


Enoncé du TP 6 Système

C. Pain-Barre

INFO - IUT Aix-en-Provence

version du 27/11/2012


 Démarrer les PC sous Linux. Les exercices sont à faire via une connexion SSH (normale) sur allegro.

1 Variables et Tableaux (suite)

Exercice 1


Manipulation de tableaux

1. Créer un tableau **tab** contenant les trois éléments **aaa**, **bbb** et **ccc**
2. Faire afficher le contenu de l'élément 0 de **tab**
3. Sans recréer le tableau, lui ajouter l'élément **ddd** (en position 3)
4. Utiliser (la valeur de) **var1** (qui devrait contenir 3) pour faire afficher le contenu de l'élément 3
5. En une seule commande qui utilise et modifie **var1**, faire afficher l'élément 3 de **tab** tout en faisant en sorte que **var1** vaille 2 à l'issue de la commande. Comme en C/C++, cela se fait simplement avec une post-décrémentation...
6. Supprimer l'élément 3 de **tab**
7. Vérifier en le faisant afficher : cela doit écrire une ligne vide
8. Créer l'élément 3 de **tab** en lui donnant comme valeur le **contenu exact** de **var2**
9. Vérifier en faisant afficher cet élément. Les espaces contenus dans **var2** doivent apparaître
10. Faire afficher le nombre d'éléments du tableau **tab**
11. Faire afficher le contenu exact (de tous les éléments) du tableau **tab**
12. Taper **OIFS="\$IFS"** pour copier le contenu de la variable **IFS** dans la variable **OIFS**
13. Modifier **IFS** pour lui donner la valeur : (deux-points)
14. Comparer le résultat des commandes suivantes :
 - (a) **echo "\${tab[*]}"**
 - (b) **echo "\${tab[@]}"**
15. En utilisant **OIFS** redonner à **IFS** son ancienne valeur
16. En utilisant les substitutions étendues — plus exactement des sous-chaînes appliquées à un tableau entier pour n'en garder qu'une partie — créer un tableau **soustab** contenant les éléments 2 à 3 de **tab**
17. Faire afficher le contenu exact de **soustab** (les espaces de l'élément 1 de **soustab** doivent apparaître)
18. Faire afficher le contenu du tableau **GROUPS**, créé par bash pour contenir la liste des groupes de l'utilisateur en session.


 Le premier nombre est le *gid* du groupe primaire (actuel si modifié avec **newgrp**) de l'utilisateur. Il est suivi des numéros de ses groupes secondaires.

Exercice 2


Utilisation des commandes **alias/unalias** pour créer/supprimer des commandes "raccourcies"

 Vérifier vos réponses en faisant afficher à chaque fois la liste des alias avec **alias**.

1. Exécuter **alias** pour voir la liste des alias existants.

 Selon l'installation, plusieurs alias peuvent être créés par défaut, tels que **ls**, **cp**, **mv** et **rm** qui se substituent aux commandes d'origine, pour les exécuter en activant certaines options (comme **-F** et **--color=auto** pour **ls**, et **-i** pour **cp**, **mv** et **rm**).

2. Exécuter **ls -l ~** et constater que certains noms de fichiers (comme les répertoires) apparaissent colorés¹.
3. Reprendre la question précédente mais en empêchant l'expansion de l'alias. Aucun fichier ne doit alors apparaître coloré.
4. Sur le modèle de **ls**, créer un alias **cp** qui en active l'option **-i**. Après en avoir contrôlé la création, le tester en tentant d'écraser un fichier par copie. Répondre **n** à la question qui devrait alors vous être posée, afin d'annuler la copie.
5. Créer un alias **del** pour une commande de suppression de fichiers sans confirmation (utiliser l'option **-f** de **rm**). Le tester en supprimant une copie d'un fichier quelconque, sur laquelle vous vous serez enlevé le droit d'écriture. Aucun avertissement ne devrait être affiché.
6. Créer un alias **tx** qui vous place dans votre répertoire `tpunix` (en fait votre répertoire de travail), quel que soit votre répertoire de travail actuel. En d'autres termes, depuis n'importe quel répertoire où vous vous trouvez, en tapant **tx** vous devrez vous déplacer dans votre répertoire `tpunix`. Cet alias doit être aussi générique que possible, en évitant de faire apparaître "en dur" le chemin de votre répertoire d'accueil. Pour le tester, vérifier que son exécution à partir d'un répertoire comme `/bin` vous ramène bien dans `tpunix`.
7. Créer un alias **copy** pour une commande de copie de fichiers sans confirmation en cas d'écrasement mais en mode verbeux. Attention !! Il ne faut pas utiliser l'alias de **cp** car l'option **-i** prend le pas sur l'option **-f** ! Tester ensuite **copy** pour copier deux fois `cigale.txt` de `tpunix` dans le répertoire `tp`. Si le résultat est celui attendu, supprimer le fichier `cigale.txt` de `tp`.
8. Supprimer l'alias **copy**.
9. Créer un alias **llh** qui affiche en paginant la liste détaillée des fichiers (cachés ou non) du **répertoire d'accueil** de l'utilisateur en session. Plus clairement, l'alias doit être générique et fonctionner quel que soit l'utilisateur.
- 10.

 Pour cet alias, on a besoin du caractère spécial ; (*point-virgule*) qui permet de séparer plusieurs commandes sur une seule ligne.


Créer un alias **ccd** qui, comme **cd**, change le répertoire de travail mais, **avant** de le faire, stocke (ajoute) dans le fichier `~/ .wds` le répertoire de travail actuel. En d'autres termes, **ccd** place dans `~/ .wds`

1. La personnalisation des couleurs utilisées par **ls** est le thème d'un prochain exercice facultatif.

une trace des répertoires de travail. Le tester et vérifier qu'en changeant plusieurs fois de répertoire par **ccd**, le fichier `~/ .wds` est correctement étendu.


 Cet alias aurait pu s'appeler **cd**. Cela ne gêne pas bash.

11. Supprimer l'alias **ccd** et le fichier `~/ .wds`
12. Pourquoi n'est-il pas possible d'écrire un alias similaire à **ccd** mais qui n'ajouterait le répertoire de travail dans `~/ .wds` qu'**après** s'être déplacé ?


 Nous créerons des alias persistants sur les PC la prochaine séance.

Exercice 3


Écriture de fonctions

 Lorsqu'une fonction prend en argument(s) un (plusieurs) nom(s) de fichier(s), protéger systématiquement la substitution de cet (ces) argument(s) avec des guillemets, au cas où un nom de fichier contiendrait des caractères spéciaux.


1. Écrire une fonction **ccd** semblable à l'alias **cd** mais cette fois il faut que ce soit le répertoire de travail **après** le changement effectif de répertoire qui soit ajouté dans le fichier `~/ .wds`

 Rappel : si l'on veut définir une fonction sur une seule ligne, il faut faire précéder la dernière accolade par un ;

2. Écrire une fonction **absolue** qui prend en argument une référence (relative ou absolue) d'un répertoire et qui écrit la référence absolue de ce répertoire. On suppose qu'il existe.
Aide : il faut se déplacer dans le répertoire en question, écrire le répertoire de travail, puis revenir au répertoire de travail précédent.

 Quand on utilise **- (tiret)** en argument de **cd**, on revient au répertoire précédent, mais celui-ci est écrit par **cd** sur sa sortie standard. Si on ne veut pas qu'il soit écrit, il faut rediriger la sortie standard vers le fichier `/dev/null`.

3. Écrire une fonction **del** qui crée le répertoire `~/ .trash` s'il n'existe pas déjà (voir les options de **mkdir**) et qui déplace en mode bavard les fichiers passés en arguments dans ce répertoire.

 Le répertoire `~/ .trash` sert ainsi de corbeille autorisant une restauration de fichiers supprimés. Néanmoins, c'est une solution très simplifiée car deux fichiers de même nom ne pourront pas être placés ensemble dans la corbeille, et les emplacements de restauration ne sont pas sauvegardés.

4. Écrire une fonction **undel** qui prend un seul fichier en argument, et qui déplace (restaure) en mode bavard ce fichier depuis le répertoire `~/ .trash` et le met dans le répertoire de travail.
5. Écrire (puis tester) une fonction **interv** qui prend 3 arguments et qui affiche le nombre de lignes indiqué en argument 2, à partir de la ligne indiquée en argument 1 du fichier indiqué en argument 3. Aucun calcul n'est nécessaire dans l'écriture de cette fonction. On suppose que les arguments numériques sont corrects.

Exemple :

```
$ interv 3 5 des_lignes.txt
ceci est la ligne n°3
ceci est la ligne n°4
ceci est la ligne n°5
ceci est la ligne n°6
ceci est la ligne n°7
```

⇨ affiche 5 lignes à partir de la ligne 3 du fichier `des_lignes.txt`

6. Écrire une fonction **extremes** qui prend un nombre quelconque d'arguments et qui écrit sur sa sortie uniquement son premier et son dernier argument, chacun sur une ligne. S'il n'y a qu'un argument, il sera écrit 2 fois. S'il n'y en a aucun, affiche deux lignes blanches.

Tester la fonction en tapant :

```
$ extremes 'a_a' bb cc 'd_d'
```

qui doit afficher :

```
a_a
d_d
```

puis en tapant :

```
$ extremes *.txt
```

qui doit afficher le premier et le dernier fichier correspondant au motif `*.txt`

Exercice 4

Écriture d'une fonction facilitant la compilation avec g++

Au cours des TP d'algorithmique/C++, vous devez **compiler** régulièrement vos fichiers sources (programmes) pour fabriquer des fichiers exécutables. Dans ce but, vous avez probablement déjà utilisé la commande **g++** (le compilateur C++ GNU).

Rappels sur g++.

On compile un fichier `fichier.cxx` avec **g++** en exécutant la commande :

```
g++ fichier.cxx
```

qui, si tout se passe bien, produit un fichier exécutable **a.out** au format ELF (*Executable and Linkable Format*).

Il est possible de spécifier un nom que doit porter le fichier exécutable, en utilisant l'option **-o référence** comme dans la commande :

```
g++ -o fichier fichier.cxx
```

qui produit le fichier exécutable `fichier` à la place de `a.out`.

Enfin, il est généralement sage de demander à **g++** d'écrire des avertissements lorsqu'il détecte du code qu'il juge hasardeux. Cela se fait en utilisant l'option **-Wall**, ce qui donne finalement :

```
g++ -Wall -o fichier fichier.cxx
```



L'absence d'avertissement ne signifie pas que le code est correct ! Inversement, certains avertissements ne signifient pas que le code est incorrect. Cependant, les avertissements de `-Wall` ne doivent jamais être négligés !

1. Se placer dans `tpunix`
2. Copier le fichier `~cpb/public/unix/sorties.cxx` dans le répertoire de travail. Afficher le contenu de votre copie.
3. S'il existe, renommer votre fichier `sorties` en `oldsorties`
4. Utiliser `g++` pour compiler (avec avertissements) `sorties.cxx` et produire un exécutable nommé **`sorties`**
5. Exécuter `sorties`, ce qui devrait afficher quelques messages (familiers)
6. Définir une fonction **`mk`** qui prend en argument le nom du fichier exécutable que l'on veut obtenir en compilant avec avertissement un fichier source C++ de même nom mais d'extension `.cxx`

Exemple :

```
mk bonjour
```

doit exécuter la commande : `g++ -Wall -o bonjour bonjour.cxx`

Aide : On peut coller une chaîne au contenu d'un paramètre positionnel. Par exemple, si le paramètre positionnel `$1` (le premier argument) contient `bonjour`, alors `$1.cxx` sera remplacé par `bonjour.cxx`

7. Supprimer le fichier `sorties`
8. Utiliser **`mk`** pour produire l'exécutable `sorties` à partir de `sorties.cxx`
9. Exécuter `sorties`, ce qui devrait afficher les messages habituels



Nous verrons au prochain TP comment rendre des fonctions persistantes.

Exercice 5

*Étude de la variable (d'environnement) **`PATH`***



Rappels. La variable **`PATH`** contient la liste des chemins, séparés par deux-points `:"`, dans lesquels `bash` recherche les commandes externes (nom d'un fichier exécutable d'une application). `bash` n'effectue cette recherche que lorsque le premier mot de la ligne de commande ne contient pas de slash et ne correspond pas à une commande interne, un alias, une fonction ou un mot clé.

1. Faire afficher le contenu de la variable **`PATH`**
2. Taper **`type ls`**
On apprend que c'est un alias (mais pas quel est l'exécutable **`ls`** qui est réellement exécuté).

3. Taper **type -P ls**

Cela devrait afficher `/bin/ls`. En effet `/bin` est le premier chemin de **PATH** qui contient un fichier nommé `ls`

4. Taper **type sorties**

bash devrait indiquer qu'il ne trouve pas cette commande. En effet, elle n'est située dans aucun chemin du **PATH** et est donc inconnue.

5. Tenter d'exécuter la commande :

sorties

qui devrait produire une erreur car cette commande externe n'est pas trouvée, ce qui explique qu'on l'a exécutée avec `./sorties` jusque là.

6. Pour que la recherche des commandes externes se fasse aussi dans le répertoire de travail, on peut l'ajouter dans le **PATH**, de préférence à la fin. Taper la commande :

PATH="\$PATH:."

qui ajoute à la fin du **PATH** le répertoire de travail (répertoire `.`)



Si le **PATH** contient le chemin `.` (*point*), pour exécuter un fichier du répertoire de travail, il n'est plus nécessaire de faire précéder son nom de `./`.

7. Taper à nouveau la commande :

sorties

qui devrait s'exécuter normalement.



Placer dans son PATH le répertoire `.` avant les répertoires normaux (`/bin`, `/usr/bin`, etc.) n'est vraiment pas une bonne idée !! Car dans ce cas, on s'expose à de graves problèmes de sécurité, comme l'illustre la suite.

8. Ajouter le répertoire de travail en début du **PATH** (ce n'est pas grave qu'il y figure aussi à la fin)9. Aller dans le répertoire `~cpb/public/bin` puis afficher son contenu. Que se passe-t-il ?

Aide : utiliser le vrai **ls** (`/bin/ls`) pour voir le contenu de ce répertoire.

10. Redonner à votre **PATH** une valeur moins dangereuse en supprimant le premier chemin (utiliser une substitution étendue). Vérifier en affichant son contenu.