

Guide de la Compilation de Gcc sous windows avec MinGW

par Philippe Dunki (<http://philippe-dunki.developpez.com/>)

Date de publication : 22/09/2007

Dernière mise à jour :

Gcc est une collection de compilateurs pouvant gérer de nombreux langages qui évoluent très rapidement. Si vite que les équipes qui le portent sous Windows n'arrivent pas à suivre le rythme. C'est la raison pour laquelle cet article vous guidera parmi les étapes à suivre pour compiler la dernière version depuis les sources.

I - Introduction

- I-A - Gcc, qu'est-ce que c'est?
- I-B - MinGW, qu'est-ce que c'est?
- I-C - GNU..., mais c'est linux?
- I-D - Pourquoi compiler les sources?
- I-E - Pourquoi MinGW et non cygwin?
- I-F - Deux passes de compilations

II - Le "minimum compilant"

- II-A - GCC-core
- II-B - GCC-ada
- II-C - Binutils
- II-D - Bison
- II-E - Flex
- II-F - Libintl
- II-G - Libiconv
- II-H - W32api
- II-I - Mingw-runtime
- II-J - Texinfo
- II-K - Où les trouver?
- II-L - Installer tout cela

III - Un système linux sous windows

- III-A - Installer MSYS
- III-B - Premiers contacts
 - III-B-1 - La console sous linux
 - III-B-2 - Derniers préparatifs
- III-C - Bienvenue sous linux
- III-D - Créer les dossiers de travail
- III-E - Pour ceux qui ont postposé la décompression des outils temporaires
- III-F - L'arborescence finale
- III-G - Convention pour les codes
- III-H - Configurer le PATH

IV - Les version testées

- IV-A - Binutils-2.17
- IV-B - GCC-4.2.1

V - La première passe

- V-A - Compiler binutils
- V-B - Compiler les bibliothèques indispensables
 - V-B-1 - Compiler w32-api
 - V-B-2 - compiler mingw-runtime
 - V-B-3 - Compiler libiconv (facultatif)
 - V-B-4 - GMPLib et MPRF
 - V-B-4-a - Compiler GMPLib
 - V-B-4-b - Compiler MPRF
- V-C - Compiler GCC
 - V-C-1 - Avant la configuration
 - V-C-2 - La commande de configuration
 - V-C-2-a - Comprendre la commande
 - V-C-2-b - Au sujet de la compilation en elle-même
 - V-C-2-c - Lancer la compilation

VI - La deuxième passe

VII - Liens et remerciements

- VII-A - Trouver les archives
- VII-B - Sources d'inspiration
- VII-C - Liens connexes

VII-D - Remerciements

I - Introduction

I-A - Gcc, qu'est-ce que c'est?

Gcc est l'acronyme de GNU Compilers Collection. Il s'agit en fait d'une collection qui regroupe des compilateurs supportant de nombreux langages, tels que le C, le C++, l'ObjectiveC, l'ObjectiveC++, le Treelang (un langage expérimental dont la principale raison d'être est de servir de support au développement de GCC), l'Ada, le FORTRAN et même le Java.

Il fournit en outre les différentes bibliothèques standard pour ces langages.

I-B - MinGW, qu'est-ce que c'est?

MinGW est l'acronyme de **M**inimalist **G**cc for **W**indows.

Le format d'exécutables sous Windows (**P**ortable **E**xecutable) est très différent de ceux qui sont utilisés sous linux. Il faut donc ajouter le support de ce format à GCC pour qu'il soit capable de générer des programmes exécutables sous cet OS et de lier des bibliothèques.

Le but du projet est, tout simplement, de fournir cette adaptation de GCC sous windows.

I-C - GNU..., mais c'est linux?

Effectivement, GNU (Gnu's Not Unix) est principalement associé à linux. Et c'est sans doute ce qui fait que GCC évolue si rapidement: En effet, les versions en développement de GCC apportent déjà le support d'une partie au moins de la prochaine norme de C++ (C++0x) qui n'est pourtant pas attendue, par les plus optimistes, avant 2009.

Ce qui explique aussi dans une certaine mesure que les mainteneurs de MinGW aient autant de mal à garder les versions aussi à jour que ce qui existe sous linux.

I-D - Pourquoi compiler les sources?

Les raisons qui pourraient vous faire décider de compiler les sources de GCC sont multiples:

D'abord, la dernière version considérée comme stable de GCC au moment d'écrire ces lignes est la version 4.2.1, alors que cette version n'est encore estampillée que "prévue" sous MinGW, pour qui, la version stable est encore la version 3.4.5 de la collection.

De plus, il est possible de disposer d'un "snapshot" hebdomadaire de la version de développement, et certains pourraient être intéressés par le fait d'en profiter.

Enfin, il faut savoir que les versions binaires de MinGW sont généralement compilées de manière à être fonctionnelles sur la plupart des processeurs: "au mieux", sous de vieux Pentium II, "au pire" avec d'antiques 3x86.

Nombreux sont ceux qui pourraient, tout simplement, souhaiter disposer d'un compilateur optimisé pour l'ordinateur dont ils disposent.

I-E - Pourquoi MinGW et non cygwin?

Deux raisons m'ont fait choisir une compilation sous MinGW plutôt que sous cygwin:

La première est que cygwin est un émulateur linux qui fournit un compilateur croisé pour windows, et qu'il nécessite donc une optique de travail plus "orientée" vers linux. Vous me direz sans doute que vous devrez de toutes manières passer par un émulateur linux pour le compiler, mais cela pourra rester votre seule expérience sous cet environnement si vous le souhaitez, alors que vous devriez y retourner chaque fois que vous voudriez compiler sous cygwin.

La deuxième tient tout simplement au fait que la bibliothèque dynamique cygwin.dll est fournie sous licence GNU/GPL, et que toute utilisation de cette bibliothèque vous forcerait à... créer des applications sous licence GNU/GPL.

Bien qu'étant personnellement fort attaché à la licence GNU/GPL, je peux concevoir que ce ne soit pas le cas de tout le monde. C'est la raison pour laquelle j'ai décidé de fournir un compilateur natif qui vous permette de choisir votre propre licence selon vos propres goûts en la matière.

I-F - Deux passes de compilations

L'un des principaux problèmes rencontrés lorsque l'on souhaite recompiler un compilateur et les différents outils qui s'y rattachent (assembleur, éditeur de liens, ...) est que nous avons à gérer des dépendances croisées entre ces outils: Il est impossible de compiler l'assembleur ou l'éditeur de liens sans avoir de compilateur, mais le compilateur a besoin de l'assembleur et de l'éditeur de liens pour fournir un exécutable fini.

Et, tant qu'à faire, il est intéressant de disposer des outils compilés avec la version du compilateur utilisé ainsi que du compilateur créé grâce aux outils utilisés.

Le problème est absent si l'on souhaite simplement optimiser les outils pour une architecture donnée, mais, si l'on souhaite compiler une autre version, il se pose clairement.

C'est la raison pour laquelle la méthode que je vous propose va donc effectuer la compilation de certains éléments deux temps (six mouvements en tout):

- Installer une version utilisable du compilateur et des outils
- Compiler les outils indispensables (assembleur, éditeur de liens, ...) avec le compilateur fonctionnel
- Compiler les bibliothèques indispensables en veillant à utiliser les outils fraîchement compilés
- Compiler le compilateur en utilisant les outils indispensables fraîchement compilés. Le système de compilation de GCC fera qu'il sera compilé plusieurs fois, et donc, il sera au final compilé automatiquement avec les versions correctes.
- Compiler une deuxième fois bibliothèques dépendant du système avec ces premières compilation des outils
- Recompiler les outils indispensables pour qu'ils le soient en utilisant la dernière version du compilateur, des outils et des bibliothèques dépendantes du système

II - Le "minimum compilant"

Par "minimum compilant", j'entend parler du minimum qui permet d'assurer une compilation réussie du système à créer.

Il faut, bien sûr, disposer du minimum permettant de compiler un code aussi simple que

```
#include <stdio.h>
int main()
{
    printf("hello world"\n);
    return 0;
}
```

mais il faut aussi l'ensemble des bibliothèques et des outils nécessaires à la compilation de l'ensemble du système.

Cependant, si vous disposez déjà d'une version fonctionnelle de GCC, vous pouvez bien sûr vous contenter de renommer le dossier dans lequel elle est installée et utiliser votre propre version, en ajoutant au besoin seulement quelques outils bien particuliers.

II-A - GCC-core

GCC-core est l'archive qui contient le compilateur C de GCC. Toute la collection, ou presque, et les outils utilisés sont compilables à partir du moment où l'on dispose d'un compilateur C.

II-B - GCC-ada

Ada est le seul compilateur de la collection GCC qui nécessite de disposer... du compilateur Ada pour en permettre la compilation.

Vous pouvez bien sûr décider de ne pas l'installer si vous ne prévoyez pas de compiler le support de ce langage.

II-C - Binutils

Binutils est une collections d'outils qui fournit tout ce qui est nécessaire à la gestion et à l'utilisation correctes de fichiers binaires (exécutables, bibliothèques dynamiques et statiques...).

Parmi ceux-ci, on trouve

- l'archiveur **ar**
- l'éditeur de liens **ld**
- l'assembleur **as**
- les outils de gestion de bibliothèques dynamiques **dllwrap** et **dlltools**
- les outils de génération et de gestion de fichiers objets **objcopy** et **objdump**
- ...

II-D - Bison

Bison est un générateur universel d'analyseurs lexicaux qui convertit une grammaire sans contexte annotée en analyseur de LALR (1) ou de GLR (2) pour cette grammaire.

Il est indispensable pour permettre la compilation de binutils.

II-E - Flex

(F)Flex est un analyseur syntaxique souvent utilisé en conjonction avec bison.

II-F - Libintl

Libintl est une bibliothèque qui fournit un support des langues étrangères (par rapport à l'anglais qui est la langue "parlée" par les outils), afin de permettre à l'utilisateur d'utiliser sa propre langue maternelle (Native Language Support).

Bison dépend explicitement de cette bibliothèque.

II-G - Libiconv

Libiconv est une bibliothèque qui permet de supporter les différentes tables de caractères, allant de la table ANSI/ASCII aux différentes tables iso8892 et utf.

Elle est aussi une dépendance explicite de bison.

II-H - W32api

Microsoft fournit un ensemble de fichiers d'en-tetes et de bibliothèques pour programmer des applications sous Windows (le Platform SDK) au moyen de son compilateur VC++.

W32Api est une adaptation des headers de Microsoft pour être utilisables avec GCC et fournit également certaines bibliothèques fréquemment utilisées telles que OpenGL ou DirectX.

II-I - Mingw-runtime

mingw-runtime, enfin, est l'implémentation, basée sur les appels windows au lieu des appels posix/linux, de la bibliothèque standard du C.

II-J - Texinfo

Texinfo est un utilitaire qui permet de générer automatiquement des fichiers d'aide en différents formats(latex, info, pdf...).

Comme la commande d'installation finale sort systématiquement sur erreur si elle n'arrive pas à créer les fichiers *.info pour certaines parties, et bien qu'il s'agisse d'un format de fichier d'aide principalement utilisé sous linux, nous nous trouvons face à l'obligation d'installer cette collection d'outils de manière temporaire.

Sur le miroir sourceforge du projet gnuwin32, vous trouverez en cherchant cette collection une archive nommée texinfo-dep, qu'il ne vous est pas utile de prendre, étant donné qu'il ne s'agit ni plus ni moins que des dlls de libintl et libiconv.

II-K - Où les trouver?

Vous pouvez trouver des archives contenant les binaires propres à MinGW (GCC-core, GCC-ada et binutils, w32api et mingw-runtime) directement **sur le site du projet MinGW** ainsi que **sur l'un des miroirs de Sourceforge relatifs à MinGW**.

Pour ce qui est des binaires de Bison, Flex, libintl, libiconv et texinfo, vous les trouverez sur **l'un des miroirs Sourceforge du projet GnuWin32**.

Soyez attentif, au moment de choisir l'archive que vous téléchargez, au nom des archives présentes: il y a généralement deux noms forts semblables, l'un contenant un élément "-src-" dans son nom et l'autre non.

Il faut choisir l'archive qui ne contient pas cet élément "-src-" sous peine... de vous retrouver avec les sources, qui ne vous serviront à rien ici.

préférez la version *.zip de bison, flex, libintl et libiconv à la version exécutable, car cette dernière contient un assistant d'installation qui les placera par défaut dans c:Program Files et qui, si j'ai bon souvenir, placera un raccourci dans le menu démarrer pour chacun d'eux...

II-L - Installer tout cela

Les archives que vous trouverez de GCC-core, de GCC-ada et de binutils pour MinGW sont au format linux, c'est à dire qu'elles ont été créées au format *.tar.gz ou *.tar.bz2.


Vous trouverez facilement des interfaces graphiques permettant de gérer ces formats sous windows telles que **winrar**, **7zip**, **alzip** ou **tugzip**

Cependant, si l'idée d'installer l'un ou l'autre de ces archiveurs ne vous plait pas, vous pouvez aussi choisir d'attendre d'avoir installé MSYS dans la section suivante pour les décompresser directement dans un environnement linux.

Pour ce qui est des archives de bison, flex, libintl et libiconv, vous devriez les trouver au format *.zip, et ne devraient pas poser de problème.

Je vous conseillerais donc d'utiliser votre gestionnaire d'archives préféré pour les extraire, MSYS étant vraiment minimal et nécessitant l'ajout d'un utilitaire nommé unzip pour extraire les fichiers de ce genre d'archives.

Pour les installer, il suffit de décompresser l'ensemble de ces archives dans un dossier unique, de préférence, en racine de disque dur, de n'importe quel nom. Par facilité, quand je parlerai de cette installation particulière, j'y ferai référence comme étant l'installation temporaire, et je considérerai qu'elle a été effectuée dans le dossier c:\gcc.

 *Comme toutes les archives contiennent sensiblement les mêmes sous dossiers (bin, include, lib, ...), il vous sera vraisemblablement demandé à chaque fois si vous souhaitez réellement déplacer ou copier ces dossiers. Vous pouvez cliquer sans crainte sur "tous", car aucun fichier ne se trouve dans deux archives en même temps.*

Il reste deux précautions importantes pour la suite des événements à prendre avant de quitter définitivement le dossier gcc:

La première est que l'extraction de Bison a placé un exécutable nommé M4.exe dans le dossier bin. Il se fait que nous installerons un émulateur linux à la page suivante et que cet émulateur dispose lui aussi d'un exécutable M4. Afin d'éviter les conflits, il faut donc supprimer M4.exe du dossier bin.

La deuxième précaution nous vient aussi de l'installation de bison. Bison n'est en définitive qu'une évolution de yacc.

Nous avons donc un fichier sans extension nommé yacc dans le dossier bin, qui n'est rien d'autre qu'un script linux permettant d'appeler bison quand on rencontre une instruction yacc, histoire de maintenir la compatibilité avec les codes sources plus anciens.

Malheureusement, et vous vous en rendrez rapidement compte si vous ouvrez le fichier avec n'importe quel éditeur de texte, le chemin d'accès à bison fait référence à un dossier nommé Bison et se trouvant dans le Program Files, ce qui ne correspond nullement à notre réalité. Il faut donc modifier de

```
#!/bin/sh
exec c:/progra-1/Bison/bin/bison -y "$@"
```

en

```
#!/bin/sh
exec c:/gcc/bin/bison -y "$@"
```

pour que tout rentre dans l'ordre.

III - Un système linux sous windows

GCC est une collection d'outils venant tout droit du monde de linux. A ce titre, il est particulièrement attaché aux outils GNU pour sa compilation.

Je ne prétend pas qu'il est impossible de le compiler sans les outils GNU prévus à cet effet (M4, make, autoconf...) mais je prétend réellement que le plus facile reste quand même d'utiliser ces outils.

L'idéal est donc de disposer d'un système qui puisse émuler linux alors que nous resterons bien au chaud sous windows. L'outil que je vous propose d'utiliser pour ce faire est fourni par l'équipe de MinGW et s'appelle MSYS.

III-A - Installer MSYS

MSYS est l'abréviation de Minimal SYStem, et va vous permettre de disposer de l'ensemble des outils qui vous permettront d'effectuer la compilation.

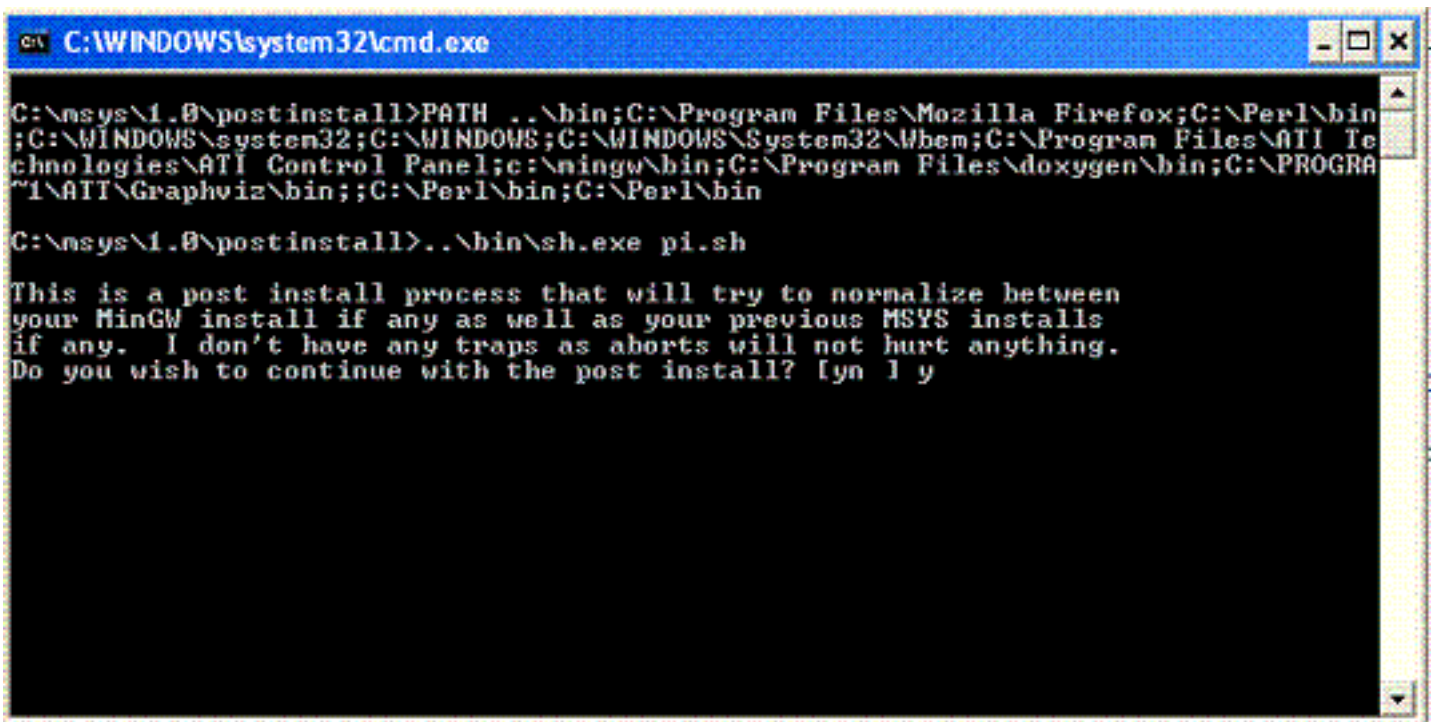
Vous pouvez le télécharger librement sur le site de MinGW ou sur l'un des miroirs mis à disposition par Sourceforge.

Le plus facile est sans doute de choisir l'installateur (msys.exe) et de garder les réglages par défaut.

Il s'installera par défaut sur c:\msys, et cela nous ira très bien.

Pour la facilité, je considérerai par la suite que l'installation de msys a été faite dans c:\msys.

Une fois l'installation finie (cela ne prend que peu de temps), il vous proposera de procéder à la configuration post-insallation, acceptez sans crainte:



```
C:\WINDOWS\system32\cmd.exe
C:\msys\1.0\postinstall>PATH ..\bin;C:\Program Files\Mozilla Firefox;C:\Perl\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Program Files\ATI Technologies\ATI Control Panel;c:\mingw\bin;C:\Program Files\doxygen\bin;C:\PROGRAM~1\ATI\Graphviz\bin;;C:\Perl\bin;C:\Perl\bin
C:\msys\1.0\postinstall>.\bin\sh.exe pi.sh
This is a post install process that will try to normalize between
your MinGW install if any as well as your previous MSYS installs
if any. I don't have any traps as aborts will not hurt anything.
Do you wish to continue with the post install? [yn] y
```

Il vous demandera alors si vous avez MinGW installé.

```

C:\WINDOWS\system32\cmd.exe

C:\msys\1.0\postinstall>PATH .\bin;C:\Program Files\Mozilla Firefox;C:\Perl\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Program Files\ATI Technologies\ATI Control Panel;c:\mingw\bin;C:\Program Files\doxygen\bin;C:\PROGRAM FILES\ATI\Graphviz\bin;;C:\Perl\bin;C:\Perl\bin

C:\msys\1.0\postinstall>..\bin\sh.exe pi.sh

This is a post install process that will try to normalize between your MinGW install if any as well as your previous MSYS installs if any. I don't have any traps as aborts will not hurt anything. Do you wish to continue with the post install? [yn ] y

Do you have MinGW installed? [yn ] n_
    
```

Bien que nous venions d'installer MinGW dans le dossier c:\gcc, le but de cette question est en fait de faire correspondre un dossier qui sera accessible sous MSYS, et que nous utiliserons comme cible de notre nouvelle installation, au dossier où MinGW serait installé. Comme cela pourrait provoquer quelques conflits pour la suite des événements, répondons non.

Le script se fendra alors d'un petit conseil concernant l'installation de MinGW:

```

C:\WINDOWS\system32\cmd.exe

C:\msys\1.0\postinstall>PATH .\bin;C:\Program Files\Mozilla Firefox;C:\Perl\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Program Files\ATI Technologies\ATI Control Panel;c:\mingw\bin;C:\Program Files\doxygen\bin;C:\PROGRAM FILES\ATI\Graphviz\bin;;C:\Perl\bin;C:\Perl\bin

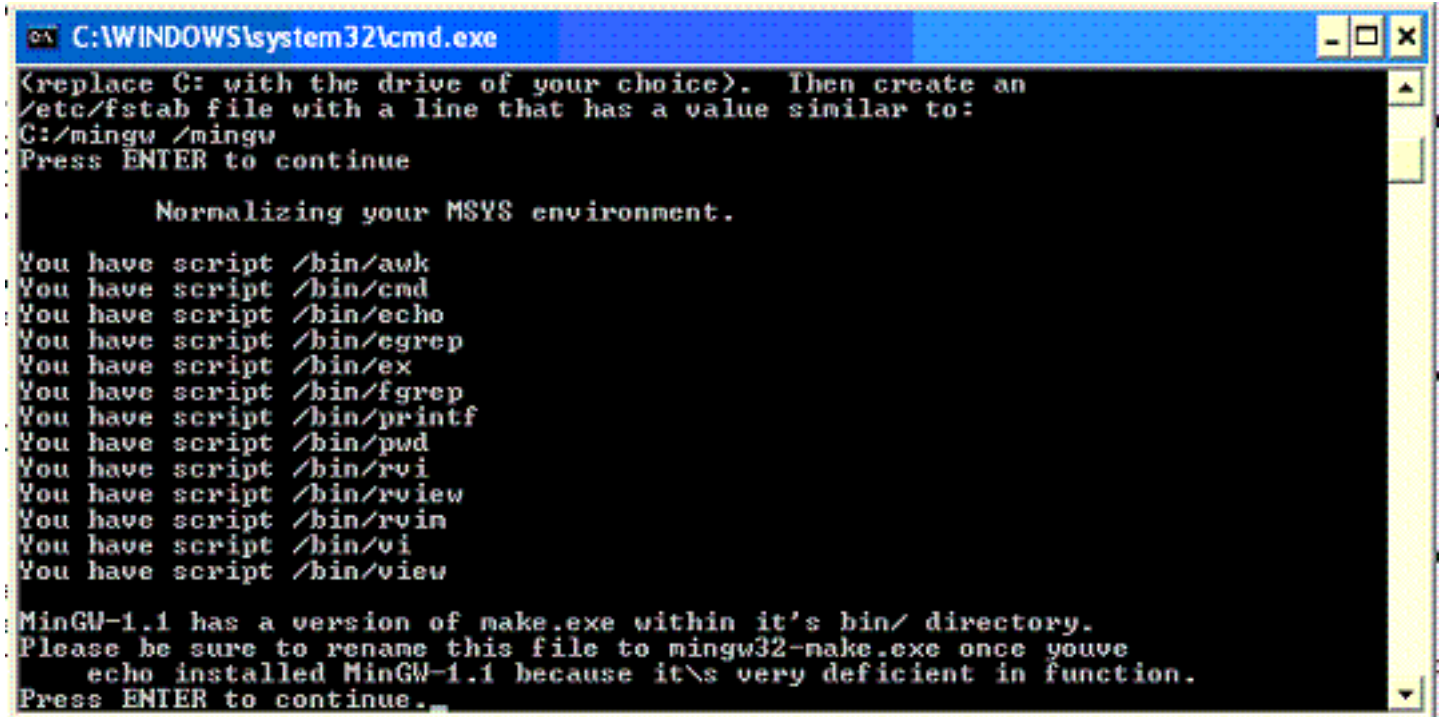
C:\msys\1.0\postinstall>..\bin\sh.exe pi.sh

This is a post install process that will try to normalize between your MinGW install if any as well as your previous MSYS installs if any. I don't have any traps as aborts will not hurt anything. Do you wish to continue with the post install? [yn ] y

Do you have MinGW installed? [yn ] n

When you install MinGW I suggest you install it to C:/mingw (replace C: with the drive of your choice). Then create an /etc/fstab file with a line that has a value similar to:
C:/mingw /mingw
Press ENTER to continue _
    
```

Et attendra que vous enfoncez la touche <ENTER> avant de vérifier la présence de quelques fonctions importante pour Msys.



```

C:\WINDOWS\system32\cmd.exe
(replace C: with the drive of your choice). Then create an
/etc/fstab file with a line that has a value similar to:
C:/mingw /mingw
Press ENTER to continue

    Normalizing your MSYS environment.

You have script /bin/awk
You have script /bin/cmd
You have script /bin/echo
You have script /bin/egrep
You have script /bin/ex
You have script /bin/fgrep
You have script /bin/printf
You have script /bin/pwd
You have script /bin/rvi
You have script /bin/rview
You have script /bin/rvin
You have script /bin/vi
You have script /bin/view

MinGW-1.1 has a version of make.exe within it's bin/ directory.
Please be sure to rename this file to mingw32-make.exe once youve
echo installed MinGW-1.1 because it's very deficient in function.
Press ENTER to continue.
  
```

Il terminera sur un conseil concernant miwngw32-make, auquel on peut ne pas être trop attentif pour l'instant, étant donné que notre but est de créer une installation "from scratch" de GCC. Deux appuis supplémentaires sur la touche <ENTER> permettront de quitter le script de post installation.

III-B - Premiers contacts

III-B-1 - La console sous linux

Ceux qui se sont déjà frottés à linux, ou au monde Unix en général peuvent passer cette sous-section sans crainte, rien de nouveau n'apparaîtra dans ces lignes.

Pour les autres, ne vous en faites pas, il n'est nullement dans mes intentions de vous donner un cours complet sur linux - il faudrait plusieurs tutoriaux pour mener ce projet à terme - et ce n'est sûrement pas le but de celui-ci.

Le but de cette sous-section est, tout simplement, de vous signaler quelques différence notables entre ce que vous connaissez des commandes "DOS" et leur équivalents linux.

- Ainsi, les paramètres passés au commandes sous DOS sont précédées du /, alors qu'ils sont précédés d'un trait d'union - pour les versions courtes ou de deux traits d'union -- pour les versions longues.
- La très grosse majorité des commandes contient une aide en ligne. Pour y arriver, il suffit de faire suivre le nom de la commande par --help (ou -h si vous êtes fainéants)
- La console linux dispose d'un système d'auto-complétion très efficace; il permet au choix de compléter le nom d'un dossier, d'un fichier, ou d'une commande. Pour en profiter, il suffit d'appuyer sur la touche <tabulation>. Le premier appuis dessus fera compléter le mot jusqu'à ce qu'il y ait "conflit": dans le cas de deux mots commençant par les mêmes lettres, la complétion s'arrêtera juste avant la première lettre qui n'est plus

identique. Le deuxième appuis affichera la liste des (dossiers, fichiers, instructions) qui correspond à ce qui a déjà été introduit.

- Linux utilise le "slash"(/) comme séparateur entre les dossiers et les fichiers et supporte "relativement mal" les espaces dans les noms. Il est donc "de bon ton" d'essayer autant que possible de les éviter. Pour accéder à un dossier ou à un fichier contenant des espaces, il faut faire précéder chaque espace d'un "back-slash" (\). Ceci dit, l'auto-complétion le rajoute automatiquement.
- La notion de "disque logique" est inconnue sous linux.

Les partitions sont tout simplement rattachées à des dossiers qui se trouvent dans l'arborescence partant de la racine du système.

Et la racine du système ne se trouve pas forcément... à la racine d'une partition.

Ainsi, quand vous aurez lancé MSYS, la racine du système (/) se trouvera en réalité dans c:\msys\1.0, et si vous voulez quelque chose qui se trouve "au dessus" de c:\msys\1.0, il faudra y accéder au départ de / (ex: /c/mingw, /c/downloads)

- Alors que windows se base sur l'extension d'un fichier pour déterminer si c'est un exécutable ou non, linux regarde un bit particulier qui lui indique qu'il s'agit d'un exécutable. Cependant, étant donné que MSYS doit fonctionner sous windows, il est également sensible aux extensions exe et autres.
- Linux est "sensible à la casse" en ce qui concerne les noms de dossiers, de fichiers et de commandes. Ainsi, si vous souhaitez accéder au dossier c:\Program Files, il faudra écrire un P majuscule et non un p minuscule, autrement, même l'auto-complétion ne trouvera pas le dossier.
- Il y a une série d'instructions qui diffèrent sous DOS et sous linux.

Il y a d'ailleurs une série d'instructions qui sont propres à linux. La liste des différences étant relativement longue, et comme vous n'aurez sans doute pas besoin d'une bonne partie d'entre elles, je me contenterai de vous expliquer les commandes nécessaires en temps utiles.

- Linux dispose d'un historique des commandes particulièrement efficace. Vous pouvez naviguer dans cette historique à l'aide des fleches de directions "haut" et "bas".

Il n'y a aucune limite quant au nombre d'instructions de cette historique, et quand bien même vous quitteriez la console, l'historique est maintenue au prochain lancement.

III-B-2 - Derniers préparatifs

Avant de lancer MSYS, nous allons nous assurer une dernière fois que tout est prêt:

Si vous n'avez pas encore créé un dossier MinGW à la racine de c:\, c'est le moment de le faire.

Linux utilise un fichier nommé fstab (**F**ile **S**ystem **T**able) qui nous sera très utile.

dans le dossier c:/msys/1.0/etc, vous en trouverez un exemple nommé fstab.sample. Nous allons nous en servir.

Comme il s'agit d'un simple fichier texte, vous pouvez demander à l'ouvrir avec n'importe quel éditeur de texte que vous appréciez. Gardez cependant en tête le fait que c'est un fichier créé sous linux... Cela signifie, entre autre, que les caractères de retour à la ligne sont représenté sous la forme CR, alors que sous windows, on utilise généralement la forme CRLF... Le résultat en est que, si votre éditeur de texte ne comprend pas cette représentation, vous verrez l'ensemble du texte sur une seule ligne, avec une série de carrés qui auraient dû être considérés comme des saut de ligne.

Au pire, vous pouvez toujours décider d'utiliser WordPad (l'éditeur de texte RTF installé d'origine sous windows), qui vous le présentera de manière correcte.

contenu de fstab.sample

```
#fstab.sample
#This is a sample file for /etc/fstab.
#Currently /etc/fstab is only read during dll initialization.
#I will eventually watch the directory for changes and reread the file.
#The line format is simple in that you give the Win32 path, followed by one or
#more space or tab delimiter, followed by the mount point. Mount points in
#typical UNIX environments must be a physical name on a drive before it can
#actually be used as a mount point. In this implementation the "must exist"
#requirement isn't enforced, however, it will be an aide to such programs as
#find and readline's tab completion if it does exist.

#You can use a # as the first character on the line as a comment indicator.
#Blank lines are ignored.

#Win32_Path Mount_Point
c:/mingw /mingw
c:/ActiveState/perl /perl
```

Comme vous l'aurez sûrement remarqué, les lignes commençant par un # sont considérées comme des commentaires.

La ligne qui nous importe surtout est la ligne

```
c:/mingw /mingw
```

Il faut veiller à ce que la première partie (c:/mingw) corresponde au dossier que vous avez prévu pour l'installation de la version compilée par vos soins.

Si vous disposez de perl, vous pouvez modifier la dernière ligne du fichier (celle qui commence par c:/ActivePerl/perl) pour lui permettre de refléter le dossier dans lequel perl est installé. Si vous n'avez pas perl, car ce n'est pas obligatoire pour notre usage, vous pouvez commenter la ligne en plaçant un # à son début.

Il ne vous reste plus qu'à enregistrer les modifications en renommant le fichier de fstab.sample en fstab (dans le dossier c:\msys\1.0\etc).

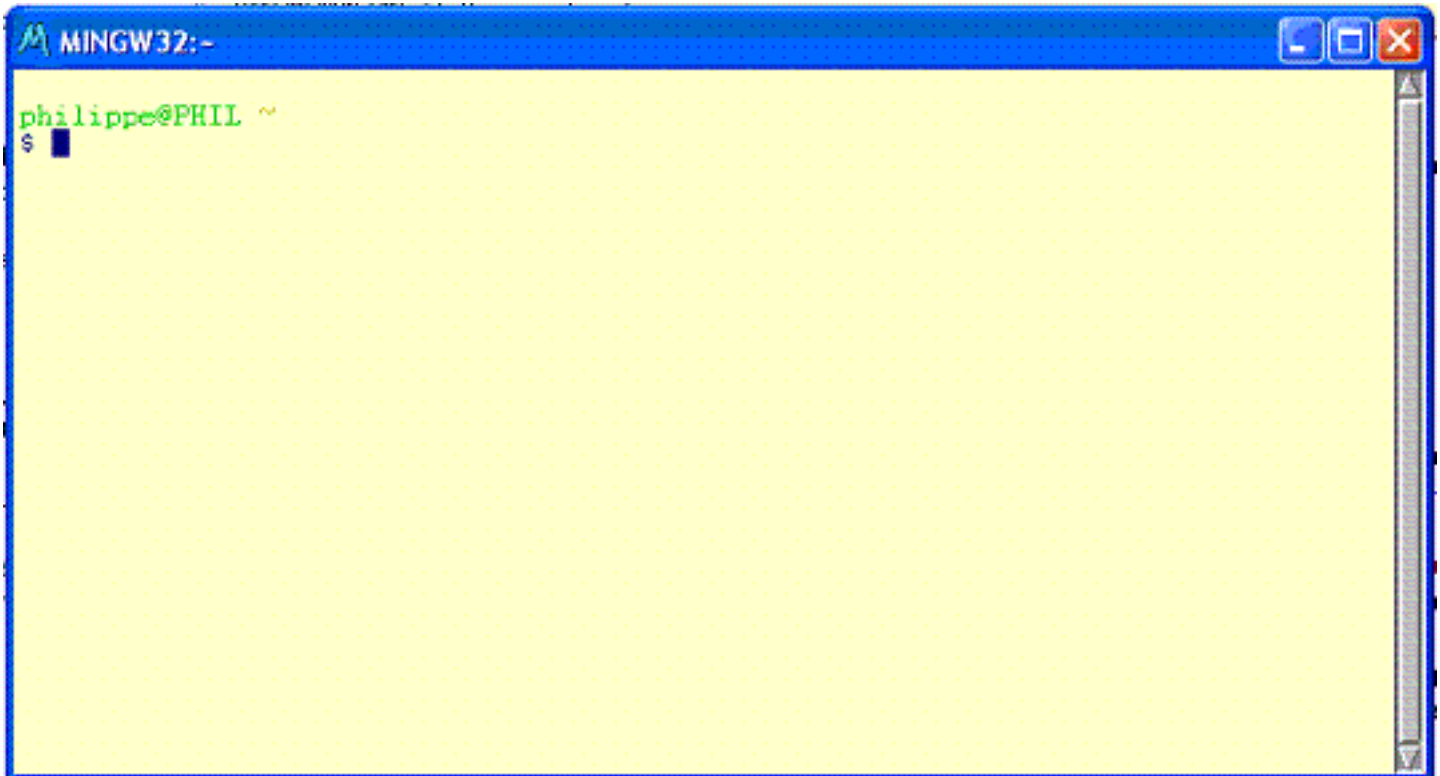
Vérifiez bien que windows ne vous rajoute pas l'extension .txt, car, sinon, il ne sera pas pris en compte

Si c'est le cas, renommez le en supprimant l'extension et acceptez l'avertissement que windows ne manquera pas de vous donner.

III-C - Bienvenue sous linux

Il est maintenant temps de lancer MSYS et d'entrer pour la première fois sous cet environnement (normalement il aura installé un raccourci dans le menu démarrer->tous les programmes->MinGW->Msys)


Vous obtiendrez une console ressemblant à



Cette console vous fournit une quantité non négligeable d'informations qui sous la forme de

```
UserName@server ~  
$
```

- **Username** représente votre nom d'utilisateur
- **server** représente le nom du serveur sur lequel vous êtes connecté, et qui est typiquement, ici, le nom de l'ordinateur
- **~** représente le dossier actuel (ce symbole indique que vous êtes dans votre dossier personnel)
- **\$** indique que vous êtes logué en tant qu'utilisateur simple (si vous aviez été logué en tant qu'administrateur, cela aurait été un #). Le fait que vous *travailliez éventuellement avec une session administrateur sous windows n'intervient absolument pas ici, mais, comme les droits accordés à l'utilisateur simple nous suffisent, il n'y a pas lieu de s'inquiéter*

 *MSYS ne connaît pas la commande su, et ne dispose d'ailleurs d'aucun des fichiers de gestion de droits que l'on retrouve habituellement sous linux (/etc/passwd, /etc/group...). Il est donc totalement impossible d'envisager de se loguer en tant que "root", mais on dispose malgré tout de l'ensemble des droits. (quand on vous dit que c'est un système minimal :D)*

III-D - Créer les dossiers de travail

Par facilité, nous allons créer un dossier de travail qui contiendra lui-même un dossier dans lequel iront les sources décompressées et un autre dans lequel nous effectuerons les compilations avec la commande

```
$ mkdir -p /work/build /work/src
```

En effet, la plupart des outils supportent difficilement d'être compilés directement dans le dossier dans lequel se trouvent leur sources. Cela a aussi l'avantage de laisser le dossier dans lequel se trouve les sources "en l'état", et donc d'éviter des manipulations hasardeuses si on veut reprendre les choses depuis le début.

III-E - Pour ceux qui ont postposé la décompression des outils temporaires

Si vous avez décidé d'attendre d'être sous linux pour décompresser binutils et GCC, c'est le moment où jamais de le faire.

En considérant que vos téléchargements sont placés dans le dossier c:\downloads, cela se fait avec les commandes

```
$ tar -vzxvf /c/downloads/gcc-core-3.4.5-20060117-1.tar.gz -C /c/gcc
$ tar -vzxvf /c/downloads/gcc-ada-3.4.5-20060117-1.tar.gz -C /c/gcc
$ tar -vzxvf /c/downloads/binutils-2.17.50-20060824-1.tar.gz -C /c/gcc
$ tar -vzxvf /c/downloads/mingw32-runtime-3.13.tar.gz -C /c/gcc
$ tar -vzxvf /c/downloads/win32api-3.10.tar.gz -C /c/gcc
```

Comprendre la commande

Tar est un utilitaire permettant de décompresser les archives linux créées avec gzip et bzip2.

Les arguments qui nous intéressent sont

- **v** verbose: produit une sortie sur console de l'ensemble des fichiers extraits ou ajoutés
- **z** permet de spécifier le format de l'archive : z pour les archive *.tar.gz ou j pour les archives *.tar.bz2
- **x** indique que l'on veut pratiquer l'extraction, et non l'ajout de document
- **f** force: indique qu'il faut forcer l'extraction du fichier s'il existe déjà

Enfin, l'argument séparé **-C** permet de spécifier un chemin de base où effectuer l'extraction (chemins relatifs admis).

III-F - L'arborescence finale

A ce stade, je considérerai que vous disposez d'une arborescence ressemblant à ceci:

```
c:
|
| downloads      <- dossier de téléchargements
| gcc            <- dossier de l'installation temporaire
|   |
|   | bin
|   | etc
|   | lib
|   | ...
|   MinGW       <- dossier de destination finale
|   msys
|   | 1.0
|   |   |
|   |   | bin      :
|   |   | etc      : <-dossiers d'installation de linux
|   |   | lib      :
|   |   | ...      :
|   |   | work
|   |   |   | build <- dossier de compilation
|   |   |   | src  <- dossier contenant les sources décompressées
```

III-G - Convention pour les codes

Une série d'instructions seront présentées dans les lignes suivantes.

Afin d'éviter toute confusion, il me semble intéressant de présenter la convention que j'ai utilisé:

Une commande commence par \$. Ce \$ représente l'invite de commande et ne doit pas être écrit.

```
$ une ligne de commande
```

Si une commande est écrite sur plusieurs lignes, chaque ligne termine par un \ qui permet à la console de savoir que la ligne continue plus loin. La ligne suivante ne commence donc pas par \$

```
$ une commande \  
sur plusieurs lignes
```


Vous pouvez décider de ne pas tenir compte de ce signe, et continuer sur la même ligne, l'interpréteur de commande passera automatiquement à la ligne en cas de besoin.

Cependant, si vous décidez de passer à la ligne (par appuis sur la touche <Enter>) vous **devez** mettre ce symbole avant le passage à la ligne.

MSYS supporte le mécanisme de copier/coller: Pour coller un texte dans la console MSYS, ce qui sera le sens le plus courant que vous utiliserez, cliquez simplement avec le bouton gauche de la souris sur celle-ci en maintenant la touche <majuscule> enfoncée.

De cette manière, il vous suffira de copier les commandes en évitant le \$ de l'invite, mais en prenant toutes les lignes si une commande est sur plusieurs lignes, et de les coller directement dans la console.

Vous pourrez alors vous déplacer dans cette commande à l'aide des flèches gauche et droite afin de modifier éventuellement un paramètre.

 *En cas de commande multiligne, vous ne pourrez vous déplacer que sur la dernière ligne de celle-ci.*

Si le chemin d'accès ne correspond pas, vous aurez simplement un avertissement spécifiant que la commande n'est pas trouvée, et vous pouvez donc confirmer la commande par appuis sur la touche <Enter>.

Si le chemin correspond mais que vous souhaitez modifier un paramètre, vous pouvez annuler une commande à tout moment simplement en appuyant simultanément sur la touche <Ctrl> et sur C.

Vous pourrez alors vous servir de l'historique qui vous présentera la commande considérée comme étant sur une seule ligne.

III-H - Configurer le PATH

Afin de pouvoir utiliser notre installation temporaire de GCC, il y a lieu de configurer `c:\gcc\bin` comme étant un chemin dans lequel aller chercher les exécutables.

Cela se fait avec la commande

```
$ PATH=/c/gcc/bin:$PATH
```

Vous pourrez ensuite vérifier si vous ne vous êtes pas trompé en invoquant la commande

```
$ gcc -v
```

qui devrait vous sortir quelque chose ressemblant à

```
Reading specs from c:/gcc/bin/./lib/gcc/mingw32/3.4.5/specs
Configured with: ../gcc-3.4.5/configure --with-gcc --with-gnu-ld --with-gnu-as
--target=mingw32 --prefix=/mingw --enable-threads --disable-nls
--enable-languages=c,c++,f77,ada,objc,java --disable-win32-registry
--disable-shared --enable-sjlj-exceptions --enable-libgcj --disable-java-awt
--without-x --enable-java-gc= Boehm --disable-libgcj-debug --enable-interpreter
--enable-hash-synchronization --enable-libstdcxx-debug
Thread model: win32
gcc version 3.4.5 (mingw special)
```

IV - Les version testées

Ainsi que je l'ai signalé, les numéros de version sont susceptibles d'évoluer très rapidement, ne serait-ce que du côté des "snapshot" étant donné qu'il en sort un par semaine.

S'il n'est pas impossible que la version 4.3 de GCC ne sorte qu'après la finalisation de la norme C++0x, bien que je ne puisse le jurer, il est donc aussi tout à fait possible que la version stable numérotée 4.2 évolue encore d'ici la stabilisation de la version 4.3.


Les numéros de version et les instructions de compilation que je fournis dans ce tutorial sont donc à prendre dans le sens où c'est ce qui fonctionnait au moment de la rédaction.

IV-A - Binutils-2.17

La version reconnue comme stable de binutils au moment d'écrire ces lignes est la version 2-17. Cependant, j'ai réussi à compiler la version "snapshot" 2.18.50 sans aucun problème.

IV-B - GCC-4.2.1

La dernière version stable de GCC au moment d'écrire ces lignes est la version 4.2.1.

 *Si vous voulez activer le support de Ada, il y a une procédure à suivre pour en assurer la compilation..*

Elle vous sera expliquée plus loin.

V - La première passe

V-A - Compiler binutils

Entrez dans le dossier build avec la commande

```
$ cd /work/build
```

et décompressez les sources de binutils dans un sous dossier de src avec la commande

```
$ tar -vxjf /c/downloads/binutils-2.17.tar.bz2 -C /work/src
```

Créez un dossier de compilation pour binutils et entrez dedans avec la commande

```
$ mkdir binutils && cd binutils
```

Invoquez la configuration de binutils avec la commande

```
$ ../../src/binutils-2.17/configure --prefix=/mingw --target=i686-pc-mingw32 \  
--program-prefix="" \  
--with-lib-path=/mingw/mingw32/lib:/mingw/lib:/usr/local/lib:/lib:/usr/lib \  
--with-gcc --with-gnu-as --with-gnu-ld --disable-nls --disable-shared
```

Comprendre la commande

- **--prefix=/mingw** indique la racine du chemin d'installation
- **--target=i686-pc-mingw32** spécifie la cible pour laquelle la compilation est effectuée
- **--program-prefix=""** indique qu'il ne faut rien rajouter au début des noms d'exécutables
- **--with-lib-path** fournit les dossiers dans lesquels ld devra s'attendre à trouver les différentes bibliothèques
- **--disable-shared** signale que les exécutables doivent être créés sans utiliser les bibliothèques partagées
- **--with-gcc** indique que l'on va utiliser GCC pour la compilation
- **--with-gnu-as** et **--with-gnu-ld** indiquent que l'on va utiliser les outils (assembleur et éditeur de liens) gnu

Comprendre l'option --target=

La syntaxe générale des options --target= (et --target= qui permet de configurer la compilation croisée) est <type processeur>-<type d'ordinateur>-<systeme>

La syntaxe est adaptée à mon cas car je dispose à ce jour d'un processeur Athlon-xp (équivalent Pentium III), et il faudra donc penser à l'adapter à votre configuration personnelle.

Pour le type de processeurs sur PC, cela peut varier, selon votre configuration personnelle, parmi les valeurs suivantes:

- **i386**: pour les processeurs de type 3x86

- **i486**: pour les processeurs de type 4x68
- **i586**: pour les processeurs de type Pentium I (et équivalents)
- **i686**: pour les processeurs de type Pentium II (et équivalents)
- **i786/i886/i986**: pour les processeurs de type supérieurs
- **IA-64**: pour les processeurs 64 bits simples et multi-core de chez intel ("Itanium Processor Family")
- **x86_64** (ou **amd64**): pour les processeurs 64 bits simples et multi-core de chez AMD

Le type d'ordinateur étant ici un PC, c'est celui que l'on utilisera le plus souvent...

Cependant, il est possible de compiler les outils GNU sur une longue série d'autres ordinateurs, allant du solaris au powerpc, en passant de manière non exhaustive par le HP-HUX ou les systèmes SCO.

Enfin, le système sera, avec les PC's, principalement mingw32 (==windows) et linux

 Vous trouverez la liste complètes des spécifications d'hôte et de cible  **directement sur le site de GCC**

Il ne semble pas à l'ordre du jour de l'équipe de MinGW d'envisager d'évoluer vers une solution 64 bits.

Il ne reste plus qu'à invoquer la compilation:

```
$ make CFLAGS="-O2 -D__USE_CRTIMP -fno-exceptions" LDFLAGS=-s
```

et l'installation

```
$ make install
```

Enfin, il est temps de faire le ménage pour la deuxième passe avec la commande

```
$ make clean
```

V-B - Compiler les bibliothèques indispensables

Lorsque nous essayerons de compiler GCC, nous nous heurterons à un problème, certes mineur, mais qu'il vaut mieux anticiper juste après avoir compilé binutils.

En effet, GCC sera compilé, par défaut trois fois: une première fois pour obtenir une version compilée et utilisable de GCC, la deuxième fois pour s'assurer que GCC ait été compilé... en utilisant la même version de GCC. et une troisième fois pour vérifier que tout a bien fonctionné.

Toute l'astuce réside dans le fait que, pour la deuxième et la troisième compilation, il cherchera les fichiers d'en-têtes et les bibliothèques... dans le dossier dans lequel il devra s'installer.

Il faut donc veiller à ce que les en-têtes et les bibliothèques se trouvent au bon endroit quand GCC en aura besoin, et, tant qu'à faire, autant qu'il dispose d'une version qui aura été compilée en utilisant l'assembleur et l'éditeur de liens avec lequel nous lui demanderons de travailler dorénavant.

V-B-1 - Compiler w32-api

retournez dans le dossier général de compilation et décompressez l'archive des sources dans le dossier source avec les commandes

```
$ cd /work/build
$ tar -vxjf /c/downloads/w32api-3.10.1-src-tar.bz2 -C /work/src
```

Comme la compilation de mingw-runtime cherchera un dossier avec les source de w32-api, sans information de version, renommons directement le dossier en vue de satisfaire la runtime de mingw avec la commande

```
$ mv /work/src/w32api-3.10-1 /work/src/w32api
```

il n'y a plus qu'à créer un dossier nommé w32api, y entrer et à effectuer la configuration, la compilation et l'installation avec les commandes

```
$ mkdir w32api && cd w32api
$ ../../src/w32api/configure --prefix=/mingw --target=i686-pc-mingw32 \
--with-as=/mingw/bin/as.exe --with-ld=/mingw/bin/ld.exe
$ make
$ make install
```

les options **--with-ld=/mingw/bin/ld.exe** et **--with-as=/mingw/bin/as.exe** permettent de spécifier l'éditeur de lien et l'assembleur à utiliser, et plus particulièrement, de dire qu'il faut utiliser la version que nous venons de compiler et d'installer.

Terminons en nettoyant le dossier en vue de la deuxième passe

```
$ make clean
```

L'avantage de la commande clean est qu'elle supprime les fichiers générés, mais laisse les Makefile et les éventuelles copies de fichiers source en place, ce qui nous permettra de ne pas devoir refaire la configuration lors de la deuxième passe

V-B-2 - compiler mingw-runtime

La compilation de mingw-runtime ne devrait pas poser énormément de problèmes, il suffit en effet de retourner dans le dossier général de compilation, de décompresser l'archive dans le dossier source, de créer un dossier de compilation pour mingw-runtime, d'y entrer, d'invoquer la configuration, la compilation et l'installation avec les commandes

```
$ cd /work/build
$ tar -vxzf /c/downloads/mingw-runtime-3.13-20070825-1-src.tar.gz -C/work/src
$ mkdir mingw-runtime && cd mingw-runtime
$ ../../src/mingw-runtime-3.13-20070825-1/configure --prefix=/mingw \
--target=i686-pc-mingw32 --with-as=/mingw/bin/as.exe --with-ld=/mingw/bin/ld.exe
```

```
$ make
$ make install
```

sans qu'il n'y ait de nouveautés particulières;).

Sans oublier de faire le ménage pour la passe suivante avec un

```
$ make clean
```

V-B-3 - Compiler libiconv (facultatif)

Libiconv n'est pas à proprement parler indispensable pour compiler GCC.

Son intérêt réside dans le fait qu'il permet de convertir le codage des textes afin de modifier la table de caractères utilisée.

La présence de cette bibliothèque est vérifiée de manière systématique lors de la compilation de GCC.

C'est la raison pour laquelle je me dis que, tant qu'à faire, autant la compiler dès le départ ;).

La suite de commande pour y arriver est:

```
$ cd /work/build
$ tar -vzxvf /c/downloads/libiconv-1.11.tar.gz -C /work/src
$ mkdir libiconv && cd libiconv
$ ../../src/libiconv-1.11/configure --prefix=/mingw host=i686-pc-mingw32 \
--with-gnu-ld --with-gnu-as --with-ld=/mingw/bin/ld.exe \
--with-as=/mingw/bin/as.exe
$ make
$ make install
$ make clean
```

V-B-4 - GMPLib et MPFR

Il y a deux bibliothèques qui seront nécessaires si vous souhaitez activer le support de FORTRAN dans GCC et dont je n'ai pas encore parlé. Il s'agit de bibliothèques mathématiques particulières nommées respectivement gmplib et mpfr.

GmpLib est l'abréviation de **GNU Multiple Precision Arithmetic Library**, et est donc, pour ceux qui auraient des problèmes avec la langue de shakespeare, une bibliothèque GNU fournissant la précision arithmétique multiple. Elle est disponible directement sur le [site de l'équipe qui s'en occupe](#)

MPFR est l'abréviation de **Multiple-Precision Floating-point computations with correct Rounding**, c'est à dire une bibliothèque qui fournit la possibilité d'effectuer des calculs à virgule flottante avec un arrondi correct. Vous pouvez également vous en procurer les sources [directement sur le site qui s'en occupe](#)

Vous pouvez, bien sûr, décider de ne pas les installer si vous ne prévoyez pas d'activer le support de FORTRAN, mais, d'un autre côté, les occasions dans lesquelles vous serez content d'en disposer, même en C ou en C++, risquent d'être nombreuses... Mais bon, à vous de choisir ;).

V-B-4-a - Compiler GMPLib

Le script de configuration de gmp lib lui permet de déterminer de manière très correcte la meilleure manière de compiler la bibliothèque en fonction de votre système. De ce fait, il suffit de lui fournir le dossier dans lequel vous souhaitez la voir installer, en précisant cependant que nous voulons utiliser notre toute nouvelle collection d'outils.

Une fois de retour dans le dossier build , il vous suffira donc de décompresser l'archive, de créer un dossier de compilation pour gmp et d'y entrer, avant de demander la configuration, la compilation et l'installation à l'aide des commandes

```
$ cd /work/build
$ tar -vxjf /c/downloads/gmp-4.2.1.tar.bz2 -C /work/src
$ mkdir gmp && cd gmp
$ ../../src/gmp-4.2.1/configure --prefix=/mingw --with-as=/mingw/bin/as.exe \
--with-gnu-ld=/mingw/bin/ld.exe
$ make
$ make install
```

Sans oublier de faire le ménage pour la passe suivante

```
$ make clean
```

V-B-4-b - Compiler MPRF

La compilation de MPRF ne devrait pas non plus poser problème. La suite désormais classique d'instructions

```
$ cd /work/build
$ tar -vxjf /c/downloads/mpfr-2.2.1.tar.bz2 -C /work/src
$ mkdir mpfr && cd mpfr
$ ../../src/mpfr-2.2.1/configure --prefix=/mingw --target=i686-pc-mingw32 --with-gnu-ld \
--with-gnu-as --with-ld=/mingw/bin/ld.exe --with-as=/mingw/bin/as.exe
$ make
$ make install
```

sera suffisante, mais il ne faudra pas oublier de faire le ménage avec

```
$ make clean
```

V-C - Compiler GCC

Nous arrivons, enfin, au but réel de cet article: la compilation de GCC.

Bien qu'ayant essayé à peu près toutes les solutions proposées, je n'ai trouvé aucun moyen de compiler correctement libjava (la bibliothèque de support du langage Java). Ce qui suit va donc permettre la compilation de tout, sauf de cette bibliothèque.

Sur les FTP de GNU.org, il est possible de trouver les sources de la collection sous deux formes: une archive regroupant les sources de l'ensemble de la collection ou plusieurs archives ne contenant chaque fois que les source d'une des parties de la collection (GCC-core (base), GCC-g++ (support de C++), GCC-ada (support d'Ada), GCC-g77 (support de FORTRAN) et GCC-objc (support de ObjectiveC).

Si vous décidez de ne pas prendre l'ensemble des sources, vous devrez au minimum prendre l'archive GCC-core et les archives qui correspondent aux langages pour lesquels vous souhaitez activer le support.

Par facilité, je considérerai que vous avez téléchargé l'archive "commune", si vous avez téléchargé les archives séparées, sera qu'il faudra décompressé chaque archive une à une.

La décompression des sources ne devrait pas poser de problème, avec l'éternelle suite de commande

```
$ cd /work/build
$ tar -vxjf /c/downloads/gcc-4.2.1.tar.bz2 -C /work/src
$ mkdir gcc && cd gcc
```

V-C-1 - Avant la configuration

L'un des principaux problèmes que l'on rencontrera au cours de la compilation est que la création de liens symboliques échoue sous MSYS, lors de la compilation de libada.

Pour résoudre ce problème, il suffira de modifier une seule variable, que nous retrouverons dans plusieurs fichiers.

Ainsi, il faudra rechercher la variable LN_S, qui est définie avec une valeur de @LN_S@, dans le fichier c:\msys\1.0\work\src\gcc-4.2.1\gcc\libada\Makefile.in aux alentours de **la ligne 30**

et la modifier pour lui donner la valeur **cp** de manière à forcer la copie des fichiers plutôt que la création de liens symboliques.

Assez bizarrement les autres bibliothèques ne souffrent pas de ce problème.

V-C-2 - La commande de configuration

La commande de configuration prend la forme de

```
$ ../../src/gcc-4.2.1/configure --prefix=/mingw --target=i686-pc-mingw32 \
--program-prefix="" --with-as=/mingw/bin/as.exe \
--with-ld=/mingw/bin/ld.exe --with-gcc --with-gnu-ld --with-gnu-as \
--enable-threads --disable-nls \
--enable-languages=c,c++,ada,fortran,objc,obj-c++,treelang \
--disable-win32-registry --disable-shared --without-x \
--enable-interpreter
```

et est, en définitive, fort proche de ce que gcc -v nous a donné.

Seules les références à libjava et ses à côtés ont été supprimées pour les raisons exposées plus haut.

V-C-2-a - Comprendre la commande

La plupart des options fournies devraient permettre à quiconque connaissant un minimum l'anglais de les comprendre.

Cependant, pour ce qui concerne les langages dont le support est activé, cette liste est exhaustive de l'ensemble des langages supportés. En effet, il est possible de remplacer cette liste par --enable-languages=all, mais cette manière

de faire n'activera le support que pour C, C++, Java, FORTRAN, Ada et ObjC, en ne prenant pas en compte celui de Objective C++ et de treelang.

V-C-2-b - Au sujet de la compilation en elle-même

Lorsque l'on décide de lancer la compilation d'un compilateur, on ne sait jamais à la base quel compilateur, ou quelle version du compilateur sera utilisée pour effectuer la première compilation.

C'est la raison pour laquelle lorsque vous lancerez la compilation de GCC, il y aura en définitive trois compilations successives effectuées. Ce phénomène de compilations successives s'appelle le **bootstrap**

- La première pour s'assurer que les suivantes seront effectuées avec la bonne version de GCC
- La deuxième pour être tout à fait sûr que ce soit bien la bonne version de GCC qui a compilé GCC
- la troisième pour s'assurer que GCC sait se compiler lui-même

Il *paraît* en outre que le compilateur est de plus en plus optimisé au fil des compilations successives (mais je ne suis pas en mesure de confirmer ni d'infirmer une telle chose)

Cependant, il faut bien se rendre compte que de telles compilations en cascade prennent du temps. Et il est tout à fait possible de modifier le nombre de ces compilations.

En effet, il est possible de n'effectuer qu'une seule compilation, par exemple si vous avez déjà la version de compilateur fonctionnelle, mais que vous voulez obtenir un compilateur croisé en rajoutant

```
all-target stage1-bubble
```

à la commande de compilation (attention, **vous risquez aussi de devoir ajouter la commande `enable-stage1-languages=all` lors de la configuration**).

Vous pouvez aussi demander à n'effectuer que deux compilations, de manière à gagner un peu de temps, quitte à ce que le compilateur ne soit pas aussi bien optimisé (ce sur quoi je ne me prononcerai pas) en rajoutant

```
bootstrap2
```

à la ligne de compilation.

Enfin, vous pouvez, au contraire, demander à effectuer une quatrième compilation, quitte à perdre encore plus de temps, mais dans l'idée qu'il sera mieux optimisé au final en rajoutant

```
bootstrap4
```

à la ligne de compilation.

De plus, il est tout à fait possible d'utiliser les drapeaux classiques pour GCC dans la ligne de compilation:

```
-J(nombre de processeur(s) ou de coeur(s))+1
```

aura de grandes chances de diminuer le temps de compilation, et ce d'autant plus que vous aurez une architecture avec de nombreux coeurs ou processeurs.

```
-m32   ou  
-m64
```

sera utile au processeurs d'architecture pour déterminer si l'exécutable doit être codé sur 32 ou 64 bits.

```
march=<architecture>
```

permettra d'affiner l'architecture que vous utilisez

```
-Os  
-O3
```

permettra d'optimier le compilateur pour la taille, ou pour la vitesse (**attention, j'ai entendu, sans être en mesure d'affirmer quoi que ce soit, que -O3 était déconseillé pour le compilateur**).

V-C-2-c - Lancer la compilation

Il n'y a plus qu'à lancer la compilation avec la commande

```
$ make CFLAGS="-O2 -fomit-frame-pointer" \  
CXXFLAGS="-mthreads -fno-omit-frame-pointer -O2" LDFLAGS=-s
```

Si vous n'avez rien précisé comme information de bootstrap, et selon le matériel dont vous disposez, vous pouvez sans doute décider d'aller au cinema, car cela prendra du temps à se faire (4 à 5 heures dans mon cas).

Une fois que la compilation aura été menée à terme, il ne restera plus qu'à demander l'installation avec la commande

```
$ make install
```

VI - La deuxième passe

La deuxième passe a juste pour but de s'assurer que l'ensemble des outils et bibliothèques ait été compilé avec la version fraîchement installée de GCC.

Il s'agira donc de refaire la compilation de binutils, de w32-api, de mingw-runtime et de libiconv en utilisant la nouvelle version de GCC.


Le plus facile étant de quitter MSYS avec la commande

```
$ exit
```

puis de le relancer, sans modifier le PATH cette fois, d'entrer dans les dossiers correspondants se trouvant dans /work/build et de relancer les commandes **make** et **make install** respectives.

Vous pourrez, si vous le souhaitez, compiler bison et flex pour qu'ils l'aient été avec les versions de compilateurs et de binutils que vous venez de compiler. Cependant, la dernière version de bison connue pour être compilable sous windows est la version 2.1 et je n'ai personnellement pas réussi à compiler la moindre version de flex sous windows, entre autres du fait des dépendances rencontrées.

Une fois la deuxième passe finie, il vous sera possible, si vous le désirez, de terminer par l'ajout de bibliothèques ou d'outils supplémentaires.

 *Maintenant que vous êtes sûr de disposer du compilateur définitif, vous pouvez, si vous le souhaitez, affiner les options de configuration en commençant par vider chaque fois le dossier (**rm -rf ***) puis en relançant la commande de configuration.*

N'hésitez pas à invoquer `configure --help` afin de trouver les options qui vous intéressent ;).







Une fois la deuxième passe terminée, vous pouvez sans problème supprimer les dossiers c:\gcc et c:\msys\1.0\work ainsi que leurs sous dossiers.

Si vous estimez ne plus avoir besoin de MSYS, vous pouvez le désinstaller sans remords (certains dossiers non vides risquent de ne pas être supprimés, pensez à vérifier manuellement :)).

Il sera également intéressant d'envisager de rajouter c:\mingw\bin à la variable PATH de windows (tous les programmes -> système->onglet "Avancé"->Variable d'environnement)





VII - Liens et remerciements

VII-A - Trouver les archives

-  **Site qui s'occupe de la bibliothèque GMP**
-  **site qui s'occupe de la bibliothèque MPFR**
-  **l'un des miroirs Sourceforge de GnuWin32.**
-  **sur l'un des miroirs de Sourceforge relatifs à MinGW**
-  **Section téléchargements du site MinGW**
-  **liste des miroirs GCC**

Vous pourrez en outre, une fois que vous avez choisi votre miroir pour GCC, accéder à l'ensemble des sources du projet GNU en cliquant sur le lien "vers un rép. de plus haut niveau"

VII-B - Sources d'inspiration

-  **How to compile GCC 4.1 sur le wiki de MinGW**
-  **delorie's web site: GNU Compiler Collection (GCC) Internals**
-  **Installing GCC sur gcc.gnu.org**
-  **le projet info-zip**

VII-C - Liens connexes


- **Introduction aux makefile**
- **Mode d'emploi de GCC**
- **Test de l'EDI code::blocks**
- **Entrée de la FAQ concernant la prochaine norme**


VII-D - Remerciements

Je tiens à remercier **Alp**, **Frank.H**, **Millie** et **Aurelien.Regat-Barrel** pour leur relecture attentive et leurs commentaires éclairés lors de la rédaction de cet article.

Je veux aussi faire une dédicace spéciale à **Nono40** pour avoir fourni l'outil sans lequel cet article n'aurait pas eu une telle qualité de présentation.

1 : "lookahead LR": méthode d'analyse permettant de concevoir des automates à pile (voir

 **ce document** pour plus d'informations)

2 : "Generalized LR": méthode d'Analyse non déterministe des grammaires non contextuelles(voir  **cet article** pour plus d'informations)