TD0: Environnement de programmation

Septembre 2015

Durant ce semestre vous travaillerez sous l'environnement Linux de la machine virtuelle que ce soit pour les TDs ou pour le projet. Dans cet environnement, vous utiliserez autant que possible le terminal afin de vous déplacer dans l'arborescence des fichiers, d'éditer votre programme, de compiler votre programme, de débugger votre programme, et de mettre vos travaux sous contrôle de version sur un répertoire distant. Pour cela, vous utiliserez les programmes suivants :

— `geany` pour l'édition,
— `gcc` pour la compilation,
— `gdb` pour le débuggage,
— `git` pour le contrôle de version.

Dans un premier temps, vous allez apprendre les fonctionnalités de base du terminal. Les commandes que vous utiliserez sont

```
man, ls, cd, pwd, mkdir, mv, rm, wc, cat
```

leur description peut être obtenue avec la commande `man commande`. Deux fonctionnalités du terminal qui s'avèrent très utiles à l'usage, et que vous vous efforcerez d'utilisez dès le début, sont la complétion automatique avec la touche tabulation, et le parcours de l'historique des commandes avec les flêches haut et bas.

Exercice 1. Terminal et compilation

a. Ouvrez un terminal (essayer la combinaison de touches `CTRL+ALT+T`). Dans quel répertoire vous trouvez-vous ? Que se passe-t'il si vous appuyer sur la touche `w` puis deux fois sur `TAB` dans le terminal ? Puis `wh` et deux fois sur `TAB` ? Puis `whe` et une fois sur `TAB` ? Que fait la commande que vous venez de taper ? Essayer `whereis ls`.

b. Déplacez vous vers le répertoire `Bureau`. Revenez dans le répertoire initial, puis encore dans le répertoire `Bureau` sans taper complètement au clavier `Bureau`.

c. Affichez ce que contient le répertoire `Bureau`. Créez un répertoire `essai`, et vérifiez que celui-ci a bien été créé.

d. Déplacez vous dans le répertoire, puis éditez un fichier vide `fic1` avec `geany` en tapant la commande `geany fic1 &`, puis fermez `geany`. Que contient le répertoire maintenant ?

e. Copiez le fichier vers un fichier `fic2`. Changez le nom de `fic2` en `fic3`. Supprimez le fichier initial `fic1`.

f. Supprimez tous les fichiers et ajoutez le programme `debug.c` qui est sur `e-campus2`. Affichez le programme dans le terminal. Combien de lignes et de caractères contient-il ?

g. Nous allons maintenant compiler le programme. Taper la commande `gcc debug.c`. Vérifiez qu'un fichier `a.out` a été créé. Executez le programme en tapant la commande `./a.out`. Manifestement ce programme contient un bug... mais vous ne le corrigerez pas pour l'instant.

h. Pour choisir un nom à votre exécutable, tapez la commande `gcc -o nom debug.c`. Vérifiez qu'un exécutable ayant le nom que vous avez choisi a bien été créé, et exécutez le.

i. Compilez en ajoutant les warnings de compilation avec la commande `gcc -o nom -Wall debug.c`. Que constatez-vous ?

Notez bien que pour compiler un programme qui utilise la librairie mathématique, il faut ajouter l'option `-lm`. Au final, cela donne la commande suivante :

```
gcc -o nom -Wall debug.c -lm
```

Exercice 2. Contrôle de version : git

Comme expliqué en cours, vous allez utiliser un gestionnaire de version qui s'appelle `git`, et un dépôt distant qui s'appelle `github`. Il y a (au moins) 2 intérêts à cela par rapport à une sauvegarde classique : d'une part vous conservez l'historique de toutes les versions et pouvez revenir en arrière en cas d'erreur ou de suppression inopinée de votre travail, et d'autre part, vous faites des sauvegardes sur un dépôt distant qui vous prémunit des accidents que pourrait toucher votre machine personnelle. L'objectif de cet exercice est de vous familiariser avec l'utilisation de cet outil que vous utiliserez tout au long du semestre (et au-delà je l'espère !).

a. Ouvrez un navigateur et allez sur la page web : `https://github.com/`

b. Créez un compte en mémorisant bien votre nom d'utilisateur et votre mot de passe.

c. Une fois que vous êtes connecté, créez un dossier qui s'appelle `IN301` en cliquant sur la croix en haut à droite de l'écran (create new repository). Votre dépôt distant sous gestionnaire de version git est créé, vous allez maintenant le récupérer sur votre machine.

d. Dans votre terminal, déplacez vous à l'endroit où vous voulez ajouter le dossier `IN301`. Créez une copie locale en tapant la commande suivante dans le terminal :

```
git clone https://github.com/moi/IN301
```

et en remplaçant `moi` par votre nom d'utilisateur. Un répertoire `IN301` a normalement été créé, ce que vous pouvez vérifier en tapant la commande `ls`.

e. Déplacez vous dans le répertoire `IN301`. Créez le dossier `td0` (commande `mkdir`) [1]. Alle dans le dossier `td0` et créez le fichier vide `essai` avec la commande `touch essai`. Nous allons maintenant ajouter ce fichier au répertoire distant.

f. Pour mettre un nouvel élément en contrôle de version, il faut utiliser la commande `git add fichier`, donc, ici, `git add essai`. Pour valider cet ajout, utiliser la commande `git commit essai`, ou la commande `git commit -a` qui validera toutes les modifications faites sur des fichiers qui sont sous contrôle de version. Une fenêtre s'ouvre pour que vous renseigniez une description de la modification. Ecrivez, par exemple, "ajout fichier td0/essai", cliquez ensuite sur `CTRL+x`, puis sur `o` (pour OUI) et `ENTREE`. Pour propager cela au répertoire distant, il reste à taper la commande `git push`. Vérifiez ensuite sur votre compte `github` que le dossier a bien été ajouté.

g. Ajoutez le fichier `debug.c` que vous avez manipulé dans l'exercice précédent dans le répertoire `td0`. En suivant les mêmes instructions que pour `td0` faites en sorte de mettre ce fichier sous gestion de version et de l'ajouter dans `github`.

h. Une fois que cela a bien été réalisé, supprimez le répertoire `IN301` de votre machine (commande `rm -rf IN301`) et vérifiez que cela a bien fonctionné. Faites maintenant un clône de votre dépôt distant `github`, et vérifiez que vous avez bien récupéré le dossier `td0` et qu'il contient le fichier `debug.c`.

i. Ajoutez du texte dans le fichier `essai` et sauvegardez. Testez alors la commande `git status`. Commitez le changement. Que renvoie la commande `git status` ? Poussez votre changement sur `github` et testez de nouveau `git status`.

Dorénavant, pour ceux qui travaillent avec une machine en prêt, vous pourrez récupérer vos travaux en début de séance en clonant votre répertoire distant.

A tout moment, vous pouvez "commiter" vos changements (ne pas oublier de faire `git add` pour les dossiers et fichiers nouvellement créés), et les propager avec la commande `git push`. Localement vous pouvez vérifier si vos fichiers sont bien enregistré avec la commande `git status`.

Enfin, notez qu'il ne faut commiter que les fichiers textes (typiquement les programmes terminant par l'extension .c) ainsi que les répertoires qui les contiennent, et non les exécutables.

1. Dorénavant, pour la feuille de td i, vous créerez un dossier `tdi` à cet emplacement

Exercice 3. Compilation et débug

Récupérez le fichier `debug.c` de l'exercice précédent. Nous allons utiliser le logiciel `gdb` afin de débugger ce programme.

a. Sans éditer le programme `debug.c`, compilez et exécutez le (voir exercice 1).

Les commandes de base du débugger `gdb` sont

```
run, quit, break, bt, print, step, next
```

b. Pour exécuter le programme dans le débugger, lancer la commande `gdb ./progDebug` dans le terminal, puis la commande `run`. Que se passe-t'il ?

c. Essayez d'ajoutez un point d'arrêt à la ligne 11 (commande `break 11`). Cela ne doit pas être possible car l'exécutable ne sait pas faire référence au fichier source. Il faut pour cela ajouter l'option `-g` à la compilation. Pour cela quittez `gdb` (commande `quit`) et recompilez en ajoutant l'option.

d. Retournez dans `gdb` et ajoutez un point d'arrêt à la ligne 11 puis lancez l'exécution du programme. Afficher la pile des appels (commande `bt`), et la valeur des variables dans la fonction courante. Continuez l'exécution du programme pas à pas en affichant les valeurs des variables régulèrement.

e. Corrigez la fonction `factorielle`, puis faites de même pour les fonctions `somme` et `maximum` en vous aidant si nécessaire de `gdb`.