Université Paris-Diderot — Licence d'Informatique 2012-2013

Examen de *Environnements et Outils de Développement*

Note : Vous avez 1h30 pour la première partie, 1h30 pour la deuxième. Vous avez droit à tous documents. La première partie est à rédiger sur papier, la deuxième à l'ordinateur. Pour rendre la deuxième partie vous devez **envoyer un mail à zack@pps.univ-paris-diderot.fr** ayant en pièce jointe une **archive tar** du contenu requis, et comme **objet : examen ed6 2013-1** . Vous pouvez consulter le Web pour la deuxième partie, mais toutes formes de communication avec les autres sont interdites.

Première partie

Exercice 1 (Le processus de compilation) *1. Décrivez le processus de compilation typique du langage C. Détaillez les différentes phases de "compilation" (au sens large), et expliquez les relations entre eux.*

2. *Expliquez l'erreur de liaison typique* undefined symbol.
3. *Expliquez la différence entre liaison statique et liaison dynamique.*

Exercice 2 (*Preprocessor*) *Le fichier* `hello.c` *contient le code suivant :*

```
#include <stdio.h>
#define INCR(x) ((x)++)
#define MULT(x, y) x * y
#ifdef DEBUG
#define INFO(s) printf("INFO: %s\n", s);
#else
#define INFO(s)
#endif
void hello(void) {
    INFO("entering hello")
    printf("Hello, World!\n");
    INFO("leaving hello")
}
int main(void) {
    int x = 41;
    printf("Stand back. I'm going to try math %d, %d!\n", INCR(x), MULT(1+2, 3));
    hello();
}
```

1. *Montrez le code C obtenu à la sortie du* preprocessor *de* `hello.c`. *(Vous pouvez ignorer les annotations de numéro de ligne et raccourcir l'inclusion de gros fichiers.)*
2. *Vous voulez maintenant activer l'affichage des messages de* debugging. *Comment pouvez vous le faire, à l'aide du* preprocessor *?*
3. *Les numéros affichés par le programme sont 41 et 7, mais le résultat que nous souhaitions (pour respecter la sémantique habituelle de l'incrémentation et de la multiplication) était 42 et 9. Expliquez comment corriger les deux* bugs.

Exercice 3 (*Make*) *1. Dans le cadre de Make, expliquez les notions de : règle, cible, prérequis, et commande.*

2. Considérez le `Makefile` *suivant :*

```
foo: main.o high.o low.o utils.o
        gcc -o foo main.o high.o low.o utils.o
main.o: main.c high.h utils.h
        gcc -c -o main.o main.c high.h utils.h
high.o: high.c low.h utils.h
        gcc -c -o high.o high.c low.h utils.h
low.o: low.c utils.h
        gcc -c -o low.o low.c utils.h
utils.o: utils.c utils.h
        gcc -c -o utils.o utils.c utils.h
clean:
        -rm -f foo main.o high.o low.o utils.o
```

donnez le diagramme de dépendances correspondant (i.e. un graphe où chaque noeud correspond à une cible et a comme fils ses prérequis)

3. Si aucun des fichiers `*.o` *n'est présent sur disque (et si tous les fichiers* `*.c` *et* `*.h` *existent), que se passe-t-il lors de l'exécution de* `make foo` *? Donnez la liste des commandes que* `make` *exécutera, dans un ordre valide.*

4. Si, après une première compilation complète, vous modifiez `utils.h` *et ensuite exécutez à nouveau* `make`*, que se passera-t-il ? Donnez la liste des commandes que* `make` *exécutera.*

Exercice 4 (*Gestion de versions*) *1. d'après vous, quel est l'intérêt des outils* `diff` *et* `patch` *? Quels sont les avantages de diff/patch pour échanger des améliorations d'un logiciel entre collègues, par rapport à envoyer des nouvelles copies complètes du même logiciel ? Y a-t-il des inconvénients ?*

2. Expliquer les différences principales entre un système de gestion de version centralisée et un système distribué.

3. Étant donné le fichier `main.c` *suivant :*

```
#include <stdlib.h>
int main(void) {
        printf("Hello, World!\n");
        exit(0);
}
```

montrez trois patch, *dans un format textuel similaire à celui de* `diff` *qui, respectivement :*

*(a) ajoute l'inclusion de l'*header `stdio.h` *avant* `stdlib.h`

(b) traduit le message "Hello, World" en Français

(c) traduit le message "Hello, World" en Espagnol ("Hola, Mundo !")

Lesquels de ces patch *peuvent être appliqués ensemble à* `main.c` *sans produire de conflits ?*

Deuxième partie

Exercice 5 (Édition de texte) *Téléchargez le fichier : `http://upsilon.cc/~zack/stuff/le_grand_meaulnes.txt`. Ce fichier contient les sept premiers chapitres du roman Le Grand Meaulnes d'Alain-Fournier. Néanmoins, il y a eu quelques petites erreurs qui se sont glissés dans ce fichier. Votre but est de les corriger. Des indications vous sont données entre `/* */`, vous soumettrez une version corrigée de ce texte en suivant les indications et en veillant à toutes les supprimer.*

Exercice 6 (Make) *Récupérez l'archive `tar` disponible à `http://upsilon.cc/~zack/stuff/ed6-exam-2013-1.tar.gz`.*

Écrivez un Makefile pour le projet C contenu dans l'archive. L'exécution de `make` doit produire un exécutable qui marche et lie tous les fichiers `*.c` ensemble. À noter : *pour la phase de liaison vous devrez lier avec les bibliothèques `libncurses`, `libnsl`, et `libtermcap`.*

Comme d'habitude pour un Makefile correct, l'exécution de `make` après des changements doit minimiser le nombre d'objets recompilés. Vous devez expliciter toutes les règles (i.e. l'utilisation de règles implicites prédéfinies est interdite) et minimiser la répétition d'information.

Exercice 7 (Git) *Produisez un* repository *Git dont l'historique ressemble le plus possible à celui montré en Figure 1. Le contenu du* repository *en terme des fichiers n'est pas importante (p.ex. vous pouvez utiliser un repository qui contient un seul fichier `toto.c`), mais les* commit, *branches, et* merge *doivent ressembler à ceux sur la figure 1.*

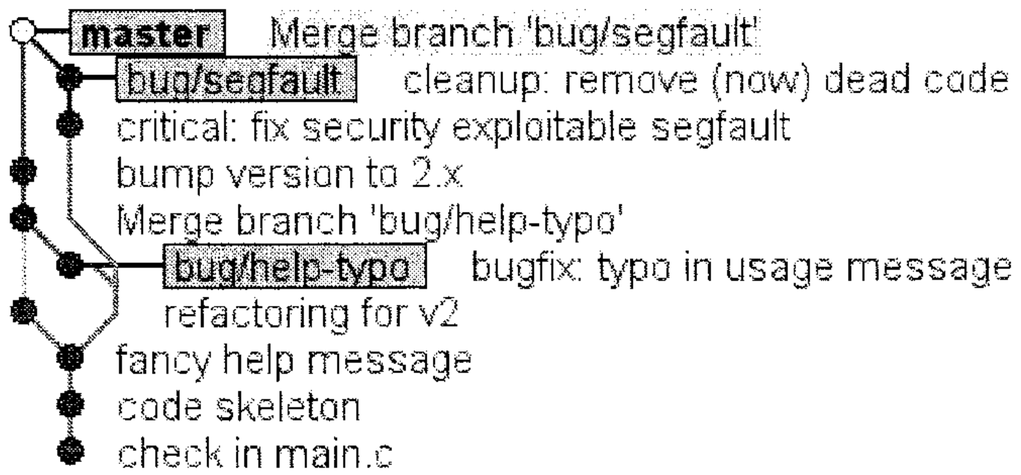

FIGURE 1 – historique Git, comme affiché par `gitk`