Introduction au shell Bash

Le shell

Un interpréteur de commandes (le "shell", la coquille qui entoure le "noyau" du système) est un programme qui sert d'intermédiaire entre l'utilisateur et le système d'exploitation.
Sa tâche essentielle est l'exécution de programmes.
Pour cela, il effectue (en boucle infinie) :

- *la lecture d'une ligne*
- *sa compréhension comme une demande d'exécution d'un programme avec d'éventuels paramètres.*
- *le lancement de ce programme avec passage des paramètres*
- *d'éventuelles redirections d'entrées-sorties*
- *les exécutions de scripts (fichiers de commandes)*

Démarrage du shell

- Lors de la création de son compte, un utilisateur est associé à un type de **shell**
- Lire le fichier **/etc/passwd** : le dernier champ contient le nom du fichier exécutable (shell par défaut) **/bin/bash**
- Le shell associé est ainsi lancé automatiquement dès la saisie du login utilisateur.
- Il poursuit sa configuration en exécutant des scripts globaux à tous les utilisateurs et des scripts liés au compte et qui permettent une personnalisation.
- Enfin, il affiche le prompt et se met en attente de la lecture d'une commande.
- Jusqu'à la commande **`exit`**, pour quitter le shell (ce qui équivaut à se déconnecter (logout))

Les scripts de connexion

1. d'abord le script **`/etc/profile`** *communs à tous les users y compris* `root`. On y trouve notamment la définition de **`umask`**
2. celui-ci cherche à exécuter tous les scripts `/etc/profile.d/*.sh` (parcourir `alias.sh` et `numlock.sh`)
3. puis il y a exécution de **`$HOME/.bash_profile`** (la variable $HOME contient le chemin vers le répertoire personnel). Il s'agit ainsi d'un fichier de démarrage personnel et paramétrable.
4. A son tour il exécute `$HOME/.bashrc` dans lequel il est recommandé de placer toutes les fonctions ou alias personnels (car `.bashrc` est exécuté dans tout shell)
5. Enfin le précédent script exécute `/etc/bashrc`, dans lequel on place les alias globaux et la définition symbolique du prompt `$PS1`
6. Puis le prompt utilisateur s'affiche et le shell attend une commande ...

Personnalisation des commandes bash

- **`/etc/bashrc`** étant le dernier script d'initialisation du shell bash, **`root`** peut y définir des alias globaux pour tous les utilisateurs
- Exemple avec **`vi`** (pour utiliser l'éditeur de Midnigth Commander lancer **`mc`**)

```
# vi /etc/bashrc
alias lll="ll | less"
alias x="startx"
alias m="mc"
:wq  (pour écrire dans le fichier et quitter vi)
```

- Puis se reloguer (`exit`) pour que ces nouvelles commandes soient prises en compte par le nouveau shell.

Personnalisation du login utilisateur

Chaque utilisateur peut ajouter des commandes shell au fichier de profil personnel,
`~/.bash_profile`
Par exemple, voici ce que j'ai mis à la fin de ce fichier :

```
clear
salut="Bonjour $USER !\nJe te souhaite bon courage ...\n\
# le dernier \ pour pouvoir continuer la commande sur la ligne suivante
# $( ..) pour obtenir le résultat de l'exécution de la commande incluse
Nous sommes le $(date) "
# -e option indispensable pour interpréter les \n
echo -e $salut
```

Les variables d'environnement système

- La liste en est accessible par la commande : **env**
- La commande **`echo`** permet d'obtenir la valeur d'une telle variable.
  Par exemple : **`echo`** `$PATH`, **`echo`** `$USER`
- Ajout d'un nouveau chemin : attention à ne pas écraser la liste des chemins existants (`PATH` en majuscules !)
  - **`PATH="$PATH:/home/jean/bin"`**
    pour ajouter le chemin vers les exécutables du rép. personnel (Attention ! pas d'espace autour du symbole =)
  - **`PATH="$PATH :./"`**
    pour toujours ajouter le répertoire courant (non présent par défaut)
- La variable **`$HOME`** contient le chemin du rép. personnel.
  La commande `cd $HOME` est abrégée en `cd`
- La variable **`$USER`** contient le nom de l'utilisateur
- **`$SHLVL`** donne le niveau du shell courant

Facilités de saisie des commandes

Comme les commandes Unix sont souvent longues à saisir, diverses facilités sont offertes :

Historique des commandes
Cette liste numérotée est accessible en tapant **`history | less`** Pour relancer la commande numéro **n**, saisir (sans espace) **!n**
On peut aussi parcourir les précédentes lignes de commandes avec les flèches (comme *doskey*) et les éditer. Ceci permet très facilement de reprendre une précédente commande pour l'éditer et la modifier.

Le clic-droit
Dans un terminal console, sélectionner un texte quelconque. Un clic-droit recopie ce texte sur la ligne de commande, même dans une autre console.

L'opérateur tilde

Le caractère tilde ~ (alt 126) **seul** renvoie au **rép. personnel** de l'utilisateur actuel.
Si l'user actif est toto, chaque occurrence du caractère ~ est remplacé par le chemin `/home/toto`

Le tilde ~ suivi d'un nom d'user, par ex jean, renvoie au **rép. personnel** de jean, c-à-d `/home/jean`
Ainsi par cette commande **`cd ~stagiaire3`** tente en vain d'aller dans le rép. **`/home/stagiaire3`**

 **TP 1**

1. Personnaliser le script **`.bash_logout`** situé dans votre répertoire personnel pour que le contenu du cache de Netscape soit effacé au moment de votre déconnexion
   Puis comme `root`, compléter le script "modèle" `/etc/skel/.bash_logout`, afin que ce "nettoyage" soit effectué pour tout nouvel utilisateur.

2. Expérimenter les "tab" et "clic-droit".
   Saisir `echo -n "Bonjour $USER ! Nous sommes le "; date`
   Puis utiliser le "clic-droit" pour exécuter cette commande dans une autre console.

3. [root@pc5 /root] **cat > essai.txt** *<Entr> envoie la saisie dans le fichier*
   Ceci est un essai tout simple ! *<Entr>*
   mais tout-à-fait intéressant. *<Entr>*
   <CTRL-D> caractère fin de fichier sous Linux
   [root@pc5 /root] **cat essai.txt** *envoie le contenu à la console*
   [root@pc5 /root] cp essai.txt ~jean copie le fichier dans /home/jean

---

Compléter une commande

Lorsqu'on tape une commande en ligne la touche **`TAB`**, l'interpréteur cherche à compléter le nom du fichier.
`home/toto ]$ less /etc/fs` **`TAB`**

S'il y a plusieurs propositions, il y a attente d'un complément d'info de la part de l'utilisateur (avec un *"tut"*).
Un autre **TAB** et l'interpréteur affiche toutes les possibilités ou en indique le nombre, s'il y en a beaucoup !

 **TP 2**

$ **cd /etc** *<TAB>*
there are 111 possibilities. Do you really wish to see them all ? (y or n)
$ cd /etc/**s** *<TAB>*
security services smb.conf syslog.conf etc ..
$ cd /etc/s**y**s*<TAB>* *# on tape y, le système complète s et ... attend*
sysconfig syslog.conf syslog.conf.inn
$ cd /etc/sys**c***<TAB>* *# on ajoute c, le système complète aussitôt à* ***sysconfig***
$ cd /etc/sysconfig/ *<Entr>*

Exo : poursuivre ainsi jusqu'à afficher le contenu du fichier de configuration de l'interface Ethernet /etc/sysconfig/network-scripts/ifcfg-eth0

Désigner un ensemble de fichiers

- Travailler avec le shell nécessite souvent de manipuler des ensembles de fichiers. L'utilisation de caractères spéciaux (appelés aussi **méta-caractères**) dans les noms de fichiers, permet de générer des modèles pour désigner ces ensembles.
  Il existe quatre constructeurs de modèles ***, ? , [] et ^**.

Modèle	Signification
*	remplace une chaine de longueur qcq, même vide
?	remplace un seul caractère qcq
[ ]	un caractère qcq de la liste ou de l'intervalle
[^]	n'importe quel caractère **sauf** ceux de la liste

  Attention ! en raison de certaines ressemblances, ne pas confondre ces constructeurs d'ensembles de fichiers avec les expressions rationnelles (utilisées par exemple dans **grep** ou **sed**)

- Un modèle de la forme **X*Y** où X et Y sont 2 chaînes quelconques, éventuellement vides, désigne l'ensemble des noms de fichiers de la forme XZY où Z est une chaîne quelconque elle aussi éventuellement vide.
- Un modèle de la forme **X?Y** désigne l'ensemble des noms de fichiers de la forme XuY, où u est un seul caractère
- Exemples

```
ll /*/*.d        tous les fichiers d'un rép de / qui se terminent par .d
ll -d /home/*    tous les sous-répertoires de /home
rm *     attention ! commande dangereuse, supprime tout le rép courant !
cp /lib/modules/*/*/*.?  /home/toto   toto copie tous les pilotes dans son r
cp /home/stage? /root/tmp
```

- Le modèle **[]** permet de sélectionner un élément de la liste ou de l'intervalle spécifié.
  Le séparateur en ligne de commandede étant l'espace, aucun espace ne doit être mis au début ou à la fin entre [ ]
- Plus précisément, un modèle de la forme **X [abc...z]Y** où X et Y peuvent être vides, désigne l'ensemble des noms de fichiers suivants: XaY, XbY ... XzY.
- Dans le cas d'un suite ordonnée de caractères comme abc ...z, on peut utiliser la notation intervalle **a-z**.
- On peut mélanger les deux notations, comme dans [a-z].[0-9], ensemble des fichiers a.0, a.1, .., b.0 b.1 etc ...

- Quelques exemples :
  - **ll a***
  - **ll [a-dA-D]*** liste les fichiers du rép. courant dont le nom commence par a, b, c ou d minuscule ou majuscule (y compris les sous-rép.)
  - **cp ventes1[00-50].xls /home/toto/bilan** copie tous les fichiers ventes100.xls jusqu'à ventes150.xls
  - **lpr ~toto/formation/plan9[345].html** imprime les 3 fichiers plan93.html, plan94.html, ..

 **TP 3**

Etudier et commenter les commandes suivantes, en étant connecté `root`

Commande	*Signification de cette commande ? que remarquez vous ?*
`ll ~/m*`	
cd	
ll *.*	où sont passés les autres fichiers ?
ll *	que viennent ici faire les répertoires ?
ll *i*	
ll [a-n]*	
ll [an]*	quelle différence ?
ll [^an]* | less	
ll *.*htm*	
ll [a-z]*/*.pl	
mkdir ~	

 # lister tous les répertoires dont le nom commence par stage, avec une variable

```
user=stage
echo $user
ll -d home/$user*
```

Les commandes du shell

référence : **`man bash`**

Analyse de la ligne de commande

- Le shell commence par découper la ligne en mots séparés par des blancs.
  Le premier mot attendu est le nom d'une commande. Les mots suivants sont considérés comme des paramètres dont la "compréhension" incombe à la commande (ces paramètres ont-ils pour la commande la signification d'options, de noms de fichiers, etc ...). **Donc la syntaxe à appliquer aux paramètres dépend de la commande.**

- Voici un exemple : supposons les comptes `stagex`, x=1..9 déjà crées.
  `grep -n stage. /etc/passwd` La commande **`grep`** attend des options précédées de -, puis un modèle (expression rationnelle) des chaines à chercher, et enfin un ensemble de fichiers où elle doit chercher.

 **TP 4**

- `grep -n sta /etc/passwd` ---> *recherche dans le fichier `/etc/passwd` la sous-chaine **sta**, en indiquant les N° de lignes (option -n)*
- `grep -nw sta /etc/passwd` ---> *recherche ... (l'option -w impose la recherche d'un mot entier, et pas d'une sous-chaine*
- `grep -nw stage. /etc/passwd` ---> recherche ...
- `grep -nw stage? /etc/passwd` ---> quelle signification pour grep du ?
- `grep -nw stage? /etc/*` --->
- `grep -n ftp* /etc/rc.d/init.d/* -->`

Valeur de retour d'une commande

- Chaque commande transmet au programme appelant un code, appelée valeur de retour (*exit status*) qui stipule la manière dont son exécution s'est déroulée.
- Par convention du shell BASH, **la valeur de retour est toujours 0 si la commande s'est déroulée correctement**, sans erreur (attention, c'est l'inverse du langage C !)
- Une valeur de retour différente de 0 signale donc une erreur, qui peut être éventuellement analysée selon cette valeur.
- Un variable système spéciale **$?** contient toujours la valeur de retour de la précédente commande. On peut afficher cette valeur avec la commande **echo**
  Exemples :

```
[toto@p00]$ ll ~
[toto@p00]$ echo $?              --> 0
[toto@p00]$ ll /root
[toto@p00]$ echo $?              --> 1, si toto n'est pas root !
```

Enchainement des commandes

- Habituellement, une ligne de commande saisie au prompt de la console ou bien écrite dans un script est une phrase composée de mots séparés par des espaces (ou des tabulations); le premier mot est considéré comme le nom d'une commande et le shell cherche à l'exécuter; les mots suivants sont des options ou paramètres de cette commande.

- Pour inhiber cette interprétation des espaces, il faut entourer le groupe de mots de quotes ou de guillemets , ce groupe sera alors interprété comme un seul paramètre.
  Exemple : recherche de la chaine *jules toto* (qui constitue un seul paramètre) sur les lignes de /etc/passwd (l'option -i pour s'affranchir de la casse)

```
grep -i "jules toto" /etc/passwd
```

- En général, on place une commande par ligne que ce soit en ligne de commande ou dans un script.
  Le point-virgule **;** a le rôle de séparateur de séquence **inconditionnel**.
  Il permet ainsi d'écrire une séquence de plusieurs commandes sur une même ligne.
  Toutes les commandes sont inconditionnellement exécutées (même si l'une d'entre elle provoque une erreur), et leur résultats respectifs sont envoyés sur la sortie standard, séparés par un retour à la ligne "\n".
  On peut connaitre la valeur de retour de chacune en interrogeant la variable **$?**

TP 5

```
Si toto6 n'est pas un utilisateur valide ?
[root@p00]$ grep toto6 /etc/passwd ;  echo $?
le groupe root existe déjà, il ne peut pas être recréé,
prévoir les codes de retour
[root@p00] #  $ who am i; echo $?; groupadd root; echo $?; date; echo $?
```

Enchainement conditionnels des commandes

- Les séparateurs **&&** et **||** sur la ligne de commande sont des séparateurs qui jouent les rôles d'opérateurs **conditionnels**, en ce sens que la 2ème commande sera exécutée en fonction du code de retour de la 1ère commande.

- Dans `commande1 && commande2`, commande2 ne sera exécutée que si le code de retour de commande1 est 0 (exécution correcte)
  Dans `commande1 || commande2`, commande2 ne sera exécutée que si le code de retour de commande1 est différent de 0 (exécution erronnée)

- Exemples : trouver leur signification

```
cd ~/tmp || mkdir $HOME/tmp
extrait de /etc/rc.d/inet.d/inetd
[ -f /usr/sbin/inetd ] || exit 0
```

Redirections des entrées-sorties

Toutes les commandes (du noyau, du shell et créées par le programmeur) sont dotées par le système de **3 canaux de communication** :

- entrée standard (stdin=standard input) pour lire des données,
- la sortie standard (`stdout`) pour envoyer des résultats
- et la sortie des erreurs (`stderr`).
- Par défaut les canaux d'entrées et de sorties communiquent avec le clavier et l'écran : les commandes et les programmes qui ont besoin de données les attendent en provenance du clavier et expédient leurs résultats pour affichage sur le moniteur.
- Il est possible de les détourner pour les rediriger vers des fichiers ou même vers les entrées-sorties d'autres commandes.
  Les symboles utilisées sont :
  - `<` redirection de l'entrée standard à partir d'un fichier (et non depuis le clavier)
  - `>` redirection de la sortie standard en direction d'un fichier (et non vers l'écran clavier) attention ! le fichier est créé .. et écrase sans préavis le fichier existant portant le même nom.
  - `>>` redirection de la sortie standard **à la fin** du fichier s'il existe déjà
  - `|` (= alt 124) enchainement de commandes (appelé aussi **tube** ou **pipe**)
    la sortie de la commande gauche est envoyée en entrée de la commande droite
    Fréquemment utilisé avec **`less`** (ou more) pour examiner l'affichage sur le moniteur.
    La valeur de retour est celle de la dernière commande.

- Tester
  `ll --help | less`

TP 6

Etudier et commenter les commandes suivantes

- Lorsqu'une commande attend une entrée clavier , taper quelques lignes (du texte qcq) puis terminer par `Ctrl-d` (symbole `EOF`=end-of-file) pour sauvegarder.
  (une liste de symboles ligne de commandes est obtenue par *`stty -a`***)**
- On repasse en mode commande ... et on envoie la commande suivante.
- `lpr` *est la commande d'impression sur la file d'attente par défaut.*
- `wc` (=word count) *compte le nombre de lignes, de mots et de caractères du fichier en entrée (suivant les options `-l, -w, -c`).*

1. Exemples :

```
cd
cat > essai.txt
cat  essai.txt
sort  < essai.txt
cat  >> essai.txt
sort  < essai.txt
sort  < essai.txt  > essai-tri.txt
cat essai-tri.txt
cat  essai.txt  essai-tri.txt
```

2. Quel est l'effet de la commande suivante ? Vérifiez
(`essai.txt` est le fichier créé précédemment)

```
wc -w  < essai.txt > mots.txt
Que se passe t-il si on enlève l'option -w ?
```

3. Pour obtenir le même affichage final, remplacez la séquence suivante par une seule commande

```
cd /etc
ll > /tmp/liste.txt
cat /tmp/liste.txt
wc -l < /tmp/liste.txt
```

4.
```
ll
ll /etc | less
ll  |  sort
ll  |  wc -l
```

5.
```
who
who |  sort
cat | sort > essai-pipe.txt
Pouvez-vous prévoir la différence entre :
cat  essai.txt | lpr
cat  essai.txt > lpr
```

6. Enregistrer dans un même fichier **truc.txt**, la liste des utilisateurs actuellement connectés, la date du jour, le nom de l'utilisateur actif et le rép. personnel trié.

Substitution de commande

- Ce procédé permet de substituer au texte d'une commande le résultat de son exécution qui est envoyé sur la sortie standard
La commande simple ou complexe (avec redirections, tubes) doit être entourée de l''opérateur antiquote ` **`Alt-Gr7`** ou être placée dans un parenthèsage précédé de **`$(...)`**. D'une manière générale, il est recommandé d'entourer l'expression de " "
- Exemple :

```
echo "`whoami`, nous sommes le `date` "
attention, pas d'espace entre $ et (
echo "$(whoami), nous sommes le $(date) "
```

 **TP 7**

- Comparer :

```
pwd
echo pwd
echo `pwd`
echo "Il y a `ls | wc -l ` fichiers dans `pwd` "
```

- tr 'A-Z' 'a-z' traduit chaque caractère majuscule reçu sur son entrée en l'équivalent minuscule . Que réalise alors cette commande ?

```
echo -n "Votre mot de passe ?
read mdp
mdp = $(echo $mdp | tr 'A-Z' 'a-z')
```

- Les substitutions de commande peuvent être imbriquées.
Attention à bien placer les " ". Exemple :

```
echo "Nombre de fichiers du répertoire personnel : $( ls -l $( pwd ))" | les
```

- Si on connait `grep` et `cut`, quelle est la signification de :

```
nom=toto
numero=$(cat /etc/passwd | grep -wi "^$nom" | cut -d: -f3)
```
