`https://www-ari.ufr-info-p6.jussieu.fr/index.php/ressources/environnement-virtuel` attention : 20 giga !
- ▶ Autre solution : installer Jessie from scratch pour Virtual Box (plus léger, plus rapide, on installe les commandes manquantes au fur et à mesure des besoins avec la commande `apt-get`). Détails ici :
`http://www-pequan.lip6.fr/~bereziatev/envdev/`
- ▶ Rester sous Mac OS ? Attention ! BSD et pas Unix, certaines commandes (`sed`, ...) sont différentes de GNU). Solution ici :
`http://www-pequan.lip6.fr/~bereziatev/envdev/`

Deuxième partie II

L'éditeur de textes Emacs

Plan

Introduction

Utilisation d'`emacs`

Utilisation d'`emacs` pour le développement

Configuration d'`emacs`

Nécessité d'un éditeur de texte puissant

- ▶ 90% du temps du développeur ou de l'administrateur est dans l'éditeur de texte :
 - ▶ écriture de code, de scripts
 - ▶ écriture de fichier de configuration
 - ▶ écriture de rapport
 - ▶ écriture de méls
 - ▶ ...
- ▶ outils d'édition puissant :
 - ▶ macros
 - ▶ copier-coller rapide
 - ▶ sélection/édition multiple
 - ▶ colorisation
 - ▶ tabulation/complétion automatique
- ▶ développement de code/script :
 - ▶ cycle d'écriture, de tests, de débogage, de correctifs
 - ▶ utilisation d'un outil de contrôle de version (`git`, non abordé dans ce cours) et surtout d'un éditeur puissant qui vérifie la syntaxe, lance des tests ou un débogueur ?
- ▶ maîtriser un éditeur de texte puissant permet de gagner un temps non négligeable dans la phase d'édition

État du marché (open source)

- ▶ Un grand nombre d'éditeurs de texte existent quelquesoit la plateforme (Linux, Mac, Windows) en open source, libre ou payant
voir <http://alternativeto.net/software/gnu-emacs/>
- ▶ Il y a aussi les IDE (Interactive Development Environment), Eclipse est open-source et disponible sur toute plateforme : nous n'en parlons pas dans ce cours. Ce sont des outils lourds mais utilisés dans l'industrie.
- ▶ Sur Linux les principaux sont `emacs` et `vi` (mais il en existe plein d'autres). Il existe des portages de ces éditeurs sur Mac et Windows.
- ▶ `emacs` a la préférence des scientifiques (ingénieur ou chercheur), `vi` a la préférence des développeurs ou des ingénieurs systèmes.

Emacs vs Vi

- ▶ Emacs fonctionne en terminal texte ou en fenêtrage X-Windows (et il existe un mode d'édition `vi`). Emacs n'est pas nécessairement installé par défaut dans les distributions Linux
- ▶ Vi ne fonctionne qu'en terminal texte, et est présent sous toutes les distributions Linux. Vim est une extension de `vi` (Vimproved) très largement utilisée et il existe un portage pour Gnome (`gvim`) pour permettre d'utiliser la souris
- ▶ Guerre des éditeurs (`emacs` vs `vim`) voir https://fr.wikipedia.org/wiki/Guerre_d'%C3%A9diteurs pour se faire son opinion, mais pour résumer :
 - ▶ `emacs` a une interface plus intuitive que `vi` et permet de débiter facilement, il est aussi peut-être plus puissant (un grand nombre d'extensions : mél, lecteur de news, brouteur web, ...)
 - ▶ `vim` est plus léger et démarre bien plus rapidement, `vi` est disponible sur tout système (POSIX)
 - ▶ cette guerre n'a plus lieu d'être : un bon *hacker* connaît, voire utilise, les deux
- ▶ Dans ce cours nous ne parlerons que d'`emacs`, les utilisateurs de `vi` sont autorisés à quitter le cours mais restent les bienvenus

Installation d'Emacs

- Sur une distribution de type Debian :

```
% sudo apt-get install emacs
```

- Sur Windows : <https://ftp.gnu.org/gnu/emacs/windows/> (non recommandé)
- Sur Mac : <https://emacsformacosx.com/> (non recommandé)
- non recommandé : `emacs` utilise le shell et surtout les commandes Unix de `/usr/bin`
- Sur Mac (recommandé) :

1. Installer Homebrew (http://brew.sh/index_fr.html) :

```
% /usr/bin/ruby -e "$(curl -fsSL \
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

2. Installer `emacs` via Homebrew :

```
% brew install emacs --with-cocoa --with-gnutls --with-imagemagick \
--with-d-bus --with-libsvg
```

3. Lancer `emacs` depuis le terminal texte permet d'avoir un comportement quasi identique à Linux.

Plan

Introduction

Utilisation d'emacs

Utilisation d'emacs pour le développement

Configuration d'emacs

Lancement

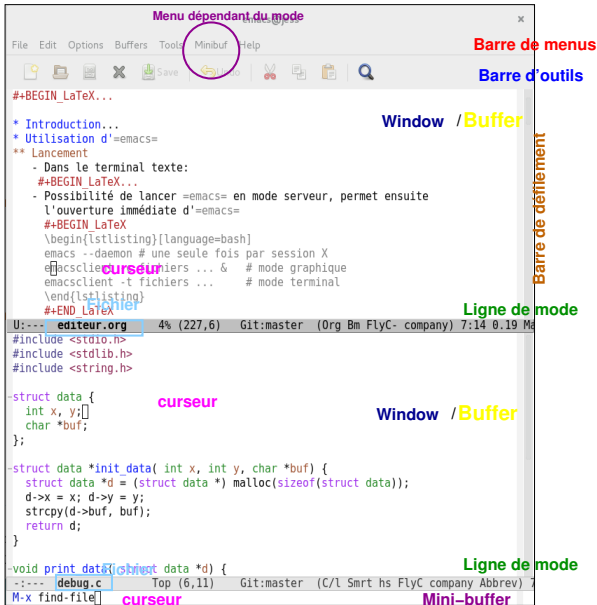
- Dans le terminal texte :

```
# mode terminal (no window)
emacs -nw fichier ... repertoires ...
# mode fenêtre graphique
emacs fichier ... repertoire ... &
export EDITOR='emacs -nw'
export VISIAL=emacs
```

- Possibilité de lancer `emacs` en mode serveur, permet ensuite l'ouverture immédiate d'`emacs`

```
emacs --daemon # une seule fois par session X
emacsclient -c fichiers ... & # mode graphique
emacsclient -t fichiers ... # mode terminal
```

emacs en mode graphique



Menu et raccourcis claviers

- ▶ En mode graphique : visiter et utiliser les menus. Il est plus efficace de connaître les commandes principales et leurs raccourcis
- ▶ Raccourcis clavier **essentiels** :
 - ▶ `C-x C-f` : charger un nouveau fichier
 - ▶ `C-x C-s` : sauver le fichier en cours d'édition
 - ▶ `C-g` : annuler une opération en cours
 - ▶ `C-x C-c` : quitter `emacs`
- ▶ Terminologie des raccourcis clavier :
 - ▶ `C-x` signifie que la touche `Control` et la touche `x` sont enfoncées simultanément
 - ▶ `C-x C-f` signifie `C-x` puis `C-f`, on peut laisser enfoncer `Control`.
`C-x` est un préfixe (il y en a d'autres)
 - ▶ `M-x` signifie "meta x", la touche `meta` vient des stations Unix (Sun et autre), sur un PC ou un Mac, c'est la touche `Alt`
 - ▶ `A-` est un synonyme de `M-`, `S-` désigne la touche `Shift`
 - ▶ La touche `ESC` remplace les touches `meta` ou `alternate` sur les claviers bizarres. Elle **ne se combine pas** avec les autres touches :
`ESC x` signifie `Escape` puis `x` et est équivalente à `M-x`

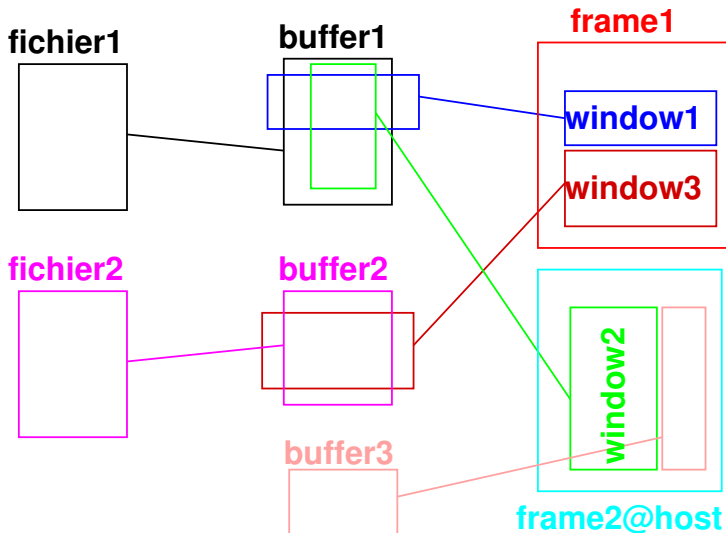
Commandes emacs

- ▶ Menu et raccourci clavier sont liés à des fonctions `emacs`
- ▶ Toute action dans `emacs` correspond à une fonction, par exemple appuyer sur la touche ordinaire telle que `a` correspond à l'appel de la fonction `self-insert-command`
- ▶ Certaines fonctions peuvent être appelées par l'utilisateur (elle sont dites *interactive*) et d'autres non
- ▶ Appel d'une fonction interactive : `M-x function` et les interactions ont lieu dans le mini-tampon
 - ▶ `M-x find-file` : charge un fichier dans un nouveau tampon. cette fonction est d'ailleurs lié (*binded*) au raccourci clavier `C-x C-f`
 - ▶ Certaines fonctions interactives attendent un paramètre (comme `find-file` qui attend un nom/chemin de fichier)
 - ▶ En mode interactif, `emacs` comprend la complétion (touche `TAB`)

Terminologie (suite)

- ▶ tampon (`buffer`) : c'est la représentation en mémoire d'un fichier (`file`) chargé et édité par `emacs`
- ▶ certains tampons sont spéciaux car ne sont pas liés à un fichier. Leurs noms commencent et finissent par `*`
- ▶ cadre (`frame`) : c'est le contenu d'une fenêtre X-window (ou Windows, ou MacOS selon le système). Une instance graphique d'`emacs` peut afficher plusieurs cadres
- ▶ `emacs` en mode terminal n'affiche qu'un seul cadre : celui du terminal
- ▶ fenêtre (`window`) : un cadre peut afficher une ou plusieurs zones de texte appelées fenêtres. Les fenêtres peuvent être rangées verticalement, horizontalement ou les deux si plus de 2 fenêtre
- ▶ une fenêtre affiche un tampon, si les tampons sont uniques, les fenêtres ne le sont pas
- ▶ barre de menu (`menu-bar`), barre d'outils (`tool-bar`)
- ▶ ligne de mode (ou barre d'état) (`modeline`)
- ▶ mini-tampon (`mini-buffer`) : zone de saisie des commandes (`M-x`) ou d'interaction entre `emacs` et l'utilisateur
- ▶ la souris, le(s) curseur(s)

Rapports géométriques

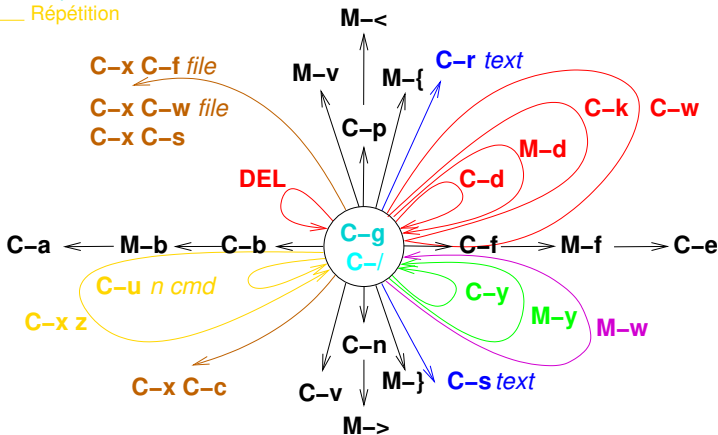


Sur les tampons et les fenêtres

- ▶ `emacs` travaille sur des tampons, il peut en avoir un grand nombre
- ▶ Gestion des tampons :
 - ▶ `C-x C-b` : ouvre un tampon contenant la liste des tampons et leurs caractéristiques
 - ▶ `C-x b` : pour afficher un autre tampon (`emacs` attend un nom, interaction dans le mini-tampon, la complétion fonctionne)
 - ▶ `C-x k` : tue le tampon courant (le supprime de la mémoire)
- ▶ Gestion des fenêtres :
 - ▶ `C-x o` : basculer d'une fenêtre à l'autre
 - ▶ `C-x 2` : scinder la fenêtre courante en deux fenêtres rangées verticalement
 - ▶ `C-x 2` : scinder la fenêtre courante en deux fenêtres rangées horizontalement
 - ▶ `C-x 1` : supprimer les autres fenêtres (autre que la fenêtre courante)
 - ▶ `C-x 0` : supprimer les autres sous-fenêtres

Édition et déplacement dans le tampon

- Déplacement
- Suppression
- Recherche
- Coller (du presse-papiers)
- Copier (dans le presse-papiers)
- Manipulation de buffers/fichiers
- Stop
- Undo
- Répétition



Copier-coller

► Raccourcis essentiels :

- `C-k` : copier et supprimer la portion de ligne à partir du curseur
- `C-SPC` : démarrer une sélection (utiliser les curseurs pour l'augmenter/la diminuer)
- `C-w` : copier la sélection et la supprimer
- `M-w` : copier la sélection sans la supprimer
- `C-y` : coller

► Possibilité de copier/coller à la Windows (menu Options, Use CUA)

► Copier-coller X-window : nécessite une souris à 2 boutons, pas pratique au *trackpad* :

- `button-1` (bouton de gauche) :
 - un simple clic ancre le début de la sélection, la fin est donnée par un simple clic de `button-3` (bouton de droite)
 - un appui prolongé sélectionne une région quelconque jusqu'au relâchement
 - un double clic sélectionne le mot courant
 - un triple sélectionne la ligne courante
 - un quadruple sélectionne la fenêtre entière
- `button-2` (bouton central, ou alors les deux boutons appuyés simultanément) colle la sélection à l'endroit où se trouve le pointeur de souris (et pas le curseur)

Macros (1)

Définition et application

- ▶ Une suite de commandes enregistrée que l'on peut réappliquer :
 - ▶ `F3` Commencer à enregistrer une nouvelle macro
 - ▶ `F4` Arrêter d'enregistrer la macro
 - ▶ `F3` pendant l'enregistrement de la macro : insère un compteur
 - ▶ `F4` après l'enregistrement de la macro : applique la dernière macro définie
- ▶ `C-u n` est un préfixe qui signifie "répéter n fois ce qui suit"
- ▶ `C-u n F4` : répéter n fois la dernière macro
- ▶ `C-u 0 F4` : répéter la dernière macro tant qu'il n'y a pas d'erreur
(`C-g` pour arrêter une boucle infinie)

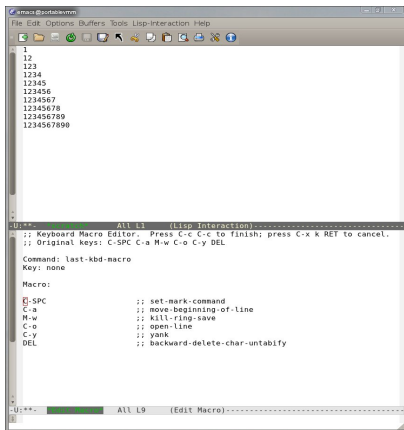
Macros (2)

Editer, enregistrer

- ▶ `C-x C-k RET` visualiser et modifier la dernière macro
- ▶ `C-x C-k b` associer un raccourci clavier à la dernière macro (alerte si le raccourci est déjà utilisé)
- ▶ `C-x C-k n nom` nomme la macro en *nom* : elle devient une fonction interactive que l'on peut appeler
- ▶ `M-x nom` appel de la macro *nom*
- ▶ `C-x C-k r` applique la macro à une région (une sélection)
- ▶ `C-x C-k C-h` liste l'ensemble des commandes commençant par `C-x C-k`
- ▶ `C-x C-k C-c n` initialise le compteur à *n*

Macros (3)

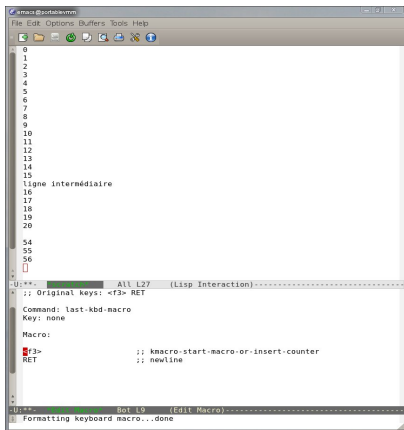
Exemple 1 : pyramide de chiffres créés à partir de la chaîne maximale



```
1234567890 ;; saisie de la chaîne maximale
<f3> ;; début d'enregistrement
  C-SPC ;; marqueur de début de sélection
  C-a ;; se déplacer au début de la ligne
  M-w ;; copier dans le presse-papier le texte
        ;; compris entre le point courant et le
        ;; début de sélection
  C-o ;; créer une nouvelle ligne avant la
        ;; position courante
  C-y ;; coller la sélection
  DEL ;; supprimer le dernier caractère
<f4> ;; fin d'enregistrement
<f4>
<f4>
<f4>
<f4> ;; on itère quelquefois et on obtient la
        ;; pyramide complète
<f4>
<f4>
<f4>
<f4>
;; ou bien
C-u 0 <f4> ;; on itère la macro tant que c'est
            ;; possible, il y a deux lignes vides
            ;; avant la pyramide, on les supprime.
```


Macros (4)

Exemple 2 : suite de nombres créée à partir de rien



```
<f3> ;; début d'enregistrement
<f3> ;; insertion d'un compteur,
      ;; affiche 0 sur la première ligne
RET ;; retour à la ligne
<f4> ;; fin d'enregistrement
C-u 15 <f4> ;; application 15 fois de la macro,
            ;; le compteur est substitué
            ;; par 1, puis 2, etc. jusqu'à 15
ligne intermédiaire ;; insertion de texte
C-u 5 <f4>      ;; application 5 fois de la macro,
                ;; le compteur est substitué par 16, etc
RET            ;; ligne blanche
C-x C-k C-c 54 ;; le compteur vaut 54
C-u 3 <f4>     ;; application 3 fois de la macro
```

Ligne d'état

- La ligne située au dessus du mini-tampon :

```
# set variable identifying the chroot you work in (used in the prompt below)
-:--- .bashrc      Top L4      (Shell-script[bash] company)
```

```
bereziat@orphee-vb:~$ emacs
bereziat@orphee-vb:~$ emacs test.el
```

- Elle indique de gauche à droite :
 - l'encodage Ascii, les drapeaux d'édition (lecture seule, tampon modifié)
 - le nom du tampon (fichier associé)
 - la position relative et la ligne courante du curseur
 - le mode majeur d'édition (ici script shell bash)
 - certains modes mineurs (ici la complétion automatique)
- La souris interagit avec cette ligne
- Cette ligne d'état est entièrement configurable. La mienne :

```
41:30 U -:ED:init-general.el      28% -master Emacs-Lisp b-loc fixm FlyC:0/4 - - Jeu jan 14 15:47 2.11
```

- Certains extensions sont dédiées à cette ligne

Documentation

- ▶ `emacs` est documenté :
 - ▶ un manuel complet au format hypertexte : `C-h r`
 - ▶ un tutorial : `C-h t`
 - ▶ une interface d'aide `C-h` (fonction, variable, combinaison de touches, ...)
 - ▶ la touche `F1` est relié à `C-h`
- ▶ le menu `Help` d'`emacs` permet d'accéder à d'autres informations, notamment un manuel de programmation `elisp`
- ▶ `C-h` interface de la documentation :
 - ▶ `C-h ?` info sur le système d'aide
 - ▶ `C-h k` info sur une combinaison du clavier
 - ▶ `C-h f` info sur une fonction
 - ▶ `C-h v` info sur une variable
 - ▶ `C-h a` (fonction `apropos`) affiche tout ce qui est connu du symbole entrée

Plan

Introduction

Utilisation d'`emacs`

Utilisation d'`emacs` pour le développement

Configuration d'`emacs`

Modes d'édition majeur

- ▶ Lorsqu'`emacs` charge un fichier, il choisit un mode d'édition (appelé mode majeur)
- ▶ Le mode est choisi en fonction de l'extension du fichier, ou de son *magic number*
- ▶ Le mode majeur affecte :
 - ▶ l'action du clavier (touches `TAB` ou `RETURN` par exemple)
 - ▶ le mode de colorisation
 - ▶ et ajoute de nouvelles fonctionnalités, certaines visibles dans le menu
- ▶ Le mode majeur peut être changé par l'utilisateur à la volée

Exemple en C et Shell

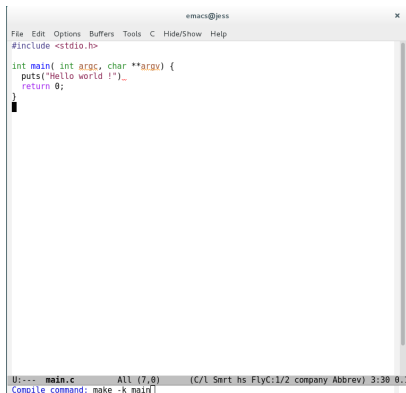
- ▶ Mode majeur C/C++ : `M-x c-mode`, `M-x c++-mode`
- ▶ Mode majeur Shell : `M-x sh-mode`
- ▶ Tabulation dans ces deux modes :
 - ▶ détermine de façon optimale le nombre de tabulations nécessaire
 - ▶ un mauvais placement de la tabulation peut indiquer un problème de syntaxe
- ▶ En C/C++ il existe plusieurs modes de tabulation `c-set-style`
- ▶ En `fortran`, `python` ou `Makefile` les tabulations sont critiques, `emacs` sera d'un grand secours.
- ▶ Un très grand nombre de langages sont supportés par `emacs`, même les plus obscures !

Modes d'édition mineur

- ▶ Un tampon possède toujours un mode majeur (`fundamental-mode` est le mode par défaut), et peut avoir plusieurs modes mineurs
- ▶ Un mode mineur peut être commun à tous les modes majeurs, et ils ajoutent des fonctionnalités que ni sont pas *a priori* en lien avec la nature du fichier édité.
- ▶ Par exemple, la colorisation est assurée par un mode mineur (`font-lock-mode`) ainsi, `M-x font-lock-mode` active/désactive ce mode mineur
- ▶ Les modes mineurs peuvent être affectés à tous les modes majeurs ou à certain.
- ▶ Quelques modes mineurs :
 - ▶ Le mode mineur `company` permet une complétion semi-automatique des mots clés, variables, nom de fichier dans le mode d'édition
 - ▶ Le mode mineur `flycheck` permet une vérification syntaxique à la volée
 - ▶ Le mode mineur `flyspell` active la correction orthographique à la volée
 - ▶ ...

Erreurs de syntaxe / Compilation

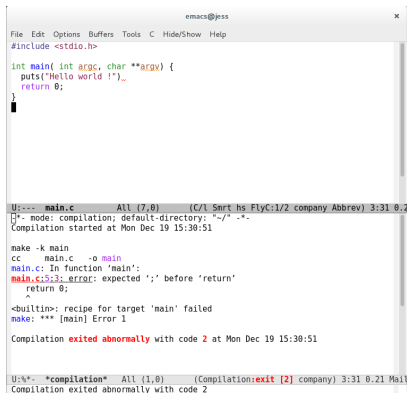
- `M-x compile` : appel à la commande `make`, erreurs de compilation sélectionnable
- le mode mineur `flycheck` avait aussi détecté le problème !



```
emacs@jess
File Edit Options Buffers Tools C Hide/Show Help
#include <stdio.h>

int main( int argc, char **argv) {
  puts("Hello world !");
  return 0;
}

U:--- main.c All (7,0) (C/l Smrt hs FlyC:1/2 company Abbrev) 3:30 0.1
Compile command: make -k main
```



```
emacs@jess
File Edit Options Buffers Tools C Hide/Show Help
#include <stdio.h>

int main( int argc, char **argv) {
  puts("Hello world !");
  return 0;
}

U:--- main.c All (7,0) (C/l Smrt hs FlyC:1/2 company Abbrev) 3:31 0.2
U:*** *compilation* All (1,0) (Compilation:exit [2] company) 3:31 0.21 Mail
Compilation exited abnormally with code 2
```


Erreurs de syntaxe / Compilation

Démonstration avec `premiersentiers.c`

```
#include <stdio>

int main(void)
{
    int somme, n_premiers_entiers, indice

    printf("calcule la somme des n premiers entiers, entrez n:");
    scanf("%d", &n_premiers_entiers);

    indice = 0;
    somme = 0;
    while (indice <= n_premiers_entiers; indice++)
        somme += indice;
}

printf("la somme est %s\n", &somme);

return 0;
}
```

Le débogueur

- ▶ `emacs` dispose d'une interface avec `gdb` (débogueur Gnu) : `M-x gdb`
- ▶ Commandes essentielles :
 - ▶ pose d'un point de contrôle : `C-c C-t` ou avec la souris dans la marge (*fringe*)
 - ▶ évaluation pas à pas : `step`, `next`
 - ▶ visualisation de la pile des appels `backtrace` et déplacement dans cette pile `up`, `down`
 - ▶ visualisation de variables ponctuellement : `print var`, à chaque instruction : `display var`, seulement quand elle change : `watch var`
 - ▶ lancement du programme : `run arguments ...`
 - ▶ `gdb` possède une aide intégrée : `help`
- ▶ `Menu/Gud/GDB-MI/Display other windows` permet d'obtenir l'affichage de la diapo suivante

Le débogueur

```
warning: Could not open OSO archive file "/BinaryCache/coreTLS/coreT
LS-35.40.1~1/Symbols/BuiltProducts/libcoretls_record.a"
warning: Could not open OSO archive file "/BinaryCache/coreTLS/coreT
LS-35.40.1~1/Symbols/BuiltProducts/libcoretls_stream_parser.a"

Breakpoint 1, main () at debug.c:22
22      struct data *d = init_data(1,2,"Hello world");
(gdb) n

Program received signal SIGSEGV, Segmentation fault.
0x00007fff902a8044 in _platform_memmove$VARIANT$Unknown () from /usr
/lib/system/libsystem.dylib
(gdb) 0

121: 6 ~-x-gud-a.out* Bot Debugger:run 1: 0 -R->locals of a.out* All Locals: _plat
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct data {
    int x, y;
    char *buf;
};

struct data *init_data( int x, int y, char *buf) {
    struct data *d = (struct data *) malloc(sizeof(struct data));
    d->x = x; d->y = y;
    strcpy(d->buf, buf);
    return d;
}

void print_data( struct data *d) {
    printf( stdout, "(%d,%d) %s\n", d->x,d->y,d->buf);
}

int main( void) {
    struct data *d = init_data(1,2,"Hello world");
    print_data(d);
    free(d);
    return 0;
}

10: 0 U ~/21012/Cours/Editeur2017/C/debug.c All C/L Server Sm 8: 0 -x->input/output of a.out* All Inferior I/O:
0 in _platform_memmove$VARIANT$Unknown of /usr/lib/system/libsystem.
1 in strcpy of /usr/lib/system/libsystem_c.dylib
2 in __strcpy_chk of /usr/lib/system/libsystem_c.dylib
3 in init_data of debug.c:13
4 in main of debug.c:22

Breakpoints Threads
Num Type Disp Enb Addr Hits What
1 breakpoint keep y 0x0000000100000f10 1 in main of debug.c: 2
21
```

Le débogueur

Démonstration avec `debug.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct data {
    int x, y;
    char *buf;
};

struct data *init_data( int x, int y, char *buf) {
    struct data *d = (struct data *) malloc(sizeof(struct data));
    d->x = x; d->y = y;
    strcpy(d->buf, buf);
    return d;
}

void print_data( struct data *d) {
    fprintf( stdout, "(%d,%d)_%s\n", d->x,d->y,d->buf);
}

int main( void) {
    struct data *d = init_data(1,2,"Hello_world");
    print_data(d);
    free(d);
    return 0;
}
```

Interface avec grep ou etags

- ▶ Dans un répertoire de fichiers sources, lancer la commande `etags fichiers` qui crée un fichier d'index (TAGS) des déclarations des fonctions, structures, variables globales, macros, ...
- ▶ Se placer sur un symbole dans un tampon et `C-x 4 .` vous emmène là où le fichier est défini (la première fois, demande de confirmation du fichier d'index)
- ▶ `man etags` pour une liste des langages supportés
- ▶ La commande `grep` est invocable depuis `emacs` : `M-x grep`. La sortie de `grep` est consignée dans un tampon `*grep*` dont les lignes sont cliquables et emmène là où `grep` a localisé le motif

Plan

Introduction

Utilisation d'`emacs`

Utilisation d'`emacs` pour le développement

Configuration d'`emacs`

Configuration d'emacs

- Au démarrage, emacs lit `~/.emacs/` ou `~/.emacs.d/init.el`
- Configuration écrite en Emacs Lisp (`elisp`), c'est un `lisp` enrichi
- Une instruction Lisp est une liste de mots délimitée par des parenthèses, le premier mot est le nom d'une fonction, les autres mots sont les arguments qui peuvent être eux-même des listes
- Configuration manuelle :

```
% emacs ~/.emacs
(tool-bar-mode 0)          ;; enlever la barre d'outil
(electric-pair-mode)      ;; insertion de paire de parentheses, accolade
                           ;; crochets, guillemets, ...
(setq inhibit-startup-screen t) ;; plus d'ecran de demarrage
(display-time)             ;; affiche l'heure dans la barre d'etat
(column-number-mode t)    ;; affiche la colonne courante
```

- Le menu `Options` permet quelques configurations basiques
- La fonction `customize` permet d'accéder à une interface interactive (`M-x customize`). Pour chercher un thème précis : `M-x customize-apropos` puis indiquer un nom, par exemple `printer`

Extensions emacs

- Les extensions `emacs` sont de simples fichiers `elisp`, elles sont centralisées sur des serveurs (dont un officiel) et installable à distance via le gestionnaire `package`
- Depuis `emacs` version 24. Dans la version 25, `package` gère plus efficacement les dépendances
- Ajouter le dépôt Melpa (plus riche que le dépôt officiel Elpa) dans le fichier `~/.emacs` :

```
(require 'package)
(add-to-list 'package-archives
  '("melpa" . "http://melpa.org/packages/") t)
```

- Voir Awesome Emacs,
<https://github.com/emacs-tw/awesome-emacs>, qui fait un éventaire d'extensions `emacs` très utiles

Étendre emacs (chez soi ou à la PPTI)

► Usage de package :

- `M-x list-package` : liste les extensions disponibles et ceux qui sont installés
- dans le tampon créé par `package` positionner le curseur sur une extension, par exemple, `company`, puis faire `i` (`Install`) puis `x` (`eXecute`) : `company` est installé

► Configuration emacs utile pour 2I022 sur mon github :

```
% cd
% mv .emacs .emacs-old
% git clone https://github.com/bereziat/2I012-emacs \
  .emacs.d
```

et installe et active :

- la complétion automatique
- la vérification syntaxique à la volée
- le répliage/dépliage des fonctions et commentaires (*folding*)
- et d'autres modes encore
- les contributions de chacun sont les bienvenues (*pull request*) !