Programmation système et réseaux
Corrigé du TP 4: Alias et fonctions

Arnaud Giersch Guillaume Latu

Les alias

Les alias constituent un moyen de réaliser des substitutions de chaînes au sein d'un shell. Le shell compare toute commande passée aux éléments de la liste courante des alias. De plus, si le dernier caractère d'un alias est un espace, il analyse aussi si le mot qui suit est un alias. Pour désactiver la recherche d'alias préfixer n'importe qu'elle commande de `\`. La commande `alias` permet d'obtenir la liste des alias définis. Pour supprimer un alias, on utilise la commande `unalias`.

1. Tapez les commandes suivantes :

   ```
   alias test='echo'
   alias me='ceci est un alias'
   alias
   test me
   \test me
   alias test='echo '
   test me
   test \me
   unalias test me
   alias
   ```

2. Lorsqu'un script est exécuté, un nouveau shell est démarré pour exécuter les commandes constituant ce script. Nous allons le voir sur un exemple. Créer un script `s1` contenant : `echo mon PID est $$`. La variable spéciale `$$` donne le PID du processus. Si vous exécutez le script `./s1` plusieurs fois de suite, la valeur de PID affichée change. Cela vous parait-il normal ?

   Correction : *C'est normal, un nouveau shell est lancé à chaque fois.*

3. Nous allons maintenant voir une nouvelle manière d'exécuter un script. Essayez d'exécuter la commande suivante : `source s1` (remarque : on peut aussi utiliser `.` au lieu de `source`, comme dans `. s1`). Que constatez-vous ? Utilisez le manuel pour comprendre ce que fait la commande `source`.

   Correction : *Le PID affiché est toujours le même (celui du shell courant). Avec ce mode lancement particulier, les commandes du script sont réalisées au sein du shell courant.*

4. Mettez la définition d'alias `alias l='ls -l '` dans le fichier *~/mesalias*. Comment faut-il exécuter ce script pour que l'alias soit défini dans le shell courant ?

   Correction : *Utiliser :* `source ~/mesalias`.

Les fonctions

Dans un shell, on peut définir des fonctions qui peuvent remplir le rôle de scripts. La différence entre scripts et fonctions, c'est qu'à l'exécution d'un script il faut accéder au fichier qui le contient, alors que ce n'est pas le cas pour les fonctions qui sont définies au sein du shell (on peu comme des variables).

Pour sortir d'une fonction, on peut utiliser la commande `return` (cela modifiera la valeur de retour `$?` au niveau du shell). Les commandes `typeset -f` ou `declare -f` listent les fonctions définies. Pour exporter une fonction vers

les shells fils, on utilisera `export -f <nom_de_la_fonction>`. La commande `type` permet de savoir de quel type (alias, script, fonction, ...) est une certaine commande exécutable ; par exemple : `type ls; type test`. Dernière remarque : les fonctions bash peuvent être récursives.

1. Tapez les commandes suivantes :

```
exf () { ps -u $USER | grep $1; }
exf bash
```

2. Écrivez une petite fonction qui permet de déterminer si l'exécution d'une fonction bash se fait au sein du shell courant ou s'il y a création d'un nouveau shell à chaque appel de fonction.

   Correction : *En utilisant la fonction* `testpid () { echo mon PID est $$; }`, *on constate que PID affiché est toujours le même. Les fonctions sont exécutées au sein du shell courant, il n'y a pas de création de nouveau processus (contrairement aux scripts).*

3. Écrivez une fonction bash affichant le contenu d'un fichier dont le nom est passé en paramètre. Vous ferez précéder chacune des lignes par son numéro de ligne (un peu comme avec `cat -n`), ainsi que la longueur de la ligne. Vous pouvez utiliser la commande `printf` (page de manuel accessible avec la commande `man -s 1 printf`).

   Correction :

```
mafonction () {
    i=1
    ([ $# -ge 1 ] && cat "$1" || cat) \
    | while read line; do
            len=$(printf %s $line | wc -c)
            printf "%4d %4d %s\n" $i $len "$line"
            let i=$i+1
    done
}
```

4. L'objectif de cet exercice est de créer une commande `monrm` qui déplacera les fichiers dans un répertoire `~/.poubelle` au lieu de les effacer.
   - Créez une fonction `monrm()` contenant la suite de commandes nécessaire pour déplacer tous les fichiers passés en argument sur la ligne de commande dans le répertoire `~/.poubelle`.
   - Ajoutez l'option `-c` à la commande telle que `monrm -c` affiche la taille du contenu du répertoire poubelle. vous pourrez utiliser à cette fin la commande `du -sk`.
   - ajouter l'option `-e` permettant de vider le contenu de la poubelle.
   - ajouter l'option `-h` affichant la mini aide en ligne suivante :
     `Usage : monrm [-h|-e|-c] [fichier]...`
   - Pour que la fonction `monrm` soit accessible dans n'importe quel shell, vous pouvez placer la définition de la fontion dans votre fichier `~/.bashrc` qui est lu à chaque fois qu'un bash est lancé.

   Correction :

```
monrm () {
    case "$1" in
    -h)
        cat<<EOT
Usage : monrm [-h|-e|-c] [fichier]...
EOT
        ;;
    -c)
        du -sk ~/.poubelle
        ;;
    -e)
        rm -rf ~/.poubelle/*
        ;;
    *)
        mv "$@" ~/.poubelle
```

```
    esac
}
```