


# Enoncé du TP 9 Système

C. Pain-Barre

INFO - IUT Aix-en-Provence

version du 16/12/2012


 Démarrer les PC sous Linux. Les exercices sont à faire via une connexion SSH (normale) sur allegro.

## 1 Environnement des processus

Lorsqu'un processus (et notamment bash) crée un processus fils, ce dernier hérite d'une copie partielle de l'environnement de son père. Dans cette partie, nous limiterons l'étude de cet héritage, entre bash et ses fils, aux éléments suivants :

- le répertoire de travail ;
- le masque de création de fichiers ;
- les fichiers ouverts ;
- les alias, variables, tableaux et fonctions.

Il est légèrement différent selon que le fils est créé pour exécuter un sous-shell ou une commande externe (ce qui comprend les scripts bash). Dans les deux cas, le fils hérite du répertoire de travail, du masque de création de fichiers et des fichiers ouverts. Si c'est un sous-shell (créé avec les parenthèses, la substitution de commandes ou dans un pipe), il hérite en outre d'une copie de tous les alias, variables, tableaux et fonctions. Mais si c'est une commande externe, seules les variables (et les tableaux et fonctions) d'environnement sont héritées. Dans tous les cas, la modification par le fils de son environnement n'a pas d'incidence sur celui du père.

 Techniquement, la création d'un processus fils est réalisée par l'appel de la fonction système **fork()** qui produit un (quasi) clone du processus père. Notamment, la mémoire du père (où sont stockées les variables, la pile d'exécution, etc.) est dupliquée (copiée) pour son fils, comme les (descripteurs de) fichiers ouverts et bien d'autres éléments. C'est à peu près ce qui se produit lorsque bash crée un sous-shell.

Néanmoins, le fils peut ensuite appeler une fonction de la famille **exec()** afin d'exécuter un fichier exécutable. Dans ce cas, une partie de son environnement, notamment la mémoire, est alors réinitialisée (remplacée par l'image du programme chargé). Seules restent les variables (tableaux/fonctions) d'environnement. C'est ce qui se produit lorsque bash crée un fils pour exécuter une commande externe.

### Exercice 1

*Étudier l'héritage de l'environnement d'un processus bash par ses fils.*

#### 1. Mise en place d'un environnement.

Sur un terminal sur allegro, se placer dans `tpunix` et taper les commandes suivantes :

```
$ MAVERNOM=normale  
$ echo "$MAVERNOM"
```

↩ pour créer une variable (normale) **MAVERNOM** contenant "normale" et vérifier son contenu ;

```
$ export MAVARENV=environnement
$ echo "$MAVARENV"
```

⇒ pour créer une variable d'environnement **MAVARENV** contenant "environnement" et vérifier son contenu ;

```
$ function MAFONCNorm { echo "message affiché par MAFONCNorm" ; }
$ MAFONCNorm
```

⇒ pour créer une fonction (normale) **MAFONCNorm** qui affiche un message reconnaissable, et vérifier son exécution ;

```
$ function MAFONCENV { echo "message affiché par MAFONCENV" ; }
$ export -f MAFONCENV
$ MAFONCENV
```

⇒ pour créer une fonction d'environnement **MAFONCENV** qui affiche un message reconnaissable, et vérifier son exécution ;

```
$ umask
```

⇒ pour vérifier le masque de création de fichiers de ce shell

```
$ alias
```

⇒ pour afficher les alias existants sur ce shell

```
$ ls -l /proc/$$/fd
```

⇒ pour afficher la liste des fichiers ouverts par le bash courant (dont le **PID** est obtenu avec **\$\$**)

📁 **/proc** est un répertoire virtuel créé et alimenté par le noyau Linux pour informer sur les processus en cours. Tout processus en activité y est représenté par le répertoire **/proc/pid**, où **pid** est son **PID**.

Ainsi, **/proc/\$\$** est le répertoire informant sur le processus courant (**\$\$**). Le répertoire **/proc/\$\$/fd** (pour *file descriptor*) informe sur ses fichiers ouverts. Il contient autant de liens symboliques que d'entrées-sorties du processus. Ces liens ont un numéro en guise de nom (appelé descripteur de fichier), et pointent vers le fichier ouvert sur cette entrée ou cette sortie. Classiquement, on retrouvera au moins les liens suivants :

- **0** qui représente l'entrée standard ;
- **1** qui représente la sortie standard ;
- **2** qui représente la sortie d'erreur ;
- **255** qui est utilisé de façon interne par bash.

## 2. Vérification de l'environnement hérité par un sous-shell.

Créer un sous-shell qui affiche une partie significative de son environnement en tapant la commande (ici présentée sur deux lignes par manque de place) :

```
$ ( echo "$MAVARNORM" ; echo "$MAVARENV"; MAFONCNorm ; MAFONCENV ;
    umask ; alias; ls -l /proc/$BASHPID/fd )
```

⇒ où le sous-shell est créé par les parenthèses. On peut noter que dans un sous-shell, son **PID** est contenu dans **BASHPID** et non dans **\$\$** qui reste celui du shell parent.

Le résultat affiché devrait montrer que le sous-shell a hérité de tous les alias, variables (etc.), et fichiers ouverts.

### 3. Indépendance de l'environnement du fils.

Le fils a son propre environnement. Ses modifications n'impactent pas l'environnement de son père. Exécuter les deux commandes ci-dessous pour le vérifier :

```
$ ( cd / ; MAVARENV="fils"; unalias ls )
```

↳ crée un sous-shell qui a modifié son environnement (puis s'est terminé)

```
$ pwd ; echo "$MAVARENV" ; alias ls
```

↳ montre que l'environnement du père n'a pas été modifié

### 4. Vérification de l'environnement d'un fils exécutant une commande externe.

Copier (ou créer) le fichier `~cpb/public/unix/verifenv.bash` qui contient le code bash suivant, affichant une partie de son environnement :

```
#!/bin/bash

echo "Verif de l'environnement de ce processus (PID $$)"

echo "Valeur de 'MAVARNORM' : ${MAVARNORM:-<inconnue>}"
echo "Valeur de 'MAVARENV' : ${MAVARENV:-<inconnue>}"
echo "Appel de MAFONCORNORM" ; MAFONCORNORM
echo "Appel de MAFONCENV" ; MAFONCENV
echo -n "Répertoire de travail : " ; pwd
echo -n "Masque (umask) : " ; umask

echo "Liste des alias :"
alias
echo "Fin des alias."

echo "Liste des entrées-sorties :"
ls -l /proc/$$/fd
echo "Fin des entrées-sorties"
```

L'exécuter et remarquer que pour ce fils **MAVARNORM** n'existe pas, ni **MAFONCORNORM**, ni aucun alias. En revanche, **MAVARENV** et **MAFONCENV** existent car elles ont été exportées.

### 5. Variables d'environnement dans un programme (C/C++).

Nous avons pour le moment travaillé sur l'environnement d'un sous-shell et d'un script bash. Les variables d'environnement (ce qui comprend les tableaux et fonctions) sont en réalité communiquées aux processus dans un tableau de chaînes de caractères ayant la forme "*variable=valeur*".

Dans un programme C/C++, ce tableau est contenu dans la variable **environ** (déclarée dans le *header* `unistd.h`). Par exemple, le code C suivant affiche les variables d'environnement qui lui sont communiquées :

```
#include <stdio.h>
#include <unistd.h>

extern char** environ;

int main (int argc, char*argv[])
{
    char **pvar;
    for (pvar=environ; *pvar != NULL; pvar++)
        printf("%s\n", *pvar);
}
```

Exécuter le fichier `~cpb/public/unix/affenv` qui contient le binaire (programme compilé) de ce code. Observer comment apparaissent **MAVARENV** et **MAFONCENV**.

**i** `affenv` est une version "maison" de la commande externe **printenv** qui fait la même chose.

6. *Création de variables d'environnement pour un processus.*

`bash` fournit un mécanisme simple pour créer des variables d'environnement juste pour une commande externe, et inexistantes pour lui-même. Il suffit d'utiliser la syntaxe suivante :

`variable=valeur {variable=valeur} commande`

qu'on avait mentionné lors de la création des variables `bash`.

Taper :

```
$ XXX=abc YYY=def printenv
$ echo XXX=$XXX YYY=$YYY
```

et observer que ces variables ont existé dans l'environnement du processus de **printenv** mais n'existent pas dans le shell courant.

7. Terminons l'exercice par l'exécution de la commande (sur une ligne) :

```
$ ( export -n MAVARENV ; ls -l /proc/$BASHPID/fd ;
  ./verifenv.bash > ficout ) < cigale.txt
```

où l'entrée standard du sous-shell est `cigale.txt` et la sortie de `verifenv.bash` est le fichier **ficout**. Afficher **ficout** et observer :

- que **MAVARENV** n'existait pas dans son environnement car son père (le sous-shell) l'en avait enlevée ;
- que son entrée standard et sa sortie d'erreur ont été héritées du sous-shell ;
- que sa sortie était `ficout`.

## Exercice 2

*Manipuler des variables d'environnement.*

Les variables d'environnement sont un moyen de communiquer des paramètres à certains utilitaires (logiciels), sans les leur spécifier en arguments sur la ligne de commandes. Quelques variables d'environnement, dites **standards**, contiennent des informations dont se servent de nombreux utilitaires. Leur nom est toujours en majuscules, et leur rôle est précis. En particulier, on peut citer :

- **HOME** : qui contient la référence absolue du répertoire d'accueil de l'utilisateur ;
- **LOGNAME** : le nom de l'utilisateur ;
- **PATH** : liste de chemins des exécutables ;
- **LANG** : indique la langue de l'utilisateur et le jeu (codage) de caractères utilisé par défaut. Ces informations constituent **la locale** et sont notamment utilisés pour déterminer dans quelle langue afficher les messages et/ou interfaces, ainsi que pour adapter les réponses attendues à la langue de l'utilisateur.

**i** Plusieurs variables sont prévues pour affiner les opérations dépendant de la locale (comparaison de chaînes, ensembles de caractères, format des nombres, des monnaies, etc.). Leur nom commence par **LC\_**, comme **LC\_ALL**, **LC\_COLLATE**, **LC\_TYPE**, etc. Si elles n'existent pas, **LANG** est utilisée.

1. Sur le terminal, afficher le contenu de la variable **LANG** qui devrait indiquer le français (et le codage UTF8)
2. Exécuter la commande **date** sans argument. Elle devrait afficher la date en français.
3. Taper :  

```
$ LANG=C
```

```
$ date
```

pour passer à la locale standard (Américain, C ANSI), puis exécuter **date** à nouveau. Cette fois, la date est affichée en anglais.
4. Redonner à **LANG** la valeur **fr\_FR.utf8** puis taper :  

```
$ rm -iv cigale.txt
```

Observer que le message demandant confirmation est affiché en français. Répondre **n**
5. Modifier **LANG** juste pour la commande **rm** en tapant :  

```
$ LANG=C rm -iv cigale.txt
```

Le message est affiché en anglais et la réponse est aussi attendue en anglais. Répondre **o** (il ne sera pas supprimé).

**i** **LANG** est largement employée par les utilitaires depuis l'internationalisation des systèmes Unix. Notamment, on change facilement de langue d'affichage des pages de manuel (avec **man**) en la modifiant. Sur allegro, seules les pages en anglais sont installées mais sur les PC, le français et l'anglais sont disponibles. En général, il vaut mieux utiliser les pages en anglais qui sont les plus à jour.

## Exercice 3 (Facultatif)

*Modification des couleurs d'affichage de **ls** par modification de la variable (d'environnement) **LS\_COLORS***

1. Taper **echo "\$LS\_COLORS"**. Cela devrait afficher quelque chose du type :

```
no=00:fi=00:di=01;34:ln=01;36:pi=40;33:so=01;35:bd=40;33;01:cd=40;33;01:or=01;05;37;41:mi=01;05;37;41:ex=01;32:*.cmd=01;32:*.exe=01;32:*.com=01;32:*.bat=01;32:*.sh=01;32:*.csh=01;32:*.tar=01;31:*.tgz=01;31:*.arj=01;31:*.taz=01;31:*.lzh=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.gz=01;31:*.bz2=01;31:*.bz=01;31:*.tz=01;31:*.rpm=01;31:*.cpio=01;31:*.jpg=01;35:*.gif=01;35:*.bmp=01;35:*.xbm=01;35:*.xpm=01;35:*.png=01;35:*.tif=01;35:
```

La variable d'environnement **LS\_COLORS** indique en fait à **ls** les couleurs qu'elle doit utiliser selon le type du fichier à afficher. Ces associations sont séparées par un **:**. La première (**no=00**) indique que pour les fichiers normaux, il n'y a pas d'attribut. L'association **ex=01;32** indique que les fichiers ayant la permission d'exécution doivent être affichés en gras et vert. Pareil pour tous les fichiers se terminant par **.cmd** (association **\*.cmd=01;32**). On peut spécifier 3 éléments d'affichage, séparés par des virgules :

- **l'attribut** : 00=aucun, 01=gras, 04=souligné, 05=clignotant, 07=inverse, 08=caché
- **couleur du texte** : 30=noir, 31=rouge, 32=vert, 33=jaune, 34=bleu, 35=magenta, 36=cyan, 37=blanc
- **couleur de l'arrière-plan** : 40=noir, 41=rouge, 42=vert, 43=jaune, 44=bleu, 45=magenta, 46=cyan, 47=blanc

**i** L'affichage dépend toutefois des capacités du terminal (ou de la fenêtre terminal). Notamment, le mode clignotant est rarement supporté.

2. Sur une Mandriva 2007, le fichier `/etc/DIR_COLORS` contient les couleurs par défaut et un certain nombre d'explications. Sur la Debian, ce fichier n'existe pas mais on obtient à peu près les mêmes renseignements en tapant la commande **`dircolors --print-database`**
3. Modifier **`LS_COLORS`** pour adapter l'affichage à votre goût.

## Exercice 4

*Script bouclant selon les saisies de l'utilisateur. Cet exercice sert aussi de prétexte pour illustrer un usage des variables d'environnement ainsi que la gestion de valeurs par défaut.*

Créer un script **`boucle.bash`** qui demande à l'utilisateur de saisir deux nombres compris entre deux limites *liminf* et *limsup*, ainsi que la somme de ces deux nombres. Le script doit ensuite comparer la somme saisie à celle qu'il aura calculée. Il doit se terminer si elles sont différentes, ou, si elles sont égales, demander à l'utilisateur s'il veut recommencer.

Les limites *liminf* et *limsup* valent par défaut 0 et 99, respectivement. Mais ces valeurs peuvent changer en fonction de l'existence des variables d'environnement **`LESLIMITESINF`** et **`LESLIMITESSUP`** :

- si **`LESLIMITESINF`** est une variable qui existe et contient un nombre, alors c'est *liminf*
- si **`LESLIMITESSUP`** est une variable qui existe et contient un nombre, alors c'est *limsup*

Bien entendu, *liminf* doit être inférieur à *limsup*. Si ce n'est pas le cas, les permuter.

Les variables d'environnement **`LESLIMITESINF`** et **`LESLIMITESSUP`** sont normalement spécifiques à ce script, et n'existent pas. Vous devrez les créer pour tester le script.

Pour vérifier si une chaîne correspond à un nombre, vous pouvez reprendre la fonction **`isnum`** écrite précédemment en TP :

```
function isnum { echo "$1" | grep -q "^[+]\?[0-9]\+$" ; }
```

Lorsqu'une information est demandée à l'utilisateur, il faut boucler tant qu'elle n'est pas correcte, c'est à dire :

- pour un nombre (autre que la somme), tant que ce n'est pas un nombre ou qu'il n'est pas compris entre *liminf* et *limsup* ;
- pour la somme, tant que ce n'est pas un nombre ;
- pour continuer, tant que la saisie n'est pas **`o`**, **`O`**, **`n`**, ou **`N`**.