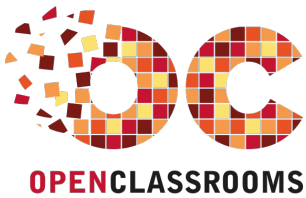
Compiler facilement Qt

Par Hiura



www.openclassrooms.com

Sommaire

Compiler facilement Qt

Par  Hiura

Mise à jour : 01/01/1970
Difficulté : Facile

Pour commencer je souhaite la bienvenue à tous les zéros ici.

Pourquoi faire un tuto sur la compilation de Qt alors qu'il en existe déjà plusieurs et que M@teo21 va sûrement aborder aussi ce sujet ? Pour moi la réponse est claire : pour avoir une autre approche que celle des tutos déjà existants, et surtout pour permettre à tous les zéros qui le souhaitent d'étudier Qt avant que le tutoriel de M@teo21 ne soit écrit.

Avant tout, je vais exposer brièvement ce que va aborder ce tutoriel.

Ici, vous apprendrez à :

- installer Qt sur votre plate-forme ;
- compiler un projet utilisant Qt (basique) ;
- compiler un projet utilisant Qt (avancé).

Quand je dis compiler, je ne parle pas de cliquer simplement sur un bouton de votre IDE : je vous montrerai comment compiler sans IDE - soit directement dans le terminal, soit dans l'invite de commandes.

Il faut bien noter que je ne vous apprendrai pas à coder quoi que ce soit. Et ceci pour une simple raison : je suis un zéro, tout comme vous, et je ne maîtrise pas assez bien Qt pour me le permettre.

La première chose que je vous conseille c'est de mettre cette page dans vos favoris.

Avant de commencer je vous explique en deux mots ce qu'est Qt : Qt est une bibliothèque permettant de créer des fenêtres ; elle n'est pas destinée aux jeux, bien qu'on puisse y intégrer des bibliothèques comme la SDL ou la SFML ; elle est utilisée par beaucoup de monde, dont des professionnels, et elle est considérée comme très puissante. Elle a une grande rivale : wxWidgets. Pour en savoir plus sur Qt, je vous invite à aller lire ce wiki.

Merci à raphamil pour son aide

Sommaire du tutoriel :

Installation de Qt

Pour commencer, on va installer Qt. Mais comme l'installation est différente sous les différents OS, je vais diviser ce chapitre.

Windows

Le téléchargement

On commence par le téléchargement. Tout d'abord rendez-vous sur cette page : http://www.qtsoftware.com/downloads-fr. Deux versions y sont proposées : l'une *Open Source* et l'autre commerciale. Il faut bien noter que la version *Open Source*, bien que gratuite, ne vous permet pas de distribuer une application sans son code. Si vous désirez conserver votre code source pour vous seulement il vous faut payer (cher). Il existe une alternative : *wxWidgets*. Vous trouverez plus d'information sur sa licence sur cette page (en).

Après avoir choisi entre *Open Source* et *Commercial*, vous devez choisir entre deux autres versions de Qt. Nous utiliserons ici la version *Développement d'applications*. Encore une séries d'options s'offre à vous. Pour ce tutoriel, prenez *Qt pour Windows : C++*. Notez aussi qu'il existe une intégration à l'IDE Eclipse, dans la rubrique *Autres téléchargements*, mais nous n'en parlerons pas.

Choisissez l'archive *qt-win-opensource-x.y.z-mingw.exe*, avec MinGW, sur n'importe quel miroir.

x.y.z sera utilisé tout au long du tutoriel pour désigner la dernière version de Qt.

L'installation

Une fois le téléchargement terminé, lancez l'exécutable.
Il n'y a que deux points que je vais éclaircir avec vous à propos de cette installation :

- le choix du répertoire d'installation ;
- l'installation du compilateur.

Le répertoire : choisissez une destination pour l'installation (personnellement je ne l'ai pas changée, simplement pour ne pas avoir à en trouver une autre). La suite du tuto est basée sur le répertoire par défaut, mais les différences sont faciles à prendre en charge.

Le compilateur : pour utiliser Qt, il vous faut le compilateur *MinGW*. On vous propose dans cette installation d'indiquer où est déjà installé ce dernier, ou de l'installer. Si vous ne l'installez pas, **vérifiez qu'il est à la même version que celui proposé par Qt**. Si vous utilisez une version plus récente de MinGW vous pouvez avoir des problèmes, notamment avec w32api.h.

Je vous recommande de faire comme ça : 

Après l'installation, on nous propose de voir la documentation et de regarder quelques exemples faits avec Qt. Je vous encourage vivement à vous balader parmi les exemples et dans la documentation pour en apprendre davantage sur Qt.

Si vous désirez désinstaller Qt, il vous suffit de vous rendre dans le dossier d'installation et de lancer l'exécutable *uninst.exe*, ou simplement via le menu démarrer.

La configuration

Les manipulation décrites ici sont prévues pour Windows XP. Il se peut donc que les manipulations ne correspondent pas totalement sous Vista.

L'installation n'est en fait pas réellement terminée : il faut encore configurer le tout.

Pour ce faire*, suivez ces étapes :

1. ouvrez le poste de travail et cliquez sur le fond (blanc) et sélectionnez "*propriétés*" ;
2. dans l'onglet "*Avancé*" cliquez sur "*Variables d'environnement*" (raccourci clavier : ALT+V) ;
3. sélectionnez la variable "*PATH*"** dans le cadre supérieur puis cliquez sur modifier ;
4. ajoutez "*C:\Qt\x.y.z\bin;C:\MinGW\bin*"*** à la fin ;
5. fermez ensuite les fenêtres en cliquant sur les boutons "*OK*" ;
6. pour terminer, il vous faut redémarrer votre PC pour que ces changements prennent effet.

* Ces étapes peuvent être évitées en lançant la "Qt Command Prompt" depuis le menu démarrer. Cette invite de commande ajoute temporairement les chemins vers Qt et MinGW dans la variable d'environnement PATH pour la durée de la session. Ainsi vous pourrez compiler vos projets sans faire cette modification de PATH.
** Il se peut que la variable PATH n'existe pas, si tel est votre cas, faites simplement "*nouveau*".
*** Si vous n'avez pas mis les répertoires par défaut de l'installation, il vous suffit de modifier le chemin d'accès (mais n'oubliez pas le /bin).

Vous vous demandez peut-être pourquoi je vous ai demandé de faire ça. Je commence par rassurer les paranos : ce que vous venez de faire ne permet pas à quelqu'un de malintentionné de rentrer sur votre ordinateur.

Si vous n'aviez pas fait cette manipulation, vous n'auriez pas pu taper simplement *make* dans la console pour compiler ; vous auriez dû à chaque fois entrer *C:\Qt\x.y.z\bin\make*. C'est quand même mieux quand c'est plus court, hein ?

On passe à la suite ?
Ce que je vais vous faire faire maintenant n'est pas absolument nécessaire mais permet d'optimiser les exemples de Qt pour votre ordinateur.

Cette fois il vous faut ouvrir "l'invite de commandes" qui se trouve dans le menu démarrer dans le dossier accessoires, ou en faisant *Exécuter* (touche Windows + R) et en entrant "*cmd*".

Maintenant il faut vous rendre dans le dossier d'installation de Qt, qui est par défaut "*C:\Qt\x.y.z*". Pour ce faire, entrez cette commande :

Code : Console

```
cd C:\Qt\x.y.z
```

Et pour finir tapez ceci :

Code : Console

```
configure.exe
```

(Le .exe est facultatif)

Il faut attendre un bon moment (mais vraiment un bon). Autant faire autre chose en attendant ; comme lire la documentation de Qt.

Vous pouvez soit la trouver en ligne sur le lien que vous avez ajouté dans vos favoris au début de ce tutoriel, soit en allant dans le menu démarrer et en lançant l'assistant de Qt qui se trouve dans le dossier intitulé "*Qt by Trolltech vx.y.z (OpenSource)*" par défaut.

Ce n'est pas terminé pour autant. Maintenant il vous faut compiler tout ça ; pour ce faire, entrez simplement "*make*" dans l'invite de commandes.

Et c'est reparti pour un bon moment d'attente (et cette fois quand je dis un bon moment, je veux parler de deux heures minimum)... On reprend la doc ?

Je vous rassure : par la suite, vous n'aurez plus besoin d'attendre aussi longtemps.

Linux

Pour ce tutoriel, je vais me baser sur Ubuntu 7.04 (Gnome), et par conséquent, dans cette méthode d'installation, les lignes de commandes ne sont valides qu'avec les "debian-like".

Pour commencer, il faut télécharger Qt. On a deux choix : soit on prend le fichier sur le site de trolltech, soit on le prend depuis le gestionnaire de paquets.

Optons pour la deuxième solution qui est plus simple à mes yeux.

Ouvrez votre gestionnaire de paquets Synaptic et recherchez *libqt4-dev*. Sélectionnez-le pour l'installation. À noter que d'autres paquets vont être téléchargés aussi, ce qui est normal.

Recherchons aussi la documentation : *qt4-doc*, et le designer : *qt4-designer*.

Une fois tous ces paquets sélectionnés, lancez l'installation.

Voilà c'est tout.

Vous pouvez accéder à la doc en local en tapant *file:///usr/share/qt4/doc/html/index.html* dans la barre URL de votre navigateur. Ou si vous désirez les visionner avec l'assistant de Qt, installez celui-ci :

Code : Console

```
sudo apt-get install qt4-dev-tools
```

Et pour l'exécuter :

Code : Console

```
assistant
```

Pour lancer le designer, c'est simple : lancez un terminal et entrez *designer*.

Mac OSX

N'ayant plus de Mac sous la main je ne peux pas tester l'installation. Je ne mets donc pas à jour la suite. Néanmoins le *.dmg* à télécharger doit surement être celui-ci (version *debug* seulement).

Le téléchargement

La première étape est toujours le téléchargement, alors rendez-vous sur cette page et choisissez l'archive qt-mac-opensource-4.X.X.**dmg** (où X est un numéro de sous-version).

L'installation

Je crois bien que c'est la plus simple de toutes. Il vous suffit d'exécuter le fichier téléchargé, une fenêtre s'ouvre qui contient trois choses. Seul Qt.mpkg est important : exécutez-le. Suivez les étapes. Vous arrivez à la fin, et vous avez réellement fini.

Vous trouverez Qt Designer, Qt Linguist, Qt Demo et Qt Assistant dans /Developer/Applications/Qt/. Amusez-vous bien !

Compiler un projet (basique)

C'est bien joli d'avoir fait cette super longue installation, mais il faut continuer.

Qu'allons-nous faire maintenant ? Simplement voir comment compiler un projet.

Il existe deux méthodes (qui sont en fait les mêmes, vous comprendrez plus tard).

La première que je vais vous exposer est très (voire trop) simple. Elle est limitée : pour certains projets elle ne marchera pas sans une modification d'un fichier. Mais elle peut être pratique si votre programme est très simple (ou pour être plus correct : s'il n'utilise pas autre chose que les modules de base de Qt, soit *gui* et *core*).

Pour vous montrer comment faire, je vais vous donner un exemple. Ce sera le plus simple pour vous et pour moi.

Commencez par créer un dossier que vous appellerez "*t10*" (c'est un exemple, peu importe en réalité son nom). Son emplacement n'a aucune importance, mais plus il est près de la racine du disque, plus c'est pratique.

Dans ce dossier, vous allez placer tous les fichiers .h et .cpp du tutoriel officiel *t10*.
Je vous donne les références pour les trouver : lien, emplacement sur votre PC : "*C:\Qt\4.3.0\examples\tutorial\t10*".
Si vous êtes sous Linux et que vous avez téléchargé Qt via le gestionnaire Synaptic, vous n'avez pas ces fichiers sur votre ordinateur. Vous pouvez les télécharger ici, choisissez l'archive qui correspond le mieux à votre plate-forme et à votre version de Qt ; cette archive vous sera utile dans le prochain chapitre.
Si vous êtes un adepte de Mac OSX, vous trouverez les sources des exemples dans /Developer/Examples/Qt/.

Vous devez donc avoir deux fichiers .h et trois fichiers .cpp dans ce dossier.

Ensuite ouvrez votre invite de commandes et rendez-vous dans ce fameux dossier ; entrez aussi les quelques lignes supplémentaires que voici.

Code : Console

```
cd "CHEMIN_D_ACCES_AU_DOSSIER_T10"\t10
qmake -project
qmake
make
```

Sous Mac OSX il y a une petite nuance : vous ne pouvez pas exécuter *make*. Mais ce n'est pas grave, car qmake a créé, au lieu d'un Makefile, un projet Xcode. Il vous suffit donc de l'ouvrir avec Xcode et de compiler. Notez que si vous ne rencontrez pas ce problème tant mieux pour vous.

Voilà c'est tout.

Mais il vous faut comprendre ce que vous avez fait sinon ça ne sert à rien.

Donc dans l'ordre vous avez fait ceci (ligne par ligne).

1. Vous vous êtes rendus dans le dossier t10.
2. Vous avez demandé à créer automatiquement un fichier projet (il contient tous les .h, .cpp et .ui du dossier) ; un fichier .pro a été créé.
3. Vous avez préparé le projet à la compilation en le précompilant.
4. Vous avez compilé le projet.

Avec Qt, il faut précompiler le projet avec qmake pour qu'il puisse être compilé correctement.

Nous aborderons plus précisément la configuration manuelle du projet dans le prochain chapitre, où il vous sera demandé d'apprendre quelques petites choses.

Compiler un projet (avancé)

Dans cette partie, je vais vous apprendre à configurer manuellement votre projet.

Mais avant, je vais vous montrer l'intérêt de le faire manuellement.

Allons-y !

Un problème bien ennuyeux

Commencez par copier les fichiers .h, .cpp et .ui de "*C:\Qt\4.3.0\examples\network\http*"* (soit 5 fichiers) dans un nouveau dossier quelconque.
* Pour les linuxiens : ces fichiers se trouvent dans l'archive que vous avez téléchargée tout à l'heure. Pour les utilisateurs de Mac OSX, rendez-vous dans /Developer/Examples/Qt/network/http/.

Ensuite essayez de compiler le projet avec la même méthode que dans le chapitre précédent.

Ce n'est pas beau ce qui nous arrive là, hein ?

Nous venons de tomber sur un problème dû à la précompilation automatique.

Je vais vous demander de comparer deux fichiers : le .pro qui s'est créé dans le dossier et le fichier que j'ai fait moi.

Code : Autre - .pro automatique

```
#############################################################################
# Automatically generated by qmake (2.01a) mer. 18. juil. 21:39:31 2007
```

```
#############################################################################

TEMPLATE = app
TARGET =
DEPENDPATH += .
INCLUDEPATH += .

# Input
HEADERS += httpwindow.h
FORMS += authenticationdialog.ui
SOURCES += httpwindow.cpp main.cpp
```

Code : Autre - Le mien

```
HEADERS     += ui_authenticationdialog.h
HEADERS     += httpwindow.h
SOURCES     += main.cpp
SOURCES     += httpwindow.cpp
FORMS       += authenticationdialog.ui

QT          += network
```

Notez que je n'y ai mis que le strict nécessaire.

Étudions les différences.

Tout d'abord, je n'ai pas mis d'en-tête, mais ça n'a aucune conséquence sur la compilation alors passons.
J'ai supprimé tout ce qui précède "*# Input*" ; ce passage n'était pas indispensable pour nous.
Ensuite j'ai mis un fichier par ligne ; comment est-ce possible ? Tout simplement parce que j'ai mis "+=" (ça ne vous rappelle rien ?) ; "-=" existe aussi, mais passons.
On peux aussi faire comme ceci :

Code : Autre

```
HEADERS     += ui_authenticationdialog.h \
          httpwindow.h
```

Ce n'est qu'une question de goût, les trois techniques ont le même effet.

Il faut néanmoins que je précise quelque chose encore sur ces 5 lignes qui est plutôt flagrant : le mot-clé "*HEADERS*" correspond aux .h/.hpp et autres variantes (header), "*SOURCES*" aux sources et "*FORMS*" aux .ui (qui sont des fichiers créés avec Qt Designer).

Et pour finir, j'ai ajouté un mot-clé : "*QT*". J'ai indiqué par ce mot-clé que le module "*network*" devait être ajouté au projet.

Qui a dit : "Un problème bien ennuyeux" ?

Essayez ensuite de compiler en remplaçant votre fichier .pro par le mien et en n'utilisant que ces deux commandes après être arrivé dans le dossier :

Code : Console

```
qmake
make
```

Et cette fois, la compilation fonctionne ! Vous êtes désormais aptes à compiler vos futurs projets utilisant Qt ! Félicitations !

Il existe de nombreuses variables permettant de configurer encore plus son projet, je vous invite à lire cette page pour plus d'infos.

Annexe

Dans cette annexe je vais vous présenter les options importantes de qmake. Pour chaque variable vous trouverez une petite fiche de description basique.
Nous allons voir :

- TEMPLATE
- CONFIG
- QT
- MOC_DIR
- OBJECTS_DIR
- DESTDIR

- TARGET
- DEPENDPATH
- LIBS

TEMPLATE

Variable	TEMPLATE
But	Indiquer la nature du projet.
Valeurs	app (par défaut) lib
Référence	Lien.

Cette variable de qmake permet d'indiquer la nature de notre projet : soit une application (*.exe), soit une bibliothèque (*.dll/*.so, *.a/*.lib). Il existe d'autres "natures" de projet, mais je ne vais pas les aborder.
Pour indiquer que notre projet est destiné à créer un exécutable, placez, si vous le désirez, *TEMPLATE = app* dans votre fichier .pro. Si vous désirez créer une bibliothèque, mettez-y *TEMPLATE = lib*.

CONFIG

Variable	CONFIG
But	Spécifier les options de compilation du projet.
Valeurs	release debug
Valeurs	warn_on warn_off
Valeurs	qt (par défaut) opengl dll staticlib
Valeurs	exceptions stl
Référence	Lien.

Cette variable de qmake permet d'indiquer les options de configuration du projet, et les options de compilation. Si vous désirez compiler votre programme pour le distribuer, ajoutez la valeur *release* à *CONFIG*. Si vous désirez le compiler pour faire des tests, vous pouvez ajouter l'option *debug* ; dans ce cas le programme sera moins optimisé et plus lourd sur le disque.
Si vous désirez désactiver les messages d'avertissement, ajoutez *warn_off*. Dans le cas contraire ajoutez *warn_on*.
Par défaut l'application que vous créez est une application Qt (*qt*). Vous pouvez préciser que vous utilisez OpenGL en ajoutant *opengl* : ainsi votre projet sera configuré pour utiliser les fonctions d'OpenGL.
Enfin, vous pouvez permettre l'utilisation des exceptions (cf. tuto de Nanoc) en ajoutant *exceptions* à la variable *CONFIG*. Vous pouvez aussi indiquer que vous utilisez la STL dans votre projet avec *CONFIG += stl*.

QT

Variable	QT
But	Ajouter des modules de Qt dans le projet.
Valeurs	core (par défaut) gui (par défaut) network opengl sql svg xml qt3support
Référence	Lien.

Si vous utilisez un module de Qt vous devez le préciser pour que la compilation se passe bien, sinon on arrive au problème de tout à l'heure avec l'exemple http.

Si vous utilisez OpenGL, ce n'est pas dans la variable *QT* qu'il faut le préciser, mais dans *CONFIG*. Ce *opengl* indique que vous utilisez le module QtOpenGL.

Si vous n'utilisez pas le module QtGui vous pouvez l'enlever comme ceci : *QT -= gui* .

MOC_DIR

Variable	MOC_DIR
But	Indiquer où sauver les fichiers .moc créés par qmake.
Valeurs	Chemin relatif ou chemin absolu.

Référence	Lien.

Un petit exemple pour illustrer ceci.
Mon projet se trouve dans le répertoire /C++/MonProjetQt/, et je veux que mes fichiers .moc soient sauvés dans /C++/MonProjetQt/tmp/moc/. J'ai deux possibilités pour arriver à mes fins :

Code : Autre - Mon fichier .pro

```
# Première solution : chemin absolu.
MOC_DIR = /C++/MonProjetQt/tmp/moc/
# Deuxième solution : chemin relatif.
MOC_DIR = ./tmp/moc/
```

OBJECTS_DIR

Variable	OBJECTS_DIR
But	Très similaire à MOC_DIR : indiquer où sauver les fichier objets (*.o).
Valeurs	Chemin relatif ou chemin absolu.
Référence	Lien.

Le fonctionnement est exactement le même que pour MOC_DIR.

DESTDIR

Variable	DESTDIR
But	Indique le répertoire de destination de *TARGET*.
Valeurs	Chemin relatif ou chemin absolu.
Référence	Lien.

Le fonctionnement est exactement le même que pour MOC_DIR.

TARGET

Variable	TARGET
But	Indiquer le nom de l'exécutable ou de la bibliothèque.
Valeurs	Chaîne de caractères.
Référence	Lien.

L'utilisation est extrêmement facile :

Code : Autre - Indiquer le nom du fichier produit

```
TARGET = MonNom
```

DEPENDPATH

Variable	DEPENDPATH
But	Indiquer les répertoires où Qt doit chercher les fichiers du projet.
Valeurs	Chemin relatif ou chemin absolu.
Référence	Lien.

Pour bien comprendre à quoi sert cette variable je vais vous montrer deux .pro qui font la même chose.

Sans	Avec

Code : Autre - Mon fichier pro avant.

```
# Indput ##############################
#--------------------------------------
SOURCES         += ./Sources/main.cpp

#--------------------------------------
HEADERS         += ./Sources/Wizard/Wizard.hpp
```

Code : Autre - Mon fichier pro après.

```
# Indput ##############################
#--------------------------------------
DEPENDPATH      += ./Sources
DEPENDPATH      += ./Sources/Wizard
DEPENDPATH      += ./Sources/FTP

#--------------------------------------
SOURCES         += main.cpp
```

```
SOURCES         += ./Sources/Wizard/Wizard.cpp

HEADERS         += ./Sources/Wizard/StepOne.hpp
SOURCES         += ./Sources/Wizard/StepOne.cpp

HEADERS         += ./Sources/Wizard/StepTwo.hpp
SOURCES         += ./Sources/Wizard/StepTwo.cpp

HEADERS         += ./Sources/Wizard/StepThree.hpp
SOURCES         += ./Sources/Wizard/StepThree.cpp

HEADERS         += ./Sources/Wizard/StepFour.hpp
SOURCES         += ./Sources/Wizard/StepFour.cpp

#--------------------------------------
HEADERS         += ./Sources/FTP/FTPTestConnection.hpp
SOURCES         += ./Sources/FTP/FTPTestConnection.cpp

#--------------------------------------
HEADERS         += ./Sources/FTP/GetFile.hpp
SOURCES         += ./Sources/FTP/GetFile.cpp
```

```
#--------------------------------------
HEADERS         += Wizard.hpp
SOURCES         += Wizard.cpp

HEADERS         += StepOne.hpp
SOURCES         += StepOne.cpp

HEADERS         += StepTwo.hpp
SOURCES         += StepTwo.cpp

HEADERS         += StepThree.hpp
SOURCES         += StepThree.cpp

HEADERS         += StepFour.hpp
SOURCES         += StepFour.cpp

#--------------------------------------
HEADERS         += FTPTestConnection.hpp
SOURCES         += FTPTestConnection.cpp

#--------------------------------------
HEADERS         += GetFile.hpp
SOURCES         += GetFile.cpp
```

C'est quand même plus lisible après, hein.

LIBS

Variable	LIBS
But	Ajouter des bibliothèques au projet pour l'édition des liens.
Valeurs	[Windows] chemin absolu [Unix] chemin absolu ou relatif
Référence	Lien.

La doc de Qt propose un exemple :

Code : Autre

```
unix:LIBS += -L/usr/local/lib -lmath
win32:LIBS += c:/mylibs/math.lib
```

Je vais vous l'expliquer un peu.

Tout d'abord, vous pouvez voir en début des deux lignes "*plate-forme*". Qmake est intelligent, il va donc utiliser la première ligne si vous compilez sur Linux et la deuxième si vous êtes sur Windows.

Ensuite, nous trouvons notre variable *LIBS*. Ici nous indiquons à l'éditeur de liens (linker) que nous utilisons la bibliothèque math.

Prenons par exemple la bibliothèque SFML (tuto pour utiliser Qt avec la SFML). Dans votre fichier pro, vous n'aurez donc qu'à ajouter :

Code : Autre

```
unix:LIBS += -lsfml-graphics -lsfml-window -lsfml-system
win32:LIBS += C:\MinGW\lib\libsfml-graphics.a C:\MinGW\lib\libsfml-
window.a C:\MinGW\lib\libsfml-system.a
```

Voilà, c'est fini. J'espère que ça vous a plu .

Bon voilà le cours est terminé, j'espère qu'il vous a plu et qu'il vous a été utile.

Merci de faire des commentaires dans la rubrique adéquate pour que je puisse améliorer ce tutoriel.

Ce tutoriel a été corrigé par les zCorrecteurs.