Thème 4
Le shell : les processus, les variables et leur portée

Références

— Polycopié UNIX : chapitres 11, 12, ...

— Transparents UNIX : sections 16 à 20

Créer un répertoire `te4` et y copier les fichiers fournis dans le répertoire `te4` du compte de l'UE.

Ex. 1 : Variable `PATH` A 20 min.

L'objectif est de tester le rôle de la variable d'environnement `PATH` sur le lancement des commandes système et utilisateur. Pour éviter de perturber la session avec ces tests, on lancera tout d'abord en arrière plan un terminal d'une couleur particulière, par exemple via la commande `xterm -bg Skyblue &` Toutes les commandes de cet exercice seront ensuite lancées à partir de ce terminal ou de l'un de ses fils.

1. Le shell mémorise les chemins d'accès aux commandes dans une table afin d'accélérer leur lancement : afficher cette table avec la commande interne `hash`. Lancer quelques commandes (`ls`, `date`, puis `ls`, ...) en surveillant l'évolution de la table avec `hash`.
   Puis réinitialiser cette table avec `hash -r` et vérifier qu'elle est vide (on pourra comparer avec la table de la session d'un autre terminal). Puis identifier quelques commandes avec la primitive `type` : `type date`, puis de même pour `xterm`, `xclock`, `cd`, `id`. Noter les résultats.
2. Afficher la liste des chemins de recherche des exécutables par `echo $PATH`. Comment afficher un chemin par ligne ?
   Puis réduire la liste des chemins de recherche des exécutables par `PATH="/bin"`
   Vérifier la nouvelle liste. Enfin lancer successivement les commandes `ls`, `xterm`, puis `/usr/bin/xterm`. Expliquer leur effet et en particulier les éventuels messages d'erreur.
3. Changer de répertoire de travail par `cd /usr/bin` Lancer `xterm`, puis `./xterm`. Expliquer.
4. Compléter la liste des chemins de recherche par `PATH="${PATH}:/usr/bin"`
   Lancer `xterm` et expliquer. Dans le dernier terminal lancé, afficher la liste des chemins de recherche par `echo $PATH`. Expliquer.
5. Ouvrir un terminal dit de « login », c'est-à-dire qui exécute les fichiers de démarrage (notamment `~/.bash_profile`)
   `xterm -ls` et remarquer l'affichage du mot du jour.
   Dans ce nouveau terminal, afficher la liste des chemins. Expliquer.
6. Copier le fichier source C `carre+invite.c` (déjà utilisé) dans votre répertoire `te4`. Quels droits avez-vous sur ce fichier ?
   Lancer `alias gcc-mni-c89` pour repérer les options de compilation imposées par ce raccourci. Compiler ce fichier source via `gcc-mni-c89 carre+invite.c` Qui peut exécuter le fichier `a.out` produit ? Avez vous eu besoin d'utiliser `chmod` pour le rendre exécutable ? Essayer de le lancer avec `a.out`, puis avec `./a.out` ; expliquer.
   Ajouter le répertoire courant à la fin de la variable `PATH`. Peut-on lancer l'exécutable `a.out` avec seulement son nom sans préciser de chemin ? Même question après avoir renommé l'exécutable en `carre-c.x`
7. B Renommer provisoirement le fichier `a.out` en `ls`. Avec ce `PATH`, comment doit-on procéder pour le lancer ? Détruire votre fichier `ls`.

Fermer le terminal « bleu ciel » afin de retrouver une configuration plus saine.

Ex. 2 : Processus AB 10 min.

Se connecter sur le serveur d'applications pour effectuer cet exercice.
Lancer une horloge en arrière plan (`xclock &`), puis un terminal (`xterm &`). Dans ce terminal, lancer une autre horloge en arrière plan, puis un autre terminal `xterm` et enfin un terminal `uxterm`.
Exécuter la commande `ps -f -U` *votre_id* où *votre_id* est donné par la commande `id`[1]. Représenter la généalogie des processus indiqués (utiliser les PPID/PID **p**arent-/**p**rocess **id**entifier). Vérifier la hiérarchie des processus lancés avec l'option `--forest` sous LINUX. Comment afficher seulement les processus associés aux terminaux `xterm` ?
Interrompre les processus liés aux horloges avec la commande `kill`.

Ex. 3 : Calcul et contrôle de processus AB 20 min.

1. Copier les fichiers sources `infini-while.c` et `infini-while.f90` en langage C et fortran90 respectivement dans votre répertoire `te4`. Expliquer pourquoi ils produisent des boucles sans fin. Compiler un de ces fichiers pour obtenir un exécutable nommé `infini-while.x` :
   `gcc-mni-c89 infini-while.c -o infini-while-c.x`
   ou
   `gfortran03-mni infini-while.f90 -o infini-while-f.x`
   Lancer cet exécutable et attendre... Afficher les processus personnels actifs avec `jobs -l`. Surveiller la charge de la machine avec la commande `top`. On repère facilement dans `top` les processus d'un utilisateur en saisissant `u` suivi de son identifiant ou de son numéro d'utilisateur (UID) fourni par la commande `id -u`[2]. Il faudra impérativement interrompre ce processus (avec par exemple `kill %n` où `n` est le numéro du job) !
2. B Faire une copie de ces fichiers nommée `infini-while+.f90` (`infini-while+.c`) et changer la multiplication en addition dans la boucle ; compiler la copie en nommant l'exécutable `infini-while+f.x` (`infini-while+c.x`). Lancer successivement ces deux exécutables et attendre : expliquer leur comportement (penser à la représentation des entiers).

Ne pas oublier d'arrêter tous les processus `infini-while.x` lancés avant de se déconnecter.

Compléments du TE 4

Traiter les exercices 4 et 5 à la fois sur le serveur et sur la machine locale.

Ex. 4 : Premiers shell-scripts AB 10 min.

1. Un fichier de commande (ou shell-script) est un fichier texte contenant des commandes destinées au shell. Un shell-script a déjà été utilisé dans la question 3 de l'exercice 1 du TE 3, p. 13.
   - Créer un fichier `premier.sh` contenant le texte ci-contre
   - Exécuter le script en saisissant : `bash premier.sh`
   - Lancer la commande `./premier.sh` Permet-elle l'exécution du script ? Ajuster les permissions du fichier pour autoriser son exécution via cette commande.

   premier.sh
   ```
   date
   hostname
   whoami
   cat /home/lefrere/M1/Config/motd
   ```

   - Lancer la commande `premier.sh` Permet-elle l'exécution du script ? Afficher la variable `PATH` par la commande `echo $PATH`.
   - Ajouter le répertoire courant dans la variable `PATH`, et effectuer de nouveau la commande `premier.sh` Conclure.
2. Les shell-scripts doivent être écrits en suivant une syntaxe particulière dépendant du type de shell utilisé. Pour connaitre le nom du shell interactif utilisé dans votre terminal, saisir `echo $SHELL`, et pour connaître le type de terminal utilisé, `echo $TERM`. Ces variables sont des variables d'environnement standard de même que `PATH`, `USER` et `HOME`.

1. Attention `ps -U` *votre_login* serait compris comme un `id` si le login est numérique, ce qui est le cas des étudiants.
2. L'argument `nom_de_login` de l'option `u` n'est pas utilisable avec les logins numériques mis en place à l'UPMC, qui seraient considérés comme des UID.

Afin d'assurer la portabilité du script, il est conseillé d'indiquer dans le fichier lui-même le shell [3] qui doit interpréter le script, quel que soit le shell père (depuis lequel le script est lancé) : `#!/bin/bash` placé sur la première ligne du shell-script. Sur une ligne, ce qui est écrit à la suite d'un `#` est considéré comme un commentaire. En particulier toute ligne commençant par un `#` est un commentaire... sauf si elle suit la syntaxe précédente (`#!`).
Créer le fichier `deux.sh` ci-contre.
Exécuter ce shell-script. Commenter.

deux.sh
```
#!/bin/bash
# Ceci est un commentaire
pwd
ls -l
# ceci est un autre commentaire
echo "---------------------------------"
cd
pwd
ls | grep 'mni' # encore un commentaire
echo "Fin du script"
```

3. Écrire un script `poids.sh` qui affiche sur la sortie standard : le nom de l'utilisateur, le chemin absolu de son répertoire d'accueil et l'espace occupé par l'ensemble des fichiers sous son répertoire d'accueil (utiliser la commande `du`).

Ex. 5 : Variables du shell **AB** **30 min.**

1. Créer un fichier `trois.sh` contenant

```
#!/bin/bash
# utilisation de la fonction echo
echo "La date du jour est: " date
echo "La date du jour est: " ; date
echo -n "La date du jour est: "; date
```

Exécuter ce fichier de commandes et commenter.

2. Créer un fichier `quatre.sh` contenant

début de quatre.sh
```
1 var1=un
2 var12=douze
3 echo ligne1: $var1
4 echo ligne2: $var1plusdutexte
5 echo ligne3: ${var1}
6 echo ligne4: ${var1plusdutexte}
7 echo ligne5: ${var1}plusdutexte
```

fin de quatre.sh
```
echo ligne6: ${var12}                                  8
echo ligne7: ${var1}2                                  9
echo ligne8: $var1et$var12                             10
echo ligne9: ${var1}et${var12}                         11
echo ligne10: ${var1} et ${var12}                      12
echo ligne11: ${var1}     et     ${var12}              13
echo ligne12: "${var1}     et     ${var12}"           14
```

Exécuter le shell-script et expliquer en détail les affichages.

3. Créer un fichier `cinq.sh` contenant

début de cinq.sh
```
1 #!/bin/bash
2 # La commande date +%X affiche
3 #  l'heure sous la forme H:M:S
4 var1=xxxxxxxx
5 date +%X
6 var1=date +%X
```

fin de cinq.sh
```
echo 1 ${var1}                 7
var1="date +%X"                8
echo 2 ${var1}                 9
var2=$(date +%X)               10
echo 3 ${var2}                 11
var2=$(date +%D)               12
echo 4 ${var2}                 13
```

Expliquer le comportement du shell-script ligne par ligne.

4. Écrire un script `espace.sh` pour obtenir un affichage du type :
`Mon nom de connexion est : numero_etudiant`
`Mon répertoire d'accueil est : /home/numero_etudiant`
`Espace utilisé: XXX  sur 500 Mo`

3. Le bash (Bourne Again Shell) est le shell par défaut de la plupart des distributions Linux, mais aussi celui du terminal de Mac OS X. Mais il existe d'autres types de shell tels que le Bourne Shell (sh), le Korn Shell (ksh), le C Shell (csh), le Tenex C Shell (tcsh) et le Z Shell (zsh).

5. La commande `read` implique une lecture au clavier c'est à dire une saisie par l'utilisateur dans le terminal. Elle permet de récupérer plusieurs variables saisies en une seule ligne. Dans le cas où l'utilisateur saisit plus de mots que demandés, la dernière variable contient les derniers mots de la ligne. Dans le cas où il n'en saisit pas suffisamment, les dernières variables sont vides.

   Créer un fichier `six.sh` contenant

```
#!/bin/bash
echo saisir trois mots
read varA varB varC
echo Affichage des valeurs saisies:
echo ${varA}
echo ${varB}
echo ${varC}
```

   Exécuter le script-shell en saisissant au clavier un, deux, trois et enfin quatre mots. Créer un fichier texte contenant une ligne de trois mots et utiliser une redirection afin d'éviter la saisie interactive.
6. Écrire un script-shell `compte_pixel.sh` qui compte le nombre de pixels à un niveau de gris donné dans un fichier d'image du type pgm en ascii et affiche ce nombre à l'écran. On utilisera le fichier `img.pgm` du TE 3. Dans un premier temps, le chemin du fichier et le niveau de gris recherché seront fixés dans le script. On vérifiera qu'il y a 16 pixels dont le niveau de gris est 200.
   Dans un second temps, ce script invitera l'utilisateur à saisir le chemin du fichier, puis le niveau de gris recherché. Si le script crée des fichiers temporaires, il doit également les effacer.

Ex. 6 : Les paramètres des scripts AB | 15 min.

1. Créer un fichier `huit.sh` contenant

```
#!/bin/bash
echo -n "la procédure $0 "
echo a été appelée avec $# paramètres
echo le premier paramètre est $1
echo le deuxieme paramètre est $2
echo la liste des paramètres est $*
echo le numéro du processus lancé est $$
```

   Exécuter la commande :
   `huit.sh VAR 5 du texte`
   puis la commande :
   `huit.sh VAR 5 "du texte"`
   Expliquer.
2. Modifier le script-shell `compte_pixel.sh` pour qu'il utilise des paramètres fournis sur la ligne de commande dès le lancement du script plutôt qu'il attende leur saisie interactive au clavier.
3. Créer le shell-script `compile.sh` qui compile un programme C source dont le nom (sans le `.c`) est passé en paramètre. On suppose que le fichier source est dans le même répertoire que celui contenant le fichier `compile.sh`. Ce script placera l'exécutable dans le répertoire d'accueil, dans un fichier de même nom mais sans extension. Ce script devra afficher tout d'abord son nom, le paramètre passé, puis la taille du fichier source et le nombre de lignes qu'il contient. Il lancera ensuite la compilation (via `gcc`) et placera l'exécutable comme demandé. Il affichera enfin la taille du fichier exécutable. Utiliser le fichier `carre+invite.c` (*cf.* TE 2) pour tester le script.

Ex. 7 : Combinaisons de commandes et commande `exit` AB | 10 min.

1. Exécuter dans votre terminal les commandes ci-contre :
   Rappeler ce que représente `$?`

```
ls /inexistant
echo $?
cat /home/lefrere/M1/Config/motd
echo $?
```

2. Exécuter ensuite dans votre terminal les commandes :
   Parmi les 4 commandes précédentes pourquoi certaines affichent `Yes` alors que d'autres non ?

```
ls /inexistant && echo 'Yes'
ls /inexistant || echo 'Yes'
cat /home/lefrere/M1/Config/motd && echo 'Yes'
cat /home/lefrere/M1/Config/motd || echo 'Yes'
```

3. En utilisant les combinaisons de commandes `&&` et `||` ainsi que la commande `exit`, modifier `compile.sh` afin que le déplacement dans le répertoire d'accueil ainsi que l'affichage du fichier exécutable ne se fasse que si la compilation a réussi.