UE141 – Systèmes d'exploitation - TP noté

1) Présentation

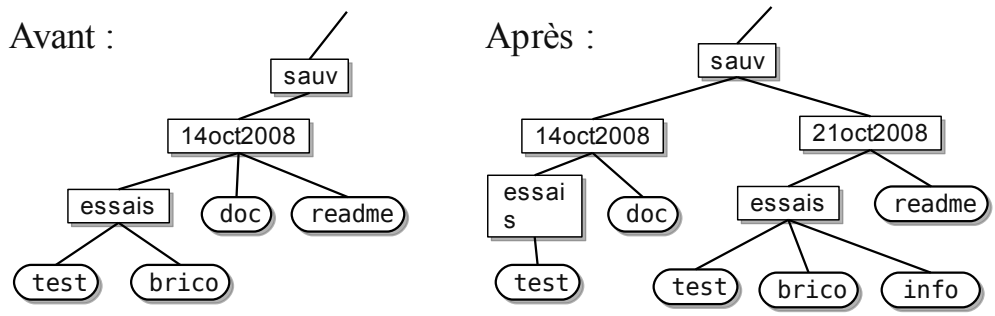
On voudrait disposer d'une commande (script) pour archiver des fichiers et des répertoires en préservant toutes les versions. On fournit une liste de fichiers et de répertoires à cette commande et :

1) elle les recopie dans un nouveau répertoire nommé d'après la date du jour, en préservant toutes les attributs des fichiers (date, protection…)
2) elle parcourt la *précédente* archive pour en supprimer les fichiers inchangés depuis *elle*.

En d'autres termes, à un moment donné, on dispose d'un répertoire dans lequel on a fait des sauvegardes avec notre logiciel. Chaque sauvegarde est dans un dossier portant la date de la sauvegarde. La sauvegarde la plus récente contient la totalité des fichiers qui existaient au moment où cette sauvegarde a été faite. Les sauvegardes plus anciennes ne contiennent que les fichiers dans leur précédents états. Quand on fait une nouvelle sauvegarde, on élimine les doublons de la précédente sauvegarde.

NB : ce mécanisme ne permet pas de savoir tout ce qu'il y avait précisément à une ancienne date donnée, mais seulement de retrouver les versions précédentes des fichiers.

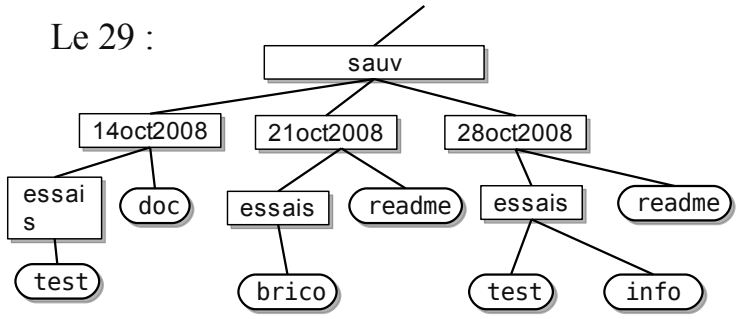
Voici un exemple :



Initialement, la sauvegarde la plus récente a été faite le 14 octobre 2008. Entre le 14 et le 21, on a édité le fichier test, créé le fichier info et supprimé le fichier doc. Le 21, on fait une nouvelle archive avec notre script. Donc, dans le répertoire 21oct2008, on retrouve la totalité des fichiers présents ce jour-là. Par contre dans 14oct2008, on n'y trouve plus que le fichier test (et son répertoire) et doc, puisque les autres fichiers n'ont pas changé et sont aussi dans 21oct2008 ; doc est resté car il n'est plus dans les fichiers actuels et test est resté aussi car il a été modifié.

Si on n'avait touché à rien entre le 14 et le 21, l'archive du 14 aurait été entièrement supprimée.

On continue à travailler après cet archivage : on édite le fichier readme et on supprime le fichier brico. L'archivage du 28 octobre donne ceci :



brico est laissé dans le répertoire du 21 puisqu'on ne le trouve plus dans le 28, readme est laissé dans le 21 car il diffère de celui du 28. Par contre, test est enlevé de la sauvegarde du 21 car il est le même que dans celle du 28.

La sauvegarde du 14 n'a pas été affectée par l'opération. Si on doit faire un nouvel archivage, on ne regardera que le répertoire du 28 et celle du 21 restera inchangée.

NB : la liste des fichiers à archiver doit toujours être la même. Par exemple, c'est le compte de l'utilisateur ou un répertoire fixe codé en dur dans le script.

2) Réalisation

1. Étude

Première chose : écrire l'algorithme abstrait en langage de description très schématique, de préférence pas sous forme d'organigrammes, mais de pseudo langage de programmation. Cette spécification permet de comprendre les idées du programme ainsi que voir comment ça peut être programmé. En particulier, il faudrait dessiner quelques schémas montrant le avant et le après, les tests effectués pour prendre des décisions, les effets de ces décisions...

Méditer ce proverbe : « Listing sans analyse, ni explication, ni exemple, ni schéma n'est que ruine d'ordinateur. »

2. Codage et test

Pour le codage, vous avez le choix du langage de script : Bash, JScript ou VBScript (ou Python pour les passionnés, Perl à condition que la syntaxe soit aussi lisible que celle de Bash).

Sous Unix, des astuces permettent de ne pas écrire un programme récursif pour parcourir les arbres de fichier : la commande tree avec une certaine option permet de générer une liste de chemins relatifs qu'on traite ensuite un par un – voir le corrigé du DS. Sous Unix toujours, il existe une commande qui permet de savoir si deux fichiers sont identiques.

Ces deux traitements devront être programmés explicitement sous Windows. Il y a des snippets sur le serveur qui montrent comment on traite une arborescence de fichiers récursivement.

3. Dossier à rendre

- La seule chose vraiment utile ce sont les explications : quelle est l'idée, comment ça marche, qu'est-ce qui se passe, qu'est-ce qu'on teste, qu'est-ce qu'on fait aux fichiers, qu'est-ce qu'il y a comme erreurs possibles. Avec ça, l'idéal, c'est un bon exemple décomposé pas à pas.

  NB : un commentaire de listing ne vaut rien. Il répond seulement à la question « comment c'est fait ? », mais pas « pourquoi c'est fait comme ça ? » – or c'est cette dernière qui est la seule importante quand il faut garantir la pérénité du logiciel : si quelqu'un reprend votre logiciel, il voudra savoir pourquoi vous faites comme ci et comme ça, pourquoi tel test, pourquoi telle boucle... Paraphraser le listing, c'est faire simplement une traduction anglais vers français mais ça n'explique rien. Donc, expliquez vos idées sous forme littéraire, sur des schémas et des exemples, puis montrez comment on code les points clés en script (ex : comparaison des dates ou des contenus en bash, obtention de la date du jour...).
- Éventuellement le listing, mais franchement c'est pas la peine s'il n'est pas bien indenté ni commenté (titre, auteur, date + chaque étape du script). Bon évidemment, il faut rendre le listing, mais ce n'est pas ça le principal, un listing rendu seul ne vaut rien. Pas la peine de commenter chaque instruction, sauf si elle contient des aspects assez pointus de syntaxe, par contre, il faut commenter des groupes d'instructions qui correspondent à des phases de traitement détaillées dans les explications.
- Une preuve de fonctionnement sous forme d'une page image de l'écran, avant, pendant et après exécution, et une ou deux indications au surligneur (TheGimp) des endroits où on voit que quelque chose d'important s'est passé. Cette sorte de listing doit toujours être assez réduite à la partie vraiment pertinente, sinon on ne sait pas où regarder.
- Une conclusion présentant un bilan du TP : combien de temps avez-vous passé dessus, qu'est-ce qui a été difficile, qu'est-ce qui reste à faire, vous semble-t-il utile, intéressant…

4. Dates limites

Il faut préparer le dossier complet sous la forme d'un fichier pdf (utilisez OpenOffice, il y a de très bons outils) accompagné du script. Nommez ces fichiers à l'aide de votre nom : ex: Jean Dupont – Compte-rendu.pdf et Jean Dupont – script.

Le tout est à déposer dans l'espace de rendu de TP de votre ENT, dans la zone de dépôt de Pierre Nerzic, dans LP GSR – UE 141 – TP noté (prochainement créée). La date est fixée au 30 novembre 2008 minuit (négociable si autre TP à rendre en parallèle).