Shell & Scripts

Le but de cette partie est d'approfondir la notion de **shell-script** sous Unix. Vous pourrez constater au cours de ces séances que le **shell** est plus qu'un interpréteur de commandes, c'est également un *langage de programmation* à part entière.

On rappelle le fonctionnement d'un shell :

1) phase de saisie de la ligne de commande
2) phase de substitution de motifs ($\sim$, $*$, ?, `$`...)
3) phase d'exécution de la commande

Le **shell** que vous utilisez se nomme le `bash`[1]. Le `bash` utilise des fichiers de configuration, qui se trouvent dans votre répertoire d'accueil. Par exemple, le fichier **.bashrc** sert à définir les initialisations des sessions interactives (rattachées à un terminal). Le fichier **.bash_profile** permet de personnaliser ses sessions.

Ces fichiers sont exécutés par le **shell** s'ils se trouvent dans votre répertoire d'accueil. Vous pouvez constater que ces fichiers sont présents dans votre espace de travail, et vous êtes invités à les consulter. Vous pouvez également en modifier le contenu afin de personnaliser votre environnement de travail.

Important : Faites une sauvegarde préalable de vos fichiers de configuration avant d'y effectuer des modifications.

1 Notion de variable

Sous votre shell, il est possible de définir des variables. Par exemple sous votre environnement est définie une variable `HOME` contenant le chemin de votre répertoire d'accueil.

1.1 Exercice Exécutez les commandes `echo HOME` et `echo $HOME`. Qu'en déduisez-vous ? Comment accède-t-on au contenu d'une variable ?

1.2 Exercice Afficher le contenu de la variable `FOO`.

1.3 Exercice Exécutez la commande `FOO=HOME`. Afficher le contenu de la variable `FOO`.

1.4 Exercice Exécutez la commande `FOO=$HOME`. Afficher le contenu de la variable `FOO`.

1.5 Exercice Observer et expliquer la suite de commandes :

```
echo $A
A=aaaaa
echo $A
bash
echo $A
```

1. Le *GNU Bourne-Again SHell* est du domaine public et a été développé par la *Free Software Foundation* sous licence GNU (si ce n'est déjà fait, je vous engage à visiter, **après le TD**, l'URL : `http://www.gnu.org`).

```
exit
export A
bash
echo $A
A=bbbbb
echo $A
exit
echo $A
```

1.6 Exercice Créez un répertoire ∼/bin. Désormais tous vos scripts seront créés dans le répertoire ~/bin. Déplacer dans ce répertoire les premiers scripts écrits lors des séances précédentes.

1.7 Exercice Modifiez votre environnement de façon à ce que les scripts inclus dans le répertoire ∼/bin soient utilisables depuis n'importe quel endroit sans en spécifier le chemin d'accès.

1.8 Exercice La modification de la variable PATH reste-t-elle valide lors d'une nouvelle connexion ou lors de l'ouverture d'un nouveau terminal ? Cela dépend de vos fichiers de configuration : la définition de la variable PATH contient-elle votre répertoire ∼/bin ? Si nécessaire, ajoutez à l'endroit approprié la ligne PATH=$HOME/bin:$PATH.

2 Mécanismes de substitution

2.1 Phases de substitution

La phase de substitution de la ligne de commande présente elle-même plusieurs sous-phases :
— substitution de commandes
— substitution de variables
— substitution de chemins

2.1 Exercice Interprétez et commentez le résultat des commandes suivantes (le dollar en début de chaque ligne est le "prompt" ou "invite de commande", à ne pas confondre avec le dollar à l'intérieur de la ligne de commande)

1) Substitution de commandes

```
$ echo ls -l
$ echo $(ls -l)
```

2) Substitution de variables

```
$ ab=toto
$ echo $ab
$ echo ${ab}ababab
$ echo $abababab
$ y=p ; ${y}wd
```

3) Substitutions de chemins

```
$ echo ~/.*
$ echo ~oldelmas/b*
$ cd
$ echo */*
$ echo ../?[x-z]*
$ echo ../?[x-z]?
```

4) Substitution arithmetique (ici le symbole ␣ représente l'espace)

```
$␣echo␣p$((␣3␣+␣4␣))q
```

2.2 Quotations

Les quotes '...' empêchent toute substitution. Les doubles quotes "..." empêchent les substitutions de chemin ainsi que le découpage en mots.

2.2 Exercice 1) Tester le rôle des quotes et doubles quotes :

```
$ x=p; echo '${x}wd'
$ echo ."*"
$ echo "~ $(pwd)"
$ "ls"
$ "ls -l"
```

2) Expliquer le comportement des commandes :

```
$ A=$(echo *)                    $ ls $A
$ echo '$A'                      $ ls "$A"
$ echo "$A"
```

2.3 Exercice Faire afficher les chaînes suivantes

`***␣BONJOUR␣***`

`␣-␣␣␣Le␣caractere␣'*'`

`/Fichier␣d'entree\`

`*␣samedi␣22␣mars␣2014,␣18:20:45␣(UTC+0100)␣*` (la date doit être le résultat d'une exécution de `date`)

`$$␣samedi␣22␣mars␣2014,␣18:21:35␣(UTC+0100)␣$$` (idem)

`$6248$` (où le nombre entre `$` est le *pid* du *shell*)

`(␣/bin/cat␣/bin/pax␣/bin/tar␣)` c'est-à-dire la liste de fichiers dans le répertoire `/bin` dont le nom est à trois lettres et dont la deuxième lettre est une de lettres a–d (utiliser une substitution de chemins).

2.4 Exercice Écrire la commande `nf` qui affiche le nombre de fichiers ne commençant pas par un point, du répertoire courant. Vous utiliserez les commandes `echo`, `wc` et `pwd`.

Faire l'exercice 0.5 de la feuille Problème.

3 Exécution d'une commande

3.1 Recherche et lancement d'une commande

3.1 Exercice Afficher le contenu de la variable `PATH`. Sauvegardez cette valeur dans une variable `PATHBACKUP` et vérifier la valeur sauvegardée. Mettre la chaîne vide dans `PATH` et essayez les commandes `cd`, `pwd`, `ls`. Que peut on en conclure ? Comment lancer `ls` sans modifier `PATH` ? Modifier la variable `HOME` en lui affectant le chemin absolu d'un de vos répertoire (pas le répertoire d'accueil). Observez ce que produit la commande `cd` sans argument. **Restaurer les valeurs de `PATH` et de `HOME`.**

3.2 Valeur de retour d'une commande et Action conditionnelle

Les variables `PATH` et `HOME` ont en fait été définies dans votre environnement et vous pouvez en modifier la valeur. Mais il existe quelques variables particulières affectées par le shell lui même. L'une d'entre elles se nomme `?`.

3.2 Exercice Exécutez la commande `ls -l` et affichez `$?`

3.3 Exercice Exécutez la commande `ls -z` et affichez `$?`

3.4 Exercice Dans votre fenêtre terminal, exécutez la commande `ls -l && echo OK || echo KO`. Puis exécutez la commande `ls -z && echo OK || echo KO`. Quel est le rôle des opérateurs `&&` et `||` ? Examinez dans chacun des deux cas précédents le contenu de la variable `?`. Expliquez son contenu. Modifiez la commande `ls -z && echo OK || echo KO` de telle sorte que le message d'erreur généré soit redirigé vers `/dev/null`.

3.5 Exercice Écrire la commande `ra` qui affiche `oui` si le répertoire courant est le répertoire d'accueil et `non` sinon. Vous utiliserez les commandes `test`, `pwd` et `echo`.

Faire l'exercice 0.9 de la feuille Problème.

4 Script shells : variables prédéfinies, set et structures de contrôle

4.1 Les variables prédéfinies

Le shell offre des variables prédéfinies facilitant la programmation sous le shell. Vous avez déjà pu observer le rôle de plusieurs de ces variables. Complétez le tableau ci-dessous :

?	
$	
#	
*	
0	
1	
2	

Cherchez éventuellement dans le manuel de `bash` les informations qui vous manquent.

4.1 Exercice Ecrivez dans un script shell les commandes permettant d'afficher le nom absolu du script lancé (chemin depuis la racine + nom de la commande), le nom de la commande seule, le nom du chemin seul. Pour en vérifier le bon fonctionnement, recopiez le script dans un autre répertoire en le renommant et exécutez-le.

4.2 Exercice Ecrivez un script shell affichant son nombre de paramètres et la liste de ses paramètres. On affichera le nom des variables contenant ces informations ainsi que leur contenu. Vérifiez que cela fonctionne en lançant le fichier de commandes avec un nombre variable de paramètres.

Exemple de résultat :

```
$ ./parameters.sh aa bb cc dd
nombre de parametres : $# = 4
liste des parametres : $* = aa bb cc dd
```

4.3 Exercice Écrire un script `prepa` qui évoqué comme `prepa` ≺*nom*≻ crée dans le répertoire courant un fichier exécutable ≺*nom*≻ dont la première ligne est `#!/bin/bash` suivie d'une ligne vide.

4.2 Les commandes set et shift

4.4 Exercice Examinez les scripts qui suivent,

petit-script-1

```
#!/bin/bash
date
echo Il est $5
```

petit-script-2

```
#!/bin/bash
set $(date)
echo Il est $5
```

et exécutez les commandes :

```
$ petit-script-1 donald riri fifi loulou picsou
$ petit-script-2 donald riri fifi loulou picsou
```

afin d'interprétez le rôle de la commande : `set` ≺ *commande* ≻.

4.5 Exercice Trouvez quel est le rôle de la variable `IFS`. En modifiant cette variable, écrivez un script qui affiche :

```
Il est 17h 45mn.
```

4.6 Exercice Écrire la commande `prc` qui affiche la profondeur du répertoire courant. Vous utiliserez les commandes `set`, `pwd`, `shift`, `echo` ainsi que la variable `IFS`.

4.3 L'instruction if

4.7 Exercice Que fait le script suivant :

```
#!/bin/bash

if [ $# = 0 ]
then
    cd ..
else
    cd $1
fi
echo " --> $(pwd)"
```

Tester cette commande et expliquer ce qui se produit à l'exécution (essayez pwd après le script). Essayer en préfixant la commande par `source`. Expliquer.

4.8 Exercice L'instruction `if` peut également utiliser entre les crochets des tests sur des fichiers ou des mots. Exemples :
`[ -f fichier ]` vrai si "fichier" existe et est un fichier régulier,
`[ -z mot ]` vrai si la longueur de "mot" est zero.
Ecrire un script qui fait une archive (à l'aide de la commande `tar -czf nom_archive liste_fichiers`) dont le nom sera donné en paramètre de la commande et qui contiendra tous les fichiers python du répertoire courant. Prévoir des messages d'erreur si la commande n'a pas de paramètre ou si le répertoire ne contient pas de fichiers python.

4.4 Itérations

4.9 Exercice Tester la boucle `for` au moyen des programmes suivants :

```
for N in un deux trois quatre
do
  echo "**$N**"
done
```

```
for N in $(ls)
do
  echo "Fichier -> $N"
done
```

Tester la boucle `while` et la commande `shift` avec le script, `decalage.sh`, suivant

```
while [ $# != 0 ]
do
  echo $1
  shift
done
```

et l'appeler avec `decalage.sh "a b" 'c' d e`

4.10 Exercice On suppose s'être placé dans un répertoire contenant N images numérotées de manière désordonnée et partielle. Exemple :

```
image0002.JPG   image0015.JPG   image0054.JPG   image0120.jpg
originals
image0010.jpg   image0023.jpg   image0067.jpg   image0127.JPG   toto tata
image0012.jpg   image0045.JPG   image0088.jpg   image0147.JPG
```

Le répertoire courant peut comporter d'autres fichiers, et les extensions des fichiers images peuvent être de casse incohérente (jpg et JPG).
Ecrire un script permettant de renuméroter les images de `0` à `N-1` et de fixer toutes les extensions à `jpg`.

Indications : Vous pourrez vous servir de la commande `printf` (penser à regarder le `man`).

Finir la feuille Problème.