Question Bash

Jérôme De Boeck, Guillerme Duvillié, Nassim Versbraegen

UlB

1 Question bash

1.1 Énoncé

Dans le but d'optimiser votre environnement de développement et permettre l'utilisation de flags (tels que `TODO`, `XXX`, …) dans votre code, vous décidez de développer un outil permettant de répertorier les différents flags contenus dans les fichiers sources d'un projet donné.

Le principe est donc de parcourir récursivement les fichiers sources du projet (ces derniers étant reconnus sur base de leur extension) et de lister les occurences de certains flags dans ces derniers.

Après mûre réflexion sur les spécifications de votre outil, vous vous arrêtez sur les choix suivants.

1.1.1 Paramètres du script

- le premier paramètre définit le répertoire correspondant au projet,
- le second paramètre définit l'extension selon laquelle les fichiers sources seront reconnus (sans le `.`),
- tous les paramètres suivants définissent les différents motifs à rechercher dans les fichiers sources.

1.1.2 Sorties

- le script doit générer un fichier résultat par flag passé en paramètre,
- le nom de chaque fichier doit respecter la convention : `timestamp_nom_du_flag` (le timestamp doit être celui du moment de l'exécution du programme et doit être le même pour tous les fichiers résultats),
- chaque fichier contient une ligne par occurence du flag trouvé et la ligne est de la forme :
  `/chemin/absolu/vers/le/fichier -> ligne du fichier contenant l'occurence`.
  La ligne du fichier contenant l'occurence correspond au contenu de la ligne APRÈS l'apparion du flag. Ainsi, pour la ligne suivante trouvé dans le fichier source
  `// 2020-07-16 TODO Une tâche`
  le fichier résultat contiendra :
  `/chemin/absolu/vers/le/fichier -> Une tâche`.

1.1.3 Conventions

- l'ensemble de fichiers résultats doit être placés dans le répertoire suivant :
  `chemin/du/projet/flaglist_output`,
- s'il existe un conflit lors de la création d'un fichier résultat, le fichier résultat préexistant doit être déplacé dans le répertoire : `chemin/du/projet/flaglist_output/archive`,

- toutes les erreurs et les conflits rencontrés doivent être ajoutés à la fin du fichier : `chemin/du/projet/flaglist_output/flaglist.log`, ce dernier devant être créé s'il n'existe pas.

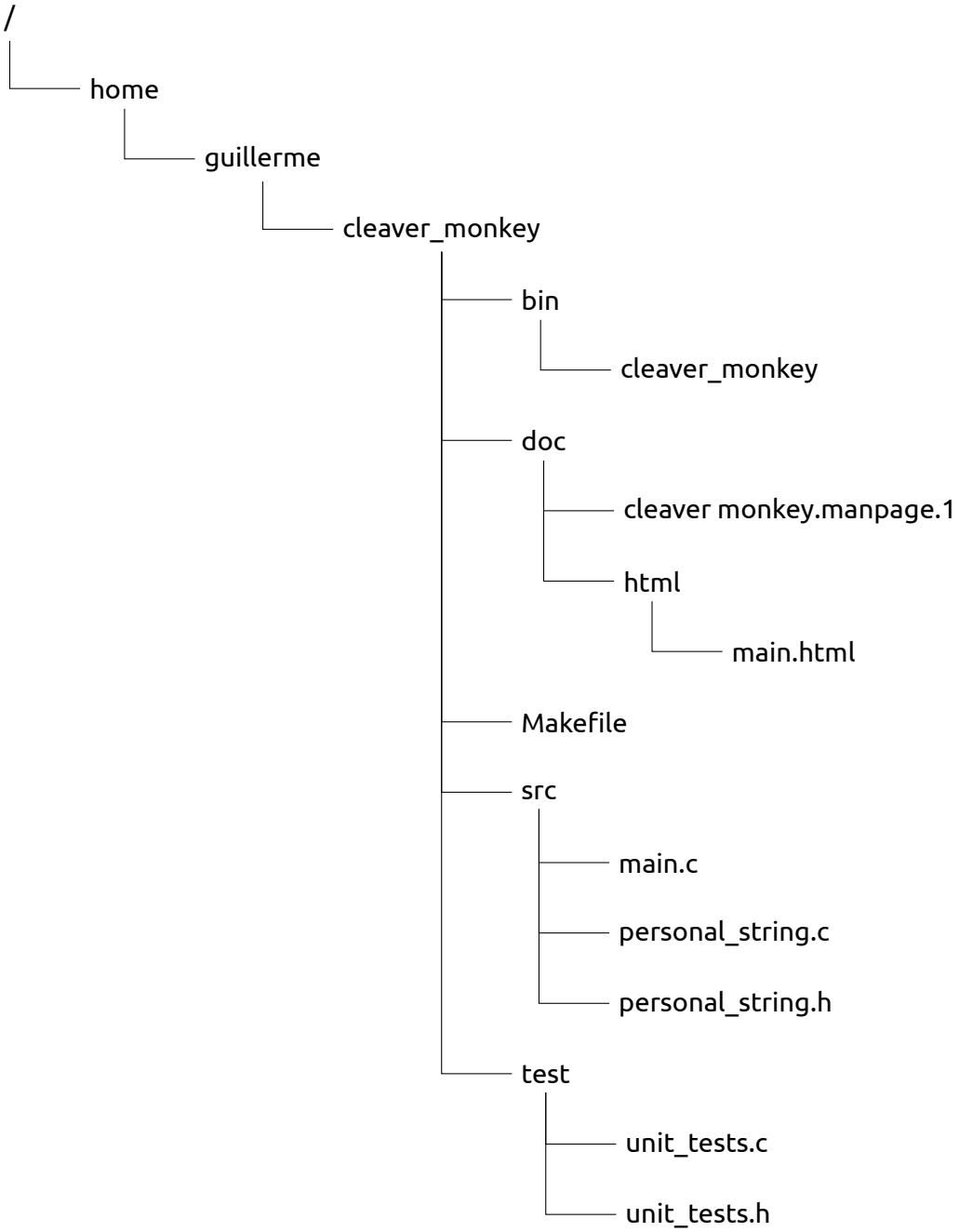
1.2 Ressources

La commande `date` permet d'obtenir le timestamp du moment présent grâce à l'appel suivant : `date +%s`.

Une arborescence de test ainsi que les fichiers résultats correspondants vous sont fournis dans l'archive jointe à l'examen.

1.3 Exemple

Supposons l'arborescence suivante :



Supposons la présence dans le code des lignes suivantes :

Listing 1 – main.c

```
...
// XXX Faire attention aux buffer overflows
...
// TODO Utiliser malloc et realloc
...
```

Listing 2 – personal_string.c

```
...
// TODO Optimiser la copie
...
```

Listing 3 – unit_tests.c

```
...
// XXX Attention tout le code DOIT être couvert par des tests unitaires
...
```

L'appel suivant :

`flaglist /home/guillerme/cleaver_monkey c XXX TODO`

crée le répertoire `/home/guillerme/cleaver_monkey/flaglist_output` contenant deux fichiers :

Listing 4 – 1594970142_TODO

```
/home/guillerme/cleaver_monkey/src/main.c -> Utiliser malloc et realloc
/home/guillerme/cleaver_monkey/src/personal_string.c -> Optimiser la copie
```

Listing 5 – 1594970142_XXX

```
/home/guillerme/cleaver_monkey/src/main.c -> Faire attention aux buffer overflows
/home/guillerme/cleaver_monkey/test/units_tests.c -> Attention tout le code DOIT
                                    être couvert par des tests unitaires
```

1.4 Consignes

Le script doit être écrit en `bash` et devra s'appeler `flaglist` (sans extension).

Seul le matériel vu lors du cours (TP et cours magistraux) peut être utilisé à ceci prêt que le recours à `find` et aux options récursives de n'importe quel programme vu en cours (`ls`, `grep`, …) est interdit.

Une attention particulière sera portée à la structuration du code en fonctions, à la qualité de ce dernier (indentation, respect des conventions, commentaires, …), aux tests d'erreur et au plagiat.