Université Paris XIII — Année 2006-2007
Institut Galilée
Formation MACS - 2ème année

INTRODUCTION À UNIX
Feuille n° 3

1 HTML et LaTeX

HTML et LaTeXsont des languages permettant de formatter des documents. Des mini-guides de LaTeXet HTML sont fournis en annexe à ce TD. Une autre bonne source d'information est la FAQ LaTeXen français :

`http ://www.grappa.univ-lille3.fr/FAQ-LaTeX`

1. HTML
   - Recopier le fichier `html_modele.html` sur votre compte et le visualiser avec un vigateur web en tapant l'adresse :
     `file:/chemin_vers_mon_compte/html_modele.html`
   - Visualisez le fichier à l'aide de l'éditeur xemacs puis modifier le fichier pour taper votre CV.
2. LaTeXen mode texte
   - Recopier le fichier `latex_modele.tex` sur votre compte et le compiler à l'aide de la commande suivante :
     `latex latex_modele.tex`
   - Visualisez le fichier le fichier `.dvi` à l'aide de la commande
     `xdvi latex_modele.dvi`
   - Utiliser l'éditeur xemacs pour modifier le fichier puis taper votre CV.
3. LaTeXen mode mathématique
   Reproduire les expressions suivantes :

$$\frac{\alpha^n}{\alpha^m} = \alpha^{n-m},$$

$$\sum_{i=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6},$$

$$\int_{x=0}^{+\infty} \frac{log^n\, x}{x^m} dx,$$

$$\lim_{x \to +\infty} \frac{log\, x}{x} = 0,$$

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

$$f(x_1, x_2, \ldots, x_n) = x_1^2 + x_2^2 + \cdots + x_n^2,$$

$$\sqrt[3]{q + \sqrt{q^2 - p^3}} + \sqrt[3]{q - \sqrt{q^2 - p^3}},$$

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2},$$

$$\begin{pmatrix} x_{11} & \cdots & x_{1p} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{np} \end{pmatrix}.$$

La matrice $b^{(k)}$ est définie par (1) :

$$b^{(k)} = \begin{pmatrix} a_{11}^{(k)^T} & a_{21}^{(k)^T}(1 : (n-k-1) \cdot m) & 0_{m,nrhs} \\ a_{12}^{(k)^T} & a_{22}^{(k)^T}(1 : (n-k-1) \cdot m) & 0_{m,nrhs} \\ 0_{nrhs,m} & 0_{nrhs,(n-k-1) \cdot m} & I_{nrhs} \end{pmatrix} \quad (1)$$

2 Gestion de programme `make, ar, ranlib, cvs ...`

Copier le fichier tp3.tgz. Décompresser et détarer le fichier.

2.1 Compilation d'un grand programme : make

Lorsqu'on développe un programme (par exemple prod_fbls.f), il y a souvent plusieurs sous programmes (ici tous les autres .f) écrits dans de multiples fichiers. Ce programme et ces sous programmes peuvent également inclure des fichiers .h (cdim.h et cfscal.h). La phase de compilation consiste à générer les objets (les .o) et la phase d'édition de liens à créer un exécutable (prod_fbls.exe).
Ces deux étapes ne doivent pas être faites systématiquement de manière complète. En effet, si l'on modifie seulement l'un des sous programmes, il suffit de regénérer le .o correspondant et de refaire l'édition de lien.
De plus, il peut exister de multiples dépendances dans les codes source via les fichiers include (les .h)...

La commande make, en se basant sur les dates de modification des fichiers, automatise ce travail complexe de gestion de la compilation.
Cette commande appelle un fichier Makefile, dans lequel doivent être listées les règles de dépendance, de compilation, les noms de fichier et les options de compilation nécessaires.
Sachant que le programme principal (fichier prod_fbls.f) fait appel aux routines `ddot, dger, daxpy, dgemv ,lsame, xerbla`*, écrire un Makefile permettant de créer l'exécutable prod_fbls.exe.*

Remarque : man make fournit une description précise de la commande make.

2.2 Structurer un code en créant des librairies : ar et ranlib

Dans un projet, il est sain de séparer dans des répertoires différents les sources (fichiers .f ou .c) les exécutables et les résultats d'exécution du programme. On peut par exemple créer des répertoires

src/ bin/ run/

où stocker respectivement les sources, les exécutables (fichiers binaires) et les résultats d'exécution (run).
Créer ces trois répertoires dans TP3/ et déplacer tous les fichiers .f et .h dans src/.
De plus, quand on a beaucoup de fichiers il faut à l'interieur des sources créer des sous-répertoires pour classer et regrouper les routines.
Dans notre exemple, créer un sous répertoire blas/ qui regroupe toutes les routines d'algèbre linéaire (Basic Linear Algebra Subroutine), c'est à dire tous les .f autres que prod_fbls.f.
Les .o correspondant (qui seront probablement peu changés) peuvent être regroupés au sein d'une librairie libblas.a.
Il suffira alors au moment de l'édition de liens, de préciser dans le Makefile que l'exécutable prod_fbls.exe est créé à partir du .o prod_fbls.o et de la librairie libblas.a (au lieu de la liste de tous les .o correspondant aux sous programmes du répertoire blas).
Pour être utilisable la librairie doit être munie d'un index (qui se trouve dans le fichier libblas.a) et qui est crée avec la commande ranlib.

Syntaxe :
ar -cr libblas.a (objets)

suivie de
ranlib libblas.a
Consulter man ar et man ranlib.

Écrire un Makefile dans le répertoire blas/ permettant de construire la librairie libblas.a.
Écrire un nouveau Makefile dans src/ créant l'exécutable prod_fbls.exe.
Prévoir dans ce Makefile une commande make install qui déplace prod_fbls.exe dans bin/ et une commande make clean qui nettoie les .o.

2.3 `make` et LaTeX

`make` n'est pas réservé à la compilation d'exécutables : il peut permettre de simplifier et d'automatiser la compilation de fichiers écrit en LaTeX. Recopier l'archive `exlatex.tgz` et extraire les fichiers qu'elle contient. On veut écrire un makefile dont la règle par défaut permette de générer les fichiers postscripts correspondant à tous les fichiers `.tex` du répertoire.

- Écrire une règle qui permet de générer le fichier `.dvi` correspondant à un fichier `.tex`.
- Écrire une règle qui permet de générer le fichier `.ps` correspondant à un fichier `.dvi`.
- Définir une variable `texfiles` désignant les fichiers `.tex` à compiler et une variable `psfiles` désignant les fichiers `.ps` correspondants (Indication : on utilisera les fonctions `wildcard` et `patsubst`).
- Définir correctement les dépendances pour `exemple07`.
- Définir la règle par défaut. Prévoir une règle `clean` qui supprime les fichiers inutiles.
- Tester !

2.4 Utiliser un débugeur : GDB et DDD

Un débogueur (en anglais, debugger) est un logiciel qui permet de déboguer, c'est-à-dire d'aider le programmeur à détecter des bogues dans un programme. Le programme à déboguer est exécuté à travers le débogueur et s'exécute normalement. Le débogueur offre alors au programmeur la possibilité de contrôler l'exécution du programme, en lui permettant par divers moyens de stopper (mettre en pause l'exécution du programme) et d'observer par exemple le contenu des différentes variables en mémoire. L'état d'exécution peut alors être observé afin, par exemple, de déterminer la cause d'une défaillance.

Le GNU Debugger également appelé gdb est le débogueur standard du projet GNU. Il est portable sur de nombreux systèmes Unix-like et fonctionne pour plusieurs langages de programmation, comme le C, le C++ et le Fortran.
L'interface de Gdb est une simple ligne de commande, mais il existe des applications frontales qui lui offrent une interface graphique beaucoup plus conviviale. L'utilitaire ddd par exemple permet de cliquer sur une ligne de code directement dans le listing pour y placer un point d'arrêt alors que gdb seul nécessite la saisie du numéro de ligne.
Le programme doit être compilé spécialement pour pouvoir être débogué. Avec les compilateurs GNU, il faut utiliser l'option `-g` (par exemple `gcc -g nomduprogramme.c -o nomduprogramme`)

2.4.1 Exercice avec DDD

Récupérez et compilez le programme dddbug.c. Ce programme trie des nombres entiers passés en arguments.

- Testez ce programme avec différents arguments. Quelle problème apparait ?
- Observez le déroulement du programme avec ddd et essayez de trouver le bug. Pour cela placer des points d'arret en différents points du programme et contrôler l'état des variables.
  Indication : Pour voir le contenu d'un tableau `a` de taille `size`, vous pouvez utiliser la syntaxe `*a@size`

2.5 Gestion des versions d'un logiciel : CVS

Il existe un outil de gestion de versions qui permet de gérer les versions successives d'un logiciel. CVS signifie Concurrent Versions System. CVS permet d'archiver différentes versions d'un système de fichiers et de garder une trace de qui a fait quelle modification sur quel fichier. Lorsqu'un projet fait intervenir plusieurs personnes, CVS gère l'édition simultanée d'un même fichier par plusieurs personnes.

Remarque : Il est prudent même lorsque l'on n'utilise pas un outil spécifique comme CVS, d'archiver différentes étapes d'un projet (avec la commande tar par exemple), afin d'être capable de revenir à une situation antérieure non buggée en cas de probléme ...