Shell (1)

1 Objectif du TD

Le but de cette séance est de vous familiariser avec 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.

2 Généralités

Pour Unix, un fichier contenant des commandes est appelé un **script** et ne nécessite que le droit d'exécution pour pouvoir être exécuté[16]. Lors d'une séance précédente vous avez pu voir qu'il existait de nombreux **shell**, i.e., `sh`, `csh`, `tcsh`, `ksh`, `zsh`, etc. Le **shell** que vous utilisez se nomme le `bash`[17]. 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.

2.1 EXERCICE Déplacez vous dans différents répertoires et observez le comportement de votre *prompt*. Placez vous dans votre répertoire d'accueil et éditez votre fichier `.bashrc`. Commentez la ligne débutant par `PS1`. Rechargez votre fichier **.bashrc** et observez à nouveau le comportement de votre *prompt*. Décommentez la ligne débutant par `PS1` et rechargez votre fichier **.bashrc**. Examinez le rôle de la variable `PS1` dans la page de manuel de `bash`.

3 Avant-propos

Avant de créer et d'utiliser vos propres **script shells**, vous allez vous familiarisez avec diverses petites notions de bases qui pourront être bien utiles lorsque vous voudrez créer vos propres fichiers scripts.

[16] Donc contrairement au système MS-DOS qui nécessitait des fichiers particulier avec l'extention `.BAT`.

[17] 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`). Ce shell contient toutes les propriétés d'interactivité du csh et ksh. Son langage de programmation est compatible avec sh.

4 Les shells

4.1 Empilement

4.1 Exercice Empilez plusieurs shells en tapant successivement `bash`, `bash`, `tcsh`, `csh`, `sh` et observer la parenté des différents processus par la commande `ps -l`[18].

4.2 Exercice Dépilez les shells, en utilisant plusieurs fois et successivement les commandes `exit` et `ps -l` afin de vérifier que vous êtes revenu à votre premier shell dans les liens de parentés.

4.2 Les premiers scripts

Créez un répertoire `script-shell`, et placez vous dedans. Créez les fichiers suivants et exécutez les.

petit-script-1

```
pwd ; ps -l ; whoami
```

petit-script-2

```
set -v
pwd ; ps -l ; whoami
```

petit-script-3

```
set -x
pwd ; ps -l ; whoami
```

Remarque 7 Un script est un fichier texte qui doit être exécuté par un shell ; il est donc normal de positionner soi-même l'accès en exécution à ce fichier.

4.3 Exercice Quelles différences induisent les commandes `set -v` et `set -x` ?

Remarque 8 Pour supprimer les options de la commande `set`, il suffit d'appliquer ce qui est préconisé dans le manuel. On peut y lire :

« `[...] Using + rather than - causes these options to be turned off. [...]` »

[18] Le champ `PID` fait rérérence au numéro du processus, tandis que le champs `PPID` fait référence au numéro du processus père.

5 Exécution d'une commande

5.1 Recherche et lancement d'une commande

Sous votre shell, il est possible de définir des variables. Par exemple sous votre environnement est défini une variable `DISPLAY` contenant le nom de votre terminal.

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

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

5.3 Exercice Exécutez la commande `FOO=DISPLAY`. Afficher le contenu de la variable `FOO`.

5.4 Exercice Exécutez la commande `FOO=$DISPLAY`. Afficher le contenu de la variable `FOO`.

5.5 Exercice Afficher le contenu de la variable `PATH`. Sauvegardez cette valeur dans une variable `PATHSVG` 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 de votre répertoire `script-shell`. Observez ce que produit la commande `cd` sans argument. **Restaurer les valeurs de `PATH` et de `HOME`**.

5.2 Valeur de retour d'une commande

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

5.6 Exercice Éxecutez la commande `ls -l` et affichez le contenu de la variable `?`.

5.7 Exercice Éxecutez la commande `ls -z` et affichez le contenu de la variable `?`.

Remarque 9 La variable `$` retourne le `PID` du shell courant. Afficher donc son contenu, et vérifiez en sa valeur à l'aide de la commande `ps -l`.

5.3 Action conditionnelle

5.8 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`.

6 Mécanismes de substitution

Interprétez et commentez le résultat des commandes suivantes :

1) Substitution des accolades
```
$ echo a{ab,c,edf}f
$ mkdir ~/Textes/a{1,2,5}
```

2) Substitution du caractère tilde
```
$ echo ~/bar
$ echo ~odelmas/bar
```

3) Substitution de variables
```
$ ab=totot
$ echo $ab
$ echo ${ab}ababab
```

```
$ echo $abababab
$ y=p ; ${y}wd
```

4) Substitution de commandes

```
$ echo `ls -l`
$ echo $(ls ~/Textes)
```

5) Substitution arithmetique

```
$␣echo␣p$((␣3␣+␣4␣))q
```

6) Substitutions de chemins

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

7) Substitutions et execution de commandes

```
$ x=p; y=$(${x}wd); echo cwd=$y
$ x=p; y=`${x}wd`; echo cwd=$y
```

6.1 Quotations

1) Déduire le rôle des quotes et doubles quotes

```
$ x=p; y='${x}wd'; echo $y
$ echo "*"*
$ echo "* $(pwd)"
```

2) Expliquer le comportement des commandes

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

6.1 Exercice Faire afficher les chaînes suivantes

`***␣BONJOUR␣***`

`␣-␣␣␣Le␣caractere␣'*'`

`/Fichier␣d'entree\`

`*␣Mon␣Jan␣20␣17:59:48␣EET␣1992␣*` (la date doit être le résultat d'une exécution de `date`)

`$$␣Mon␣Jan␣20␣18:07:15␣EET␣1992␣$$` (idem)

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

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

7 Les script shells et mécanismes de substitution

Un fichier script contient une suite de commandes shell. Pour exécuter le script deux méthodes sont possibles. Soit vous tapez, le nom du shell suivi du nom du script à interpréter :

```
$ bash mon-script
```

soit vous tapez directement le nom du script :

```
$ mon-script
```

Dans ce dernier cas, le fichier script doit posséder la permission en exécution et pour garantir son exécution par un shell de type `sh`, la première ligne de ce fichier doit indiquer le chemin absolu du shell qui sera utilisé. La syntaxe de cette ligne est la suivante :

```
#!/bin/sh
```

Si cette ligne est omise, c'est le shell de type `bash` qui est utilisé par défaut. Dans tous les cas, prenez la bonne habitude d'insérer systématiquement cette ligne particulière dans vos script shells.

7.1 Exercice Créez un fichier exécutable `cmd.sh` contenant :

```
#!/bin/sh
ps -f
```

Dans votre fenêtre `terminal`, lancez la commande `cmd.sh`. Si tout va bien, vous devriez voir s'afficher la liste de vos processus et en particulier un processus (à peu de chose près) de la forme `/bin/sh cmd.sh`. Modifier la ligne `#!/bin/sh` afin d'utiliser un autre shell que `sh` et réexaminez le résultat produit par l'exécution de la commande `cmd.sh`.

7.1 Les variables prédéfinies

Le shell offre des variables prédéfinies facilitant la programmation sous le shell. Vous avez déjà pu observer le rôle des variables `?` ou `$`. Cherchez dans le manuel de `bash` le rôle des variables `#`, `*`, `@`, `0`, `1`, `2`, `3`, etc.

7.2 Exercice Insérez dans l'un de vos script shell, une commande permettant d'afficher le nom absolu du script lancé (chemin depuis la racine + nom de la commande). Pour en vérifier le bon fonctionnement, renommez votre fichier en `autre_nom.sh` et exécutez le.

7.3 Exercice Insérez une commande affichant le nombre de paramètres du fichier de commandes de la manière suivante :

$$nomvar = 'nb\_param'$$

où *nomvar* est le nom de la variable contenant cette information et *nb_param* son contenu. Vérifiez que cela fonctionne en lançant le fichier de commandes avec plusieurs paramètres.

7.4 Exercice De la même façon, insérez une commande affichant la liste des paramètres du fichier de commandes. Trouvez une autre façon d'écrire la commande précédente pour obtenir le même résultat.

7.2 Substitution de commandes et de variables

7.5 Exercice Examinez les scripts qui suivent,

petit-script-4

```
#!/bin/sh
date
echo Il est $4
```

petit-script-5

```
#!/bin/sh
set `date`
echo Il est $4
```

et exécutez les commandes :

`$ petit-script-4 donald riri fifi loulou picsou`

`$ petit-script-5 donald riri fifi loulou picsou`

afin d'interprétez le rôle de la commande : `set` $\prec$ *commande* $\succ$.

7.6 Exercice Écrire un script qui affiche l'heure sous la forme :

```
Il est 17:45:26.
```

Consultez le manuel de `bash` afin d'examiner le rôle de la variable IFS.

7.7 Exercice En modifiant la variable `IFS`, écrire un script qui affiche :

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

7.8 Exercice Créez un répertoire ∼`/bin`. Désormais tous vos scripts seront créer dans le répertoire `~/bin`.

7.9 Exercice Modifiez votre environnement de façon à ce que les scripts inclus dans le répertoire ∼`/bin` soit utilisable depuis n'importe quel endroit sans en spécifier le chemin d'accès. Pour ce faire vous modifierez, dans vos fichiers de configuration, la variable `PATH` de telle sorte que celle-ci prenne en compte votre répertoire ∼`/bin`[19]. Attention : la variable `PATH` prend peut-être déjà en compte votre répertoire ∼`/bin`[20]. Pour le vérifier faites `echo $PATH`.

7.10 Exercice Rendre la modification de la variable `PATH` valide d'une connexion sur l'autre.

7.11 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` suivi d'une ligne vide.

7.12 Exercice Écrire, sans utiliser de structure de contrôle, les commandes suivantes :

1) `nf` : affiche le nombre de fichiers ne commençant pas par un point, du répertoire courant[21] ;
2) `ra` : affiche `oui` si le répertoire courant est le répertoire d'accueil et `non` sinon[22] ;
3) `prc` : affiche la profondeur du répertoire courant[23] ;

7.13 Exercice Écrire le script `ifdef` tel que l'exécution de `ifdef` ≺*nom*≻`.h` place en tête de ≺*nom*≻`.h` les lignes

```
#ifndef ≺NOM≻_H
#define ≺NOM≻_H
```

et la ligne

```
#endif /* ≺NOM≻_H */
```

à la fin. On pourra utiliser la commande `tr(1)` pour les conversions de minuscules en majuscules.

[19] Vous rajouterez a l'endroit aproprié la ligne : `PATH=$HOME/bin:$PATH`
[20] Dans ce cas vous n'avez rien à modifier.
[21] Vous utiliserez les commandes `echo`, `wc` et `pwd`.
[22] Vous utiliserez les commandes `test`, `pwd`, `echo` et les opérateurs `&&` et `||`.
[23] Vous utiliserez les commandes `set`, `pwd`, `shift`, `echo` ainsi que la variable `IFS`.