# Introduction à Unix
# Mon premier projet sous Linux

**Résumé**

Le CodeBlock magique fait tout mais comment ? Voilà une question que vous allez pouvoir éluder durant ce cours. Vous allez mettre en place votre première chaîne de compilation pour vous éloigner un peu plus de la magie et comprendre les dessous d'une réalité : La Matrice n'existe pas ! A travers cette chaîne de compilation, vous apprendrez le format standard des projets Linux et Open Source mais également les nombreuses possibilités qui s'ouvrent à vous au delà du F9.

## 1 Ma première compilation (... à la main)

*Utilisez un éditeur de texte (Gedit par exemple) pour recopier le code source suivant :*

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  int i;

  printf("Hello World!\n");

  return 0;
}
```

### 1.1 Compilation simple

Votre code source est dans le fichier `helloworld.c`.
*Compilez-le avec la commande :*

```
gcc -o helloworld helloworld.c
```

### 1.2 Attention

En fait, ce code n'est pas très pro, il y a une variable inutile.
*Recompilez avec la commande :*

```
gcc -Wall -o helloworld helloworld.c
```

Une compilation ne devrait jamais produire de *warning*. Même s'il n'empêche pas la compilation, les *warnings* mettent en avant les ambigüités de votre code.

## 2 Mon premier *make*

La commande `make` permet de construire un résultat (généralement un programme) en ne reconstruisant ses dépendances que si elle ne sont plus à jour. Exemple, vous avez un gros projet de plusieurs centaines de millier de lignes. La compilation de tout le projet prends bien 3 bonnes minutes. Comme vous êtes un bon développeur, vous avez séparé le code ne plusieurs fichiers source. Comme vous le savez, avant d'être compilé en binaie, chaque fichier source (.c) est compilé dans un format intermédiaire (objet .o). Si vous ne modifiez dans votre projet qu'un seul .c, un seul .o doit être recompilé et non pas tout le projet. C'est ce que fait make.

*Désarchivez l'archive du TP :*

```
tar xvzf tp_c.tar.gz
```

### 2.1 Le make simple

*Constuisez le projet en appelant la commande* `make`. *Le fichier* `makefile` *est utilisé implicitement. Étudiez ce fichier pour comprendre ce qu'il fait.*

```
make
```

### 2.2 Le make générique

*Nettoyez le projet :*

```
make mrproper
```

La commande `make` dispose d'une syntaxe puissante qui vous permet d'écrire des règles de construction réutilisables pour différents projet

*Reconstruisez le projet, mais cette fois en spécifiant un fichier particulier :*

```
make -f general_makefile
```

*Étudiez ce fichier pour comprendre ce qu'il fait. Vous remarquerez qu'il suffit de changer la valeur de la variable EXEC (première ligne) pour compiler un autre projet.*

## 3 Les arguments d'un programme

Le code du programme *musicinfo* vous est donné. Il permet de générer un rapport sur l'état de votre bibliothèque musicale du TP précédent.

*Rajouter du code dans le fichier* `main.c` *pour gérer les arguments du programme. Vous devez récupérer les différents chemins des dossiers à parcourir. Vous devez également gérer une option* `--count` *permettant de compter le nombre de musique de votre bibliothèque.*