Rapport de projet

Administration des systèmes

Nicolas Soares & Nathan Sancho De La Rosa LO14 – Florent Retraint

Introduction

Introduction :

Le projet de L014 a pour objectif de créer un synchroniseur de systèmes de fichiers sous forme de script bash. A la fin de la synchronisation, les arbres A et B doivent être identiques ou dans un état très proche.

Notre projet compte trois fichiers différents :

- Le script (*synchroniseur*), il contient les différentes fonctions qui permettent de réaliser la synchronisation entre deux systèmes de fichiers.

- Le fichier de journal (*.synchro*), il contient les chemins A et B ainsi que l'historique des synchronisations réussies.
  Une entrée dans le journal correspond à :
  Chemin_A , chemin_B , type , permissions , taille_en_octets , date et heure dernière modification

- Le journal de conflits (*.journal_conflit*), il contient les conflits qui ont pu survenir lors d'une synchronisation, il est ensuite consultable par l'utilisateur afin de connaître la source de l'erreur. S'il n'y a pas de conflits, ce fichier n'est pas créé.

I - Réflexion

I - Réflexion :

Avant de se lancer dans la rédaction du script nous sommes d'abord passé par une phase de réflexion.

Pour notre script nous avons commencé par réaliser des schémas afin de bien visualiser la tâche qui devait être réalisée dans les grandes lignes. Rien qu'en essayant de faire fonctionner ce script manuellement des idées ont été dégagées. Qu'elles s'avèrent bonnes ou mauvaises par la suite ce n'était pas là le problème, nous voulions seulement éviter de s'enfermer dans une seule idée, s'enfoncer dans celle-ci sans penser à des issues de secours ou à d'éventuels problèmes qui pouvaient survenir.

Un autre élément important est que nous voulions que notre script soit assez modulaire.
Cela permet d'identifier plus rapidement un problème s'il y en a un, il est également plus simple d'apporter des modifications au code si nous voulons le faire évoluer.
Pour répondre à ce besoin de modularité, nous avons décidé de découper le script en différentes fonctions qui ont chacune un rôle bien défini.

Le fichier *.synchro*, qui est notre fichier de journalisation est également un point important du projet. Nous avons déterminé à quoi devait ressembler une entrée dans ce fichier mais la question était : Comment récupérer ces informations ?
Dans un premier temps nous avons pensé à la commande *ls –l*. Elle permet de retrouver la plupart des informations que l'on cherche, puis avec des commandes telles que *grep* ou *cut* il nous est facile d'isoler ce que l'on cherche précisément. Mais en effectuant des recherches nous avons trouvé la commande *stat* qui permet d'afficher l'état d'un fichier. Avec l'option *-c* nous pouvons personnaliser l'affichage ce qui nous facilite la tâche et économise des commandes (*grep, cut*…).

II – Le synchroniseur

II – Le synchroniseur :

Nous avons essayé d'être le plus clair dans notre code, que ce soit, dans le nommage de nos fonctions, de nos variables et dans nos commentaires. Ainsi, il serait redondant et très long d'expliquer en détail ce que font nos commandes, c'est pourquoi nous allons essayer de résumer le fonctionnement du script dans son ensemble.

Avant tout, il faut compléter les 3 variables globales du script à savoir :

- *Chemin_absolu* qui correspond à l'endroit où se trouveront les fichiers de journalisation et de conflits.
- *Chemin_branche_A et chemin_branche_B* qui correspondent aux répertoires à synchroniser.

```
1  #!/bin/bash
2
3  #branches à synchroniser
4  chemin_absolu='/home/nicos/'
5  chemin_branche_A='/home/nicos/A'
6  chemin_branche_B='/home/nicos/B'
7
```

C'est la seule chose que l'utilisateur du programme doit modifier pour le bon fonctionnement du synchroniseur.

Lors de l'exécution, le script vérifie l'existence du journal de conflit (*existence_journal_conflit*) car, s'il existe on le supprime. Nous souhaitons que le journal de conflit dresse un bilan des conflits qu'il y a eu lors d'une synchronisation, nous ne voulons pas que ce soit un historique de tous les derniers conflits des synchronisations précédentes.

Ensuite le script vérifie l'existence du journal de synchronisation (*existence_journal_synchro*). Lors de la 1ère synchronisation nous créons ce journal puis nous avons fait le choix de toujours le garder. Pour le mettre à jour, nous supprimons les lignes obsolètes ou bien nous journalisons les changements en écrivant à la fin du fichier.

S'en suit une phase de comparaison des deux arbres (*comparaison_A_par_rapport_a_B* et *comparaison_B_par_rapport_a_A*), ces fonctions effectuent un travail préliminaire à la synchronisation. Il faut s'imaginer que les deux arbres sont actuellement différents, des fichiers existent dans *B* et non dans *A* et inversement. Le but est de rendre les deux arbres identiques, c'est-à-dire que pour tout dossier ou fichier *pA*, il existe un dossier ou fichier *pB*.

Pour cela nous avons décidé de mettre l'utilisateur au cœur de la synchronisation car nous estimons que seul lui connaît le résultat qu'il désire. Lors de cette phase, le script repère les fichiers ou dossiers qui ne sont présent que dans l'un des deux arbres, ensuite il donne 2 choix à l'utilisateur qui sont :

- Supprimer le fichier/dossier afin qu'il ne soit pas synchronisé. Si une entrée journal existait pour ce fichier/dossier, on la supprime.
- Copier/coller le fichier/dossier dans l'autre arbre afin qu'il soit synchronisé et on effectue une entrée dans le journal de synchronisation.

Ainsi en faisant cela pour chaque dossier/fichier de A puis de B nous arrivons à deux arbres pouvant paraître identique, les métadonnées et le contenu des fichiers peuvent cependant être différent. On passe donc à la prochaine étape de la synchronisation qui est la fonction *parcourt_des_arbres*.

Cette fonction parcourt les deux arbres en parallèle afin de vérifier que les types de fichiers sont les même. C'est-à-dire si *pA* est un dossier, alors *pB* doit en être un également. Si nous sommes dans le cas ou *pA* est fichier et *pB* un dossier (ou inversement), nous faisons face à un conflit que nous avons décidé de gérer.

Encore une fois nous faisons appelle à l'utilisateur qui se voit offrir 3 options :

- Supprimer le conflit, en supprimant le *pA* et le *pB*.
- Transformer les deux en fichiers, on supprime celui qui est un dossier pour venir copier/coller le fichier à la place.
- Transformer les deux en dossier, on supprime celui qui est un fichier pour venir copier/coller le dossier à la place.

Il reste cependant un conflit que nous avons fait le choix de ne pas gérer et nous allons expliquer pourquoi.

Lorsque *pA* et *pB* sont des fichiers mais qu'aucun d'eux ne correspond au journal de synchronisation de par le changement de leurs métadonnées, le script créer le journal de conflit afin d'y écrire l'incident.

Tout d'abord nous n'affichons pas la différence de contenu des deux fichiers pour différentes raisons. Il faut être sûr d'avoir le droit de lecture du fichier, ce qui n'est pas toujours le cas, et il se peut que les fichiers soient très différents. Ainsi afficher des pages entières de ce qui diffère n'est pas vraiment pertinent. De plus, le fait que les fichiers soient différents du journal de synchronisation peut simplement venir des métadonnées, et non du contenu, comme par exemple des droits d'accès.

Voilà ce qui nous a décidé de créer le journal de conflit. Ainsi, l'utilisateur a le temps, à la fin de la synchronisation, d'aller consulter les fichiers qui posent problèmes afin de gérer ce conflit de la manière qu'il le souhaite.

Conclusion

Conclusion :

Le projet a été un travail très enrichissant qui a permis de mettre en pratique un grand nombre de connaissances que l'on a pu acquérir durant l'UE.

Le fait d'avoir beaucoup réfléchi avant de nous avoir lancé nous a permis de ne pas s'égarer durant la réalisation du synchroniseur. Il y avait de nombreuses manières de répondre au sujet mais notre choix a été de placer l'utilisateur au centre de la synchronisation et c'est ce que nous avons réalisé en lui laissant seulement des choix simples à effectuer.

Evidemment tout n'a pas fonctionné du premier coup, c'est pourquoi nous prenions beaucoup de notes sur notre travail afin de pouvoir échanger, trouver de nouvelles solutions et surtout garder des traces de ce qui ne fonctionnait pas toujours. Le fait de découper notre code en différentes fonctions nous beaucoup aider à ce niveau car nous pouvions repérer facilement d'où venait les erreurs. C'était plus simple pour les repérer, les tester et les corriger.