Cours Outils de Develloppement

Mendy Fatnassi

10 décembre 2020

Table des matières

Chapitre 1

Latex

Pour executer un document .tex via un terminal c-a-d en faire une version PDF , il suffit de taper $pdflatex nom_doc.tex

Installer un package :
Télécharger le .zip sur le site CTAN (librairie latex) puis copier les fichier du package dans le repertoire usrlocalsharetexmftexlatex ensuite mettez a jour la liste des packages dans la base de donnée latex avec sudo texhash .

1.1 En-tete d'un document LaTeX

Un document LaTeX commence par un en tete puis par le corps du document dans lequelle on vas ecire nos document .

```
\usepackage[latin1]{inputenc} % accents
\usepackage[T1]{fontenc}      % caractères français
\usepackage{geometry}         % marges
\usepackage[francais]{babel}  % langue
\usepackage{graphicx}         % images
\usepackage{verbatim}         % texte préformaté
```

Ensuite on peux placer ces quelques commande pour reinseigner des information sur le document .

```
\title{Rapport de stage}      % renseigne le titre
```

```
\author{Prénom Nom}          %   "   "   l'auteur
\date{18 juin 2007}          %   "   "   la future date de parution
\pagestyle{headings}         % affiche un rappel discret (en haut à gauche)
                             % de la partie dans laquel on se situe
```

Le corps du document seras delemité par des balises `\begin{},\end{}` , il faudra compiler le source LaTeX deux fois de suite pour qu'il gère correctement la table des matières :
Une première fois pour générer la table des matière et la seconde pour l'insérer dans le document.

```
\begin{document}
\maketitle % genere le titre du document
\tableofcontents % genere la table des matieres
...
\end{document}
```

1.2 Des balises en Latex

1.2.1 Les caractere

Les caractères suivant sont utilisés par LaTeX pour la compilation (ils ont une signification particulière pour la mise en forme du texte) et ne peuvent donc figurer tels quels dans le texte :

`$ & % # { } _` Il faudra les faire précéder d'un "\".
sauf pour :
« \ » qui s'écrit `\textbackslash`,
« ~ » " `\textasciitilde`,
« ^ » " `\textasciicircum`.

Pour les accents la syntaxe est la suivante : accent + lettre . Un accent peut etre aigue (´) ou grave (`).
Exemple : é (aigue) = `é`

1.2.2 Mise en forme

```
\underline
\textit ou \emph
\textbf
```

1.2.3 Inclure une image

Inclure le package `\usepackage{graphicx}` et utiliser la balise : `\includegraphics[width=1\textwidth,center]{monimage.jpg}`

1.2.4 Tableaux

On declare un tableau de la facons suivante :

```
\begin{center}
\begin{tabular}{|option|}
\hline
1 & 2 & 3 \\ \hline
a & b & c \\ \hline
4 & 5 & 6 \\ \hline
\hline
\end{tabular}
\end{center}
```

on obtient un tableau 3*3 avec comme option : l|c|r pour l'alignement de chacune des colonne .

1.2.5 Les environnements

-Pour afficher du code (C,HTML,...) :

```
\usepackage{listings}
\begin{DDbox}{\linewidth}
\begin{lstlisting}
....Du texte....
\end{lstlisting }
\end{DDbox}{\linewidth}
```

1.3 Latex pour les Mathematiques

Inclure \usepackage{amsmath}.Pour qu'un texte soit de la forme d'une expression mathématique on place le texte que l'on veux entre un bloc '${text}$'.

1.3.1 Synthaxe

```
\times = x (multiplication)
\div = symbole diviser
\ldots = ... (3 petits points)
\cdot = . (point centré , scalaire)
\overbrace{1,\ldots,n} = crochet du haut (ensemble)
\underbrace{1,\ldots,n} = crochet du bas (ensemble)
\neq =  (difference)
\equiv =  (equivalent)
\approx =  (approximation)
\simeq =  (approximation , plus au moins egal)
\leq, \geq =  ,
\leqslant, \geqslant =  ,
\ll, \gg =  ,
\pm =
\Rightarrow =
\Leftrightarrow =
\overline = barre au dessus (X barre)
\forall =  (pour tous)
\exists =  (existe)
\emptyset =  (vide)
\in =  (élément de)
\notin =  (pas un élément de)
\prod =  (produit)
\sum =  (somme) pour les indices \sum\limits_{i=0}^n
\wedge =  (et)
\vee =  (ou)
\cap =  (point d'intersection)
\cup =  (unité)
\subset =  (compris dans)
\notin =  (n'est pas compris dans)
```

1.3.2 Fonction Conditionnel

```
\[
    f(x)=
\begin{dcases}
    \frac{x^2-x}{x},& \text{if } x\geq 1\\
    0,              & \text{otherwise}
\end{dcases}
\]
```

Les ensembles :
il faut utiliser `\usepackage{amsfonts}` et la commande `\mathbb{R}` ce qui donne R

```
La fonction $f$ est définie par
\[
   f(x) = x-1
\].
On a alors
\begin{equation}
   f(x) = 0 \iff x = 1
\end{equation}
```

Donne :

La fonction f est définie par

$$f(x) = x - 1\,.$$

On a alors

$$f(x) = 0 \iff x = 1.$$

Liste de fonction prédéfini :
`\sin, \cos, \tan, \cot, \arcsin, \arccos, \arctan, \coth, \sinh, \cosh, \tanh, \ln,`

Si l'on veut mettre du texte normal au sein de l'équation, il faut utiliser la fonction `\text{texte}` Si l'on veut mettre une lettre ou quelques lettres en romain, on utilise `\mathrm{texte}`. De manière générale, les variables et

les grandeurs physiques sont en italiques alors que les constantes « mathématiques » sont en romain.

1.3.3 Lettres Grecque

Pour utiliser les lettres grecques, il suffit de taper leur nom en caractères latins précédé d'une contre-oblique.
Par exemple :

`\alpha` donne α; `\chi`*donne*χ; `\omega`, `\Omega`*donnent*ω, Ω.`\epsilon`*donne*ϵ, `\varepsilon`*do*

1.3.4 Exposant et Indices

Pour mettre du texte en exposant, on le place dans un bloc et on le fait précécder d'un chapeau « `^` ».

Pour mettre du texte en indice, on place le texte dans un bloc et on le fait précéder d'un tiret de soulignement « `_` ».

L'opérateur somme (sigma majuscule S) s'écrit `\sum` ; pour écrire les limites de la somme, il suffit de les metre en indice et exposant :

```
$\sum u_{n}$ ; $\sum_{i=1}^{n} (v_{i-1}+u_{i})$
ou \sum\limits_{i=0}^n //si On veux que les indice soit placer en dessous
```

Donne : $\sum u_n$; $\sum_{i=1}^{n}(v_{i-1} + u_i)$

par exemple :

```
$ u_n = 2^n $ donne un = 2n
$ u_{n+1} = 2^{n+1} $ donne un+1 = 2n+1
```

1.3.5 Fraction & Racines carré

Une équation se place entre deux signes dollar « $ ».
Pour écrire une fraction, nous utilisons la fonction

```
\frac{dividende}{diviseur}
```

exemple :

```
$ \frac{a+b}{c-d} $
```

Pour mettre des grandes parenthèses, il faut mettre \left devant celle de gauche et \right devant celle de droite :

```
$ \left( \frac{a+b}{c-d} \right)$
```

Pour les racines carré on utilise : `\sqrt{}`

1.3.6 vecteur

Vecteur :

```
\overrightarrow{AB}
ou
\vec{u}
```

Norme d'un vecteur :

```
\lVert levecteur \rVert
```

ou ||u||

1.3.7 limite de fonction

La limite d'une fonction s'écrit de cette facons :
`$\lim\limits_{x \rightarrow +\infty} f(x)$`
ce qui donne : $\lim\limits_{x \rightarrow +\infty} f(x)$
et
`$\lim\limits_{\subtrack{x \rightarrow -2 \\ x>-2}} f(x)$`
$\lim\limits_{\subtrack{x \rightarrow -2 \\ x>-2}} f(x)$

1.3.8 Integrale

Une integrale se note `$\int_a^b f(x) \, \mathrm dt$` .
$\int_a^b f(x) \, \mathrm{d}t$

1.3.9 Matrice & Vecteur Colonne

On peux ecrire un vecteur colonne comme une matrice :

```
\begin{pmatrix}
a & b & c\\
d & e & f\\
g & h & i
\end{pmatrix}
```

Ce qui donne :

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

on peux utiliser `\binom{n}{p}` pour superposer les 2 coeficients.

Chapitre 2

Debogage avec gdb

Pour lancer l'option de debogage lors de la compilation d'un programme , il faut utiliser l'option -g de gcc , exemple :
$gcc -g nom_prog.c -o nom_exec . Ensuite il faut lancer gdb , rien de plus simple : $gdb nom_exec .

Voici quelque commande :

run/quit	Demarrer/Quitter
continue	Reprendre l'execution
break num_[ligne|fonction] | break nom$_fic$: $nom\_[ligne\|fonction]$	Point d'arret
delete	Effacer tout les points d'arret
next|next nb_ligne	Executer une (ou n) lignes pas
step|step nb_ligne	Executer une (ou n) lignes pas
finish	Execute jusqu'au retour de la
print expr	Afficher la valeur de expressio
up/down	Remonte/descend dans la pile
bt | bt full	affiche la pile d'appels

info breakpoints : Affiche les point d'arret.

watch : On peux aussi introduire des watchpoint grace a la commande **watch num_ligne**$\|nom\_fonc, celaapoureffetd'interomprel'executionduprogrammelorsquelavaleurd'un$

backtrace : $Permetd'afficherlapiled'execution.$

where : $Affichelapiledesappels.$

Chapitre 3

Doxygen

Doxygen parmet de creer de la documentation pour un code source , le fichier seras parsé par doxygen et creer la documentaion necessaire grace au commentaire inserer dans le code source .

$ doxygen -g nom $_{fichier} -- > Genereunfichierdeconfigurationpardefautnommmdoxyfile.$
$\$doxygennom_{d}oxy -- > executedoxygen, appliquefichierdeconfigurationdoxygensurlesfichiercom$

Pour commenter un code en doxygen on utiliseras les delimiteur suivant :

```
/**
* \file nom_fic.c
* \brief description
* \author mendy
*/
```

Quelque balise :

```
\file <name>: Creer un bloc de documentation pour un fichier source ou d'en tete
\brief <description> : Permet d'ajouter une description
\author <name> : Nom des auteurs
\version <numeros> : Le numeros de la version du programme
\date <date>

\fn <declaration fonction> : Permet d'ajoute un bloc de documentation pour une fonct
\param <nom_param> : Descrit les parametre de la fonction
```

```
\return <description_param> : descrit le parametre de retoure de la fonction

\struct <nom_struct> <nom_header_file> <nom_header> : Bloc de documentation p
```

Chapitre 4

github

On se place dans un nouveau repertoire ou celui de notre choix puis on initialise un depot Git $git init ensuite il faut y ajouter tous les fichier dans l'index de git $git add .
Lorsqu'on modifie un repository, on doit enregistrer nos modifications dans Git en faisant un $git commit.
L'option -m nous permet de lui envoyer un message decrivant les modifications effectuees.

$git status => Affiche les commits.
$git push => Permet d'envoyez les modifications sur notre respository GitHub.
$git pull => Permet de recuperer des modifications sur notre respository GitHub.
$git log => Historique des commits.

$git rm . =>Quand on supprime des fichier, il reste dens l'indexe de fichier git pour cela on peux utiliser la commande "rm git ." pour validé la suppression des fichier apres un rm.
Note : L'expression "." et "*" sont differente. Exemple :
-$git rm * supprime tout les fichier de l'indexe ($git rm *.png -> supprime toute les image de type png).
-$git rm . supprime uniquement les fichier de l'indexe marqué comme "supprimé".

Git peux se parametrer grace a la commande $git config .Cela permet de voir et modifier les variables de configuration qui contrôlent tous les aspects de l'apparence et du comportement de Git.

Fichier /etc/gitconfig : Contient les valeurs pour tous les utilisateurs et tous les dépôts du système. Si vous passez l'option –system à git config, il lit et écrit ce fichier spécifiquement.

Fichier /.gitconfig : Spécifique à votre utilisateur. Vous pouvez forcer Git à lire et écrire ce fichier en passant l'option –global.

Fichier config dans le répertoire Git (c'est-à-dire .git/config) du dépôt en cours d'utilisation : spécifique au seul dépôt en cours.

La première chose à faire après l'installation de Git est de renseigner votre nom et votre adresse de courriel. C'est une information importante car toutes les validations dans Git utilisent cette information.

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

Pour verifier les parametres on peux utiliser la commande : $git config –list

-Gestion id et mp en memoire , il y a 3 mode :

-Par défaut, rien n'est mis en cache. Toutes les connexions vous demanderont votre nom d'utilisateur et votre mot de passe.

cache : conserve en mémoire les identifiants pendant un certain temps.Par defaut le temps d'expiration est de 15min.

store : sauvegarde les identifiants dans un fichier texte simple sur le disque, et celui-ci n'expire jamais .

Vous pouvez choisir une de ces méthodes en paramétrant une valeur de configuration Git :

```
$ git config --global credential.helper cache
```

Certains de ces assistants ont des options.
L'assistant store accepte un argument –file <chemin> qui permet

de personnaliser l'endroit où le fichier texte est sauvegardé (par défaut, c'est /.git-credentials).

L'assistant cache accepte une option –timeout <secondes> qui modifie la période de maintien en mémoire.

Exemple :

```
$ git config --global credential.helper 'store --file ~/.git-credentials'
```

Version moins securisé :

```
$git config --global user.name "your username"
$git config --global user.password "your password"
```

On peux aussi verifier dans les fichier `~/.gitconfig` et `~/.git-credentials` si les information presente sont bien enregistré .

Chapitre 5

Compilation Makefile

Un Makefile peut être écrit à la main, ou généré automatiquement par un utilitaire (ex : automake,gmake etc). Il est constitué d'une ou de plusieurs règles de la forme :

cible : dépendances

commandes
Ce qui suit présente la création d'un Makefile pour un exemple de projet. Supposons pour commencer que ce projet regroupe trois fichiers exemple.h,exemple.c et main.c.
Un fichier Makefile de base de ce projet pourrait s'écrire :

Exemple :

```
mon_executable : exemple.o main.o
gcc -o mon_executable exemple.o main.o

exemple.o: exemple.c
gcc -o exemple.o -c exemple.c -Wall -O

main.o: main.c exemple.h
gcc -o main.o -c main.c -Wall -O
```

On peux ameliorer notre makefile en rajoutant quelque cibles all : , clean : et mrproper : note Makefile devient donc :

Exemple :

```
all: mon_executable

mon_executable: exemple.o main.o
gcc -o mon_executable exemple.o main.o

exemple.o: exemple.c
gcc -o exemple.o -c exemple.c -Wall -O

main.o: main.c exemple.h
gcc -o main.o -c main.c -Wall -O

clean:
rm -f *.o core

mrproper: clean
rm -f mon_executable
```

Il est possible de définir des variables dans un Makefile. Elles se déclarent sous la forme NOM=valeur et sont appelées sous la forme $(NOM), à peu près comme dans un shellscript.Parmi quelques variables standards pour unMakefilede projet C ou C++, on trouve :
–CC : qui désigne le compilateur utilisé ;
–CFLAGS : qui regroupe les options de compilation ;
–LDFLAGS : qui regroupe les options d'édition de liens ;
–EXEC ou TARGET : qui regroupe les exécutables.

Exemple :

```
CC=gcc
CFLAGS=-Wall -O
LDFLAGS=
EXEC=mon_executable

all: $(EXEC)

mon_executable: exemple.o main.o
$(CC) -o mon_executable exemple.o main.o $(LDFLAGS)
```

```
exemple.o: exemple.c
$(CC) -o exemple.o -c exemple.c $(CFLAGS)

main.o: main.c exemple.h
$(CC) -o main.o -c main.c $(CFLAGS)

clean:
rm -f *.o core
mrproper: clean
rm -f $(EXEC)
```

Il existe aussi, et c'est encore plus intéressant car très puissant, des variables internes au Makefile, utilisables dans les commandes ; notamment :

```
-$@+: nom de la cible ;
-$<: nom de la première dépendance ;
-$^: liste des dépendances ;
-$?: liste des dépendances plus récentes que la cible ;
-$*: nom d'un fichier sans son suffixe;
```

Exemple :

```
CC=gcc
CFLAGS=-Wall -O
LDFLAGS=
EXEC=mon_executable

all: $(EXEC)

mon_executable: exemple.o main.o
$(CC) -o $@ $^ $(LDFLAGS)

exemple.o: exemple.c
$(CC) -o $@ -c $< $(CFLAGS)

main.o: main.c exemple.h
```

```
$(CC) -o $@ -c $< $(CFLAGS)

clean:
rm -f *.o core
mrproper: clean
rm -f $(EXEC)
```