Travaux Dirigés Programmation C avancée

Informatique 1ère année.

—*Julien Allali* - `allali@enseirb-matmeca.fr` —

Lors du développements de projets, il arrive souvent que nous faisions appel à des bibliothèque extérieures (c'est le principe de "ne pas ré-inventer la roue").

1 Une nouvelle bibliothèque

Créer un répertoire de travail dans lequel nous allons ajouter une la bibliothèque : `statelib`. Celle-ci est disponible sur la plateforme github.

▸**Exercice 1.** *Découverte.*
Quel est le but de cet bibliothèque ? Qui en est l'auteur ? quelle est la licence ?

▸**Exercice 2.** *Téléchargement et installation.*
Quel est le système de gestion de source utilisé pour ce projet ?

Donner le commande permettant de rapatrier les sources dans le répertoire.

Quel est le système de compilation utilisé ? Donner les commandes permettant de compiler cette bibliothèque.

Quel est le système utilisé pour la documentation ?

▸**Exercice 3.** *Test de la bibilothèque et installation.*
Compiler la bibilothèque. Exécuter les programmes de demonstration et regarder le code source à l'origine de ces programmes.

Pour pouvoir utiliser la bibliothèque, il faut installer celle-ci. La commande `make install` *permet de procéder à l'installation des fichiers nécessaires. Tester cette commande.*

- *quelle instruction dans les fichiers CMakeLists.txt permet d'indiquer quels fichiers installer et où les installer ?*
- *comment indiquer le répertoire où l'on souhaite installer ces fichiers ?*

Créer un répertoire `usr` *à la racine de votre compte et effectuer l'installation de la bibliothèque dans ce répertoire.*

2 Diff et patch

Nous allons maintenant modifier cette bibliothèque et communiquer ces modifications à l'aide des outils `diff` et `patch`.

▸**Exercice 4.** *Modifications.*
Faire une copie du répertoire `statelib` *en* `statlib_new`. *Les modifications que vous ferez ci-dessous seront effectuées dans* `statlib_new`.

La documentation de la fonction `state_start` *est incomplète, en effet rien n'est indiqué quand à la valeur de retour en cas d'erreur (transition non déclarée dans la machine). Modifier la doc en conséquence.*

Dans le répertoire `demos`, *ajouter un nouvel exemple d'utilisation de la bibliothèque de votre choix. Penser à modifier le* `CMakeLists.txt` *afin d'automatiser la compilation de votre exemple. A titre de suggestion, vous pouvez écrire :*

- *un jeu de recherche d'un nombre*
- *un jeu de tic-tac-toe*

▸**Exercice 5.** *Diff.*
À l'aide de la commande diff, nous allons analyser les modifications que vous venez d'apporter à la bibliothèque statelib. Pour cela, utiliser la commande `diff -r original new`.

▸**Exercice 6.** *Generation d'un patch.*
Afin de partager vos modifications avec un autre groupe, vous allez contruire un patch. Pour cela utiliser la commande `diff -rupN statelib statelib_new > patch`.

Examiner le contenu du fichier patch ainsi créé.

Echanger vos fichiers patch avec un autre groupe.

▸**Exercice 7.** *Application d'un patch.*
Nous allons tenter d'appliquer le patch reçu par l'autre groupe dans vos sources. Pour cela faire une copie de `statelib` *en* `statelib_patched`. *Dans ce dernier répertoire, appliquer le patch avec la commande* `patch < fichier_de_patch_recu`. *Utiliser la commande diff pour vérifier les modifications apportées par le patch.*

Faire une copie de `statelib_new` *en* `statelib_new_patched` *et essayer d'appliquer le patch dans ce dernier répertoire. Analyser les problèmes que vous rencontrez et proposer des solutions.*

▸**Exercice 8.** *Nouveau patch.*
Récupérer le fichier `patch.print` *disponible en ligne sur la page du cours. Que contient ce patch ? Reprendre l'exercice précédent avec ce fichier.*

3 Git

▸**Exercice 9.** *Application des modifications*
Dans le répertoire `statelib`, *appliquer le patch* `patch.print` *fournit. Appliquer également vos modifications (votre programme de démo).*

A partir de maintenant, nous travaillons dans le répertoire `statelib`

▸**Exercice 10.** *Add et commit*
La commande `git status` *permet d'afficher les modifications en cours. Utiliser cette commande pour visualiser les modifications. Quelle(s) commande(s) permet(tent) d'enregister ces modifications localement ?*

▸**Exercice 11.** *Transmission des modifications entre utilisateurs.*
Vous ne pouvez pas transmettre vos modifications directement sur github. Pour transmettre vos modifications à un autre groupe, vous pouvez relocaliser le dépôt d'origine et récupérer les modifications depuis ce dépot. En pratique, nous avons deux groupes G1 et G2.

Le groupe G1 entre la commande `pwd` *depuis le répertoire* `statelib` *et transmet le chemin complet au groupe G2.*

Le groupe G2 va entrer les commandes suivantes :

```
git remote add  G1 file://<chemin vers de dépot de G1>
git pull G1 master
```

Le pendant de la commande `pull` *est la commande* `push` *qui permet d'envoyer (et non recevoir) des modifications. Dans le cas présent, vous ne pourrez pas utiliser* `push` *car pour cela il faudrait avoir les droits nécessaires pour modifier les fichiers de G1. Une solution serait d'utiliser des clés ssh (mais cela est assez compliqué et dépasse de loin ce td).*

La possibilité de transmettre ainsi des modifications entre utilisateurs sans passé par un dépôt central est ce qui caractérise les gestionnaires de sources décentralisés. C'est également une fonction assez difficile à maitriser dans un premier temps.

▸**Exercice 12.** *qtcreator*
Depuis le répertoire `statelib`, *lancer la commande* `qtcreator CMakeLists.txt`. *Ouvrer le fichier* `src/state.c` *et ajouter une ligne de commentaire dans celui-ci. Dans le menu "Outils" puis "Git", utiliser la fonction "dépôt local / diff".*

Comme tout les IDE, QtCreator intégre la gestion des dépôts git/svn/cvs/bazaar/mercurial...

4 SVN

Dans cette partie, nous allons gérer les sources de la table de hachage dans un dépôt local svn.

▸**Exercice 13.** *Création d'un dépôt local :*
À la racine de votre compte, créer un répertoire .depots. *Dans ce répertoire créer un nouveau dépôt svn nommé* pg106 *avec la commande* `svnadmin --compatible-version 1.5 create pg106` *(l'option ajoutée est nécessaire du fait de la configuration locale).*

Ceci créer un dépôt svn. **Attention : on ne travaille jamais directement dans le dépôt mais dans un autre répertoire synchronisé avec le dépôt !**

Dans votre répertoire de travail, synchroniser vous avec le dépôt en utilisant la commande `svn checkout file://$HOME/.depots/pg106`. *Vous devriez obtenir la création d'un répertoire pg106 vide (à l'exception du dossier* `.svn`*).*

▸**Exercice 14.** *Structure de travail.*
Dans le dépôt créer trois répertoires : trunk, branches et tags. Transmettez ces modifications au dépôt.

▸**Exercice 15.** *Ajout des sources.*
Ajouter dans le trunk l'ensemble des sources de la table de hachage (`CMakeLists.txt`, `src/hash.c`, `src/hash.h` *etc...).*

Attention à ne pas ajouter des répertoires cachés type .git ou .svn.

Avant de transmettre les modifications au dépôt (commit), vérifier celles-ci avec la commande svn status.