DUT-S1

*M1 101 - Utilisation des Systemes Informatiques*

# Projet SHELL - TP noté sur 3 scéances

## 1 Consignes

Le travail est à faire en **binôme** et à rendre **avant** le **18/01/2016** sur la plateforme d'enseignement sous la forme d'un fichier `.tar.gz` contenant deux fichiers **uniquement** :

1. Le script Shell nommé `gestionnaire.sh`
   - commenté (cartouche fonction + code facile à comprendre)
   - aéré
   - indenté
   - présence du nom des auteurs dans le fichier
2. Un rapport **PDF** qui précise :
   1. les auteurs du travail
   2. ce que fait le script
   3. la syntaxe d'appel du script
   4. des traces d'exécution
   5. les difficultés rencontrées lors du projet
   6. le travail qui a été réalisé
   7. les amélioration éventuellement faites

- Toute remise **hors délais (17/01/2015 23h50)** ne sera pas corrigée (note de 0)
- Vérifiez dès maintenant que vous avez accès à la plateforme
- Tout rendu fait en monôme ou trinôme ne sera pas corrigé (note de 0)
- Le **strict** respect des consignes sera pris en compte dans la notation (vous codez ce que l'on demande, pas ce que vous voulez faire)

## 2 Présentation du sujet : le gestionnaire de commande

Il s'agit d'un script Shell qui permet le développement de programmes C++ à partir de commandes d'un menu. Ce script prend en entrée optionnelle le nom d'un fichier C++ (sans l'extension .cpp). Si ce fichier n'est pas fourni en argument, le script devra demander le nom à l'utilisateur au début du script (toujours sans extension .cpp). Pour cela, le gestionnaire devra au préalable afficher tous les programmes C++ existants dans le dossier[1].

*Ceci fait que ce script ne fonctionne que pour le cas de la compilation de projet ne contenant qu'un seul fichier.*

1. 1 programme=1 fichier

Il doit être aussi possible d'utiliser le script shell avec l'option `-h`. Dans ce cas, le script devra afficher un message indiquant comment utiliser le script. Par exemple : la commande `./gestionnaire.sh - h` affiche :

```
Usage: ./gestionnaire.sh [fichier]
fichier est un fichier C++ existant ou pas,
donné sans extension.
```

Si le fichier n'existe pas, il faut créer le fichier .cpp avec le template suivant :

```
#include <iostream>

using namespace std;

int main(int argc, char **argv) {
  cout << "Hello Wordl !" << endl;
  return 0;
}
```

Les commandes du menu sont désignées par leur nom complet ainsi que chaque raccourci valide et sont insensibles à la casse :

- **Voir** permet d'afficher le contenu du fichier C++ à l'écran page par page (la variable d'environnement PAGER devrait contenir l'éditeur par défaut, sinon ce rabattre sur un éditeur installé sur la machine tels que less, more, most, ...).
- **Éditer** permet de lancer l'édition du fichier C++ (la variable d'environnement EDITOR devrait contenir l'éditeur par défaut, sinon ce rabattre sur un éditeur installé sur la machine tels que emacs, vim, nano, jed, ...).
- **Générer** permet de compiler le fichier C++ en utilisant g++. Bien séparer la compilation (génération d'un fichier objet) de l'édition des liens (génération d'un exécutable). En cas d'erreur lors de la compilation, il demande aussi la possibilité d'afficher les erreurs (les erreurs ne devront être affichées sur l'écran que si l'utilisateur le demande) à l'aide du pager.
- **Lancer** lance le programme. Au préalable, le script devra vérifier que le fichier exécutable existe et qu'il a le droit d'exécution positionné. Si le droit est mal positionné, il l'active en conséquence.
- **Débugguer** lance le programme (compilé en mode debug) depuis gdb.

— **Imprimer** imprimer le fichier C++ (l'opération est effectuée virtuellement avec `a2ps` et `ps2pdf` afin de générer un fichier PDF).
— **Shell** lance un "sur shell" avec un prompt différent de celui du Shell. Dans ce cas, le script affiche un message qui averti que l'utilisateur devra taper "exit" pour revenir dans ce script.
— **Quitter** arrête définitivement le script.

# 3 Travail facultatif

Le travail facultatif ne peut être fait qu'en complément du travail initial (sinon, il ne sera pas corrigé). Dans tous les cas, vous devez avoir un dossier par version de projet.

## 3.1 Utilisation de Makefile

Nous n'avons pas vu les Makefiles en cours, mais c'est l'opportunité de l'apprendre maintenant. Utilisez un Makefile pour compiler votre projet.

## 3.2 Gestion de projets à plusieurs fichiers

Permettre à ce script de fonctionner sur un dossier contenant plusieurs fichiers .cpp et .h. Cette fois, le script prend en entrée le nom du dossier. Les commandes voir et éditer devront au préalable demander le nom du fichier concerné. Le fichier Makefile devra être construit avec la commande `gcc -MM` pour les dépendances entre fichiers (le mieux est d'inclure un fichier Makefile.dependencies, construit avec `gcc -MM` dans le fichier Makefile).