Guide de Survie pour `emacs`, `make`, `cvs`, `latex` et `ocaml`*

Les manuels d'`emacs`, `make` et `cvs` sont disponibles chez GNU à l'adresse :

`http ://www.gnu.org/manual/`

De nombreux documents sur `latex` sont disponibles à l'adresse :

`http ://tex.loria.fr/`

La documentation des outils de développement d'Ocaml se trouve à l'adresse :

`http ://caml.inria.fr/`

Ces documentations sont à consulter en ligne – ne pas les imprimer SVP

1 Commandes de base pour l'éditeur `emacs`

Lancer et quitter emacs

`emacs&`	Lancer l'éditeur en arrière plan
`ctrl-x ctrl-c`	Sortir de l'éditeur

Fichiers

`ctrl-x ctrl-f`	Charger (éditer) un fichier dans un tampon
`ctrl-x ctrl-s`	Sauver un fichier (tampon courant) sur disque
`ctrl-x i`	Insérer le contenu d'un autre fichier dans le tampon courant

Tampons

`ctrl-x b`	Choisir un autre tampon
`ctrl-x k`	Détruire un tampon

Couper, copier et coller

`ctrl-SPC`	Poser une marque
`ctrl-w`	Couper de la marque au curseur
`ctrl-k`	Couper du curseur à la fin de la ligne
`Meta-w`	Copier de la marque au curseur
`ctrl-y`	Coller le dernier élément collé ou copié
`Meta-y`	Après `ctrl-y` récupère les derniers éléments coupés/collés

Déplacement et Multi-fenêtrage

`ctrl-a`	Déplacer le curseur au début de la ligne
`ctrl-e`	Déplacer le curseur à la fin de la ligne
`ctrl-x 0`	Fermer la fenêtre courante
`ctrl-x 1`	Ne garder qu'une seule fenêtre à l'écran
`ctrl-x 2`	Couper la fenêtre en deux verticalement
`ctrl-x 3`	Couper la fenêtre en deux horizontalement

Le mode Tuareg d'`ocaml`

`f5`	Compiler
`f6`	Recompiler
`f7`	Positionner le curseur sur la prochaine erreur
`ctrl-x ctrl-e`	Évaluer la phrase sous le curseur
`ctrl-c ctrl-b`	Évaluer le tampon
`ctrl-c ctrl-r`	Évaluer de la marque au curseur
`ctrl-c t,m,l,i,w,f,b`	Insérer les formes `try`, `match`, `let`, `if`, `while`, `for` ou `begin`

Autres commandes utiles

`ctrl-g`	Quand ça va mal !
`ctrl-_`	Annuler la dernière commande
`ctrl-s`	Rechercher un motif (`ctrl-g` arrête la recherche)
`Meta-%`	Remplacer un motif par un autre
`Meta-x describe-mode`	Pour trouver toutes les infos sur le mode courant d'ocaml

2 Gestion de projets : `make`

Forme des règles du fichier `Makefile`

`cible :`	*dépendances*
`<tab>`	*actions*

Exécuter un `Makefile`

`make cible`	La première cible est exécutée par défaut

Macros

`NOM=VALEUR`	Pour déclarer une nouvelle macro
`$(NOM)`	Pour utiliser la macro NOM
`$@`	Nom de la cible à reconstruire
`$<`	Nom de la dépendance à partir de laquelle la cible est reconstruite
`$^`	Liste de toutes les dépendances

Insertion de makefiles externes

include <file>	Pour inclure le fichier <file> dans votre Makefile

Règles des suffixes

.SUFFIXES : .ml .cmo	Pour ajouter des suffixes
.ml.cmo : ocamlc -c `$<` -o `$@`	Règle implicite pour construire un fichier .cmo à partir d'un fichier .ml. La macro `$<` représente le fichier d'entrée et `$@` celui de sortie.

Commentaires

# commentaire	Le commentaire se termine à la fin de la ligne

3 Gestion de versions : cvs

3.1 Côté serveur

Création du *Repository*

mkdir CHEZMOI/cvs	*Attention* CHEZMOI doit être un chemin absolu !
cvs -d CHEZMOI/cvs init	Initialisation du nouveau *repository* cvs

Importation d'un projet

cd ~nom/projet	Les fichiers sources de votre projet de compilation sont ici
cvs -d CHEZMOI/cvs import projet nom new	Pour créer une nouvelle entrée dans le *repository* cvs et y importer les fichiers du répertoire ~nom/projet Après cette commande, vous pouvez effacer le répertoire contenant les sources (~nom/projet).

3.2 Côté client

cvs -d CHEZMOI/cvs co projet	Pour créer un répertoire de travail projet contenant une copie de l'entrée projet du *repository* cvs

Vous devez vous trouver dans votre répertoire de travail pour les commandes suivantes.

Synchronisation avec le *repository*

`cvs up`	Mise à jour du répertoire de travail

Ajouter un fichier au *repository*

`cvs add fichier`	Pour ajouter un fichier texte ou un répertoire
`cvs add -kb fichier`	Pour ajouter un fichier binaire

Supprimer un fichier du *repository*

`rm fichier`	Supprimer le fichier de votre répertoire de travail
`cvs remove fichier`	Puis le supprimer du *repository*

Validation des changements

`cvs commit`	Après une modification d'un fichier

4 L'outil de mise en page `latex`

Structure du document

```
\documentclass[a4paper,12pt]{article}
\usepackage[T1]{fontenc}
\usepackage[latin1]{inputenc}
\usepackage[french]{babel}

\begin{document}

    Placer le contenu de votre document ici

\end{document}
```

Sections

```
\part{titre}
 \section{titre}
  \subsection{titre}
   \paragraph{titre}
    \subparagraph{titre}
```

Listes, Énumérations et Descriptions

Listes	Énumérations	Descriptions

`\begin{itemize}`
`  \item ...`
`   ⋮`
`  \item ...`
`\end{itemize}`
	`\begin{enumerate}`
	`  \item ...`
	`   ⋮`
	`  \item ...`
	`\end{enumerate}`
		`\begin{description}`
		`  \item[Élement1] ...`
		`   ⋮`
		`  \item[Élement2] ...`
		`\end{description}`

Centrer un texte

```
\begin{center}
   ...                ou   \centerline{...}
\end{center}
```

Souligner l'importance d'un mot

`\emph{`*texte*`}`	Pour mettre un texte en italique
`\textbf{`*texte*`}`	Pour mettre un texte en gras

5 Les outils de développement `ocaml*`

Extensions des fichiers

`.ml`	fichier source
`.mll`	fichier source `ocamllex`
`.mly`	fichier source `ocamlyacc`
`.mli`	fichier interface
`.cmo`	fichier objet (bytecode)
`.cma`	fichier d'une bibliothèque objet (bytecode)
`.cmi`	fichier interface compilé
`.cmx`	fichier objet (natif)
`.cmxa`	fichier d'une bibliothèque objet (natif)

Les options du compilateur `ocamlc`

`-c`	compile sans faire l'édition de liens
`-o` *fichier*	spécifie le nom du *fichier* exécutable
`-i`	affiche toutes les déclarations globales compilées
`-pp camlp4o`	utilise camlp4 comme pré-processeur
`-g`	ajout d'informations de mise au point
`-v`	affiche la version du compilateur

Directives de la boucle d'interaction `ocaml`

Attention ! Toutes les directives doivent être précédées du symbole `#`.

`#quit ;;`	sort de la boucle d'interaction
`#directory "dir" ;;`	ajoute le répertoire `dir` dans la liste des chemin d'accès aux fichiers sources et compilés
`#load "file-name" ;;`	charge un fichier `.cmo` en mémoire
`#use "file-name" ;;`	compile et charge un fichier source en mémoire
`#trace f ;;`	trace les arguments de la fonction `f`
`#untrace f ;;`	enlève la trace de la fonction `f`
`#untrace_all ;;`	enlève toutes les traces

L'outil d'analyse lexicale `ocamllex`

`ocamllex` *lexer*`.mll` produit dans *lexer*`.ml` le code Ocaml d'un analyseur lexical. Le format d'un fichier `.mll` est le suivant :

```
{ prologue }
let ident = regexp ...
rule entrypoint =
  parse regexp { action }
    | ...
    | regexp { action }
and entrypoint =
  parse ...
and ...
{ epilogue }
```

Expressions régulières	
`'c'`	le caractère `c`
`_`	n'importe quel caractère
`eof`	fin du fichier
`"abc"`	la chaîne de caractères `abc`
`[` *ens* `]`	ensemble de caractères ; les ensembles *ens* valident sont `'c'`, `'a' - 'z'` et `'a' 'b'`
`[ ^` *ens* `]`	tous les caractères excepté ceux appartenant à *ens*
`regexp *`	répétition
`regexp +`	répétition stricte
`regexp ?`	la chaîne vide ou `regexp`
`regexp1 | regexp2`	alternative
`regexp1 regexp2`	concaténation

Actions	
`Lexing.lexeme lexbuf`	retourne la chaîne reconnue
`Lexing.lexeme_start lexbuf`	retourne la position absolue dans le texte d'entrée du *début* de la chaîne reconnue
`Lexing.lexeme_end lexbuf`	retourne la position absolue dans le texte d'entrée de la *fin* de la chaîne reconnue
`entrypoint lexbuf`	Rappel le *lexer* sur le point d'entrée `entrypoint`

L'outil d'analyse syntaxique `ocamlyacc`

`ocamlyacc` *parser*`.mly` produit dans *parser*`.ml` le code Ocaml d'un analyseur syntaxique

Le format d'un fichier `.mly` est le suivant :

```
%{
   prologue
}%
   declarations
%%
   regles
%%
   epilogue
```

Le format des règles est :

```
nonterminal :
   symbol ... symbol { action }
  | ...
  | symbol ... symbol { action }
;
```

Déclarations	
`%token T1 ... Tn`	déclare les *tokens* `T1`...`Tn`
`%token` <*type*> `T1 ... Tn`	déclare les *tokens* `T1`...`Tn` avec attributs de type *type*
`%start nt1 ... ntn`	déclare les non-terminaux `nt1`...`ntn` comme points d'entrée de la grammaire
`%type` <*type*> `nt1 ... ntn`	spécifie le type des attributs des non-terminaux `nt1`...`ntn`
`%left T1 ... Tn`	déclare les symboles `T1`...`Tn` associatifs à *gauche* (utilisez `%right` pour une associativité à *droite* et `%nonassoc` pour les symboles *non-associatifs*). Tous les *symboles* d'une même ligne ont la même *priorité*. Ils ont une priorité plus grande que les symboles déclarés avant et plus faible que ceux déclarés après. La directive `%prec T` est utilisée dans une règle pour modifier la priorité d'une production (par défaut, il s'agit de la priorité du terminal – s'il existe – apparaissant le plus à droite dans la production)

L'outil de documentation `ocamlweb`

L'outil `ocamlweb` vous permet de produire un document au format LaTeX (et donc ensuite aux formats DVI et PostScript) à partir d'un code source Ocaml. Les commentaires y sont interprétés comme étant écrit en LaTeX, et peuvent contenir des références à du source Ocaml entre crochets. On pourra ainsi écrire :

```
(* La fonction [power] ci-dessous implante la puissance
   entière de manière efficace. [power x n] calcule $x^n$, en
   supposant $x \ge 0$. *)

let rec power x n =
  if n = 0 then
    1
  else
    let y = power x (n / 2) in
    if n mod 2 = 0 then y * y else x * y * y
```

Pour mettre réellement du code entre commentaires, on utilise des commentaires de la forme `(*i` ...`i*)`. On utilise `ocamlweb` de la manière suivante :

```
ocamlweb -o fichier.tex fichier.ml ...
```

Faire `ocamlweb -h` pour obtenir la liste complète des options. Sur l'exemple ci-dessus, cela donne :

La fonction *power* ci-dessous implante la puissance entière de manière efficace. *power x n* calcule x^n, en supposant $x \geq 0$.

let rec $power\ x\ n =$
  if $n = 0$ then
    1
  else
    let $y = power\ x\ (n\ /\ 2)$ in
    if n mod $2 = 0$ then $y \times y$ else $x \times y \times y$
