UNIX - TP3

H. Evenas – X. Roirand

Prénom Nom : Lucien CARRE	**Date : 25/11/24**
Groupe : 1D2	

REMARQUES : comme pour le TP2, à chaque question vous ferez une copie d'écran du contenu de la fenêtre de votre terminal ou un copier/coller des lignes apparaissant dans votre terminal, c'est à dire de la ou les commandes saisies avec les réponses générées par l'interpréteur de commandes pour toutes les questions ou alors vous indiquerez la réponse à la question qui est posée.

Dans ce TP3 nous supposons que vous avez retenu les commandes vues et testées dans les deux premiers TP, qui vous permettront de manipuler les droits, mais aussi de se déplacer dans les répertoires, ainsi que de les créer et de les afficher.

On rappelle que le shell est l'interpréteur de commandes qui permet d'accéder au système d'exploitation. A l'ouverture du terminal, un processus shell est lancé, avec tout un environnement de travail : variables prédéfinies, fichiers de démarrage, facilités diverses (rappel des dernières commandes, création d'alias, etc..).
Il esiste différentes familles de shell :

- Bourne Shell (sh, bash)
- Cshell (csh, tcsh, ..)
- KornShell (ksh)..

Pour ce TP, nous utiliserons le bash – qui est, normalement , le shell lancé à l'ouverture de votre fenêtre de Terminal.

Notions d'entrée standard et sortie standard

Les commandes UNIX utilisent 3 fichiers standard pour leurs entrées-sorties :

- **stdin** : le fichier d'entrée standard, canal 0, associé par défaut au clavier
- **stdout** : le fichier de sortie standard, canal 1. Quand une commande s'exécute, les résultats sont normalement envoyés sur cette sortie (associée par défaut à l'écran).
- **stderr** : le fichier de sortie d'erreur standard, canal 2. Quand une commande qui s'exécute produit une erreur, le message d'erreur est envoyé sur cette sortie (associée par défaut à l'écran).

Variables

En plus des usages que vous avez vus jusqu'ici, le shell propose d'autres fonctionnalités (c'est en fait un véritable langage de programmation, comme nous le verrons par la suite). En particulier, comme en Java, il est possible d'utiliser des *variables*.

Une variable est repérée par un nom appelé *identificateur*, qui peut être n'importe quelle suite de caractères commençant par une lettre ou le caractère « souligné » ('_') et ne contenant que des lettres, des chiffres ou le caractère souligné.

On peut assigner une valeur à la variable var avec l'instruction var=*valeur* (sans espace avant ni après le signe '='). Cette valeur peut être une chaîne de caractères ou un entier par exemple.

On accède à la valeur associée à une variable en faisant précéder son identificateur du symbole $, comme par exemple dans la commande `echo $var` qui affiche la valeur de la variable var.

Exercice 1 – *Variables du shell*

1. Vérifiez que la variable SHELL a déjà une valeur lorsque vous lancez votre terminal. Donnez la commande utilisée. Quelle est cette valeur ?

La commande : « echo $SHELL » renvoie la valeur :
/bin/bash

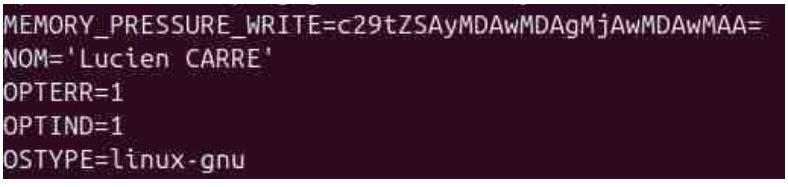
2. Déclarez une variable NOM contenant votre prénom **et** votre nom. Comment résoudre le problème de l'espace entre le prénom et le nom ? Affichez la valeur de NOM pour vérifier que l'affectation est correcte.

Pour résoudre le problème de l'espace, on met des guillemets à la valeur de la variable.
Donc on tape la commande : NOM= "Lucien CARRE"
Et quand on tape : echo $NOM
Cela nous renvoie : Lucien CARRE

3. La commande **set** utilisée sans argument affiche la liste de toutes les variables dont la valeur est instanciée (=initialisée) dans le shell courant. Utilisez la commande `less` pour faire défiler le résultat de la commande `set` page par page, à l'aide d'un **tube**. Vérifiez que la variable précédemment définie apparaît bien dans cette liste.

Note : un **tube** (ou pipe en anglais), symbolisé par le caractère **|** , est un concept essentiel d'UNIX qui permet de faire communiquer 2 processus, en redirigeant la sortie standard (stdout) d'une commande vers l'entrée standard d'une autre commande (stdin). Par exemple, dans la commande suivante : **set** `|` **less** la sortie standard de la commande `set` va servir d'entrée pour la commande `less` qui est une commande permettant de paginer (afficher page par page) la sortie de la 1ère commande.

En tapant set | less
On retrouve bien notre variable dans la liste



4. Écrivez une ligne de commande qui affiche « Bonjour, Machin Chose ! » (remplacez Machin par votre prénom et Chose par votre nom) en utilisant la variable NOM.

5. Depuis le terminal courant, ouvrez un nouveau shell en tapant la commande bash, puis affichez la **valeur** de la variable NOM (quand vous avez terminé, tapez exit pour revenir au shell précédent). Faites de même dans un nouveau terminal lancé depuis la barre d'icônes du bureau. Qu'en déduisez-vous sur la portée des variables du shell ?

Dans le bash :

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ bash
aryouko@aryouko-IdeaPad-1-14ALC7:~$ echo $NOM

aryouko@aryouko-IdeaPad-1-14ALC7:~$ exit
exit
aryouko@aryouko-IdeaPad-1-14ALC7:~$ echo $NOM
Lucien CARRE
```

Dans un nouveau terminal :

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ echo $NOM
```

La portée de la variable est locale à notre terminal et à notre bash, elle ne va pas plus loin.

Dans certains cas, on souhaite que la valeur d'une variable soit accessible également aux processus fils du shell courant, c'est à dire à tous les process qui seraient lancés depuis le terminal ou shell courant (souvenez-vous que le shell est un programme, qui une fois lancé est un process). De telles variables sont appelées variables *d'environnement*.

Exercice 2 – *Variables d'environnement*

1. Affichez la liste des **variables d'environnement** grâce à la commande env. Que remarquez-vous par rapport à la sortie de la commande set ?

On nous affiche les variables d'environnement et il y en a peu par rapport au nombre de variables globales.

2. On transforme une variable `var` en variable d'environnement grâce à la commande `export var`. Transformez la variable NOM en variable d'environnement et répétez la question 5 de l'exercice 1. Qu'en déduisez- vous sur la portée des variables d'environnement ?

Dans le bash :

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ export NOM
aryouko@aryouko-IdeaPad-1-14ALC7:~$ echo $NOM
Lucien CARRE
aryouko@aryouko-IdeaPad-1-14ALC7:~$ bash
aryouko@aryouko-IdeaPad-1-14ALC7:~$ echo $NOM
Lucien CARRE
```

Dans un nouveau terminal :

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ echo $NOM
```

Les variables d'environnement permettent au fils du shell courant de pouvoir utiliser la variable mais ne permet pas au shell indépendant de l'utiliser

3. La commande cd, que vous connaissez bien, permet de changer de répertoire courant. Utilisée sans argument, elle `change` le répertoire courant au répertoire personnel de l'utilisateur. Ce comportement est en fait déterminé par la valeur d'une certaine variable d'environnement.

Repérez cette variable dans la liste des variables d'environnement, et faites en sorte que la commande cd renvoie par défaut vers le répertoire racine. Indiquez dans la réponse comment vous avez fait ? (et n'oubliez pas de donner les copies d'écran).

Remettez ensuite la variable que vous avez modifiée, à sa valeur initiale.

Variable à modifier : `HOME=/home/aryouko`

Variable modifié :

`HOME=/`

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ HOME=/
aryouko@aryouko-IdeaPad-1-14ALC7:/home/aryouko$ cd
aryouko@aryouko-IdeaPad-1-14ALC7:/$ ls
bin                dev   lib64                mnt   sbin                swap.img  var
bin.usr-is-merged  etc   lib.usr-is-merged    proc  sbin.usr-is-merged  sys
boot               home  lost+found          root  snap                tmp
cdrom              lib   media                run   srv                 usr
```

Exercice 3 – *La variable d'environnement* PATH

1. Quelle est la valeur de la variable d'environnement PATH ? A quoi sert le caractere ':' dans cette variable ?

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin
```

Le « : » sert à séparer les différentes valeurs

2. Créez dans votre répertoire personnel (home directory) un répertoire que vous nommerez bin_perso, et créez dans ce répertoire un fichier, nommé fic, contenant la ligne suivante :

```
echo "ceci est le résultat de l'exécution du fichier fic."
```

Détaillez les commandes utilisées (essayez de faire le plus court possible)

mkdir bin_perso
nano bin_perso/fic
(Copie et colle la ligne puis Ctrl+X, O, Entrée)

3. Vérifiez que vous avez les droits d'exécution sur le fichier fic. Modifiez-les si nécessaire. Félicitations, vous venez de créer votre premier exécutable shell. Exécutez-le depuis le répertoire bin_perso en tapant **./fic**, et décrivez le résultat.

```
aryouko@aryouko-IdeaPad-1-14ALC7:~/bin_perso$ chmod 777 fic
aryouko@aryouko-IdeaPad-1-14ALC7:~/bin_perso$ ./fic
ceci est le résultat de l'exécution du fichier fic.
```

4. Essayez maintenant de taper **./fic** depuis un autre répertoire. Cela vous permet-il d'exécuter le fichier fic ? Que faut-il faire pour pouvoir exécuter **fic** depuis un autre répertoire que bin_perso ? Donnez les commandes pour le faire.

```
aryouko@aryouko-IdeaPad-1-14ALC7:~/bin_perso$ cd ..
aryouko@aryouko-IdeaPad-1-14ALC7:~$ ./fic
bash: ./fic: Aucun fichier ou dossier de ce nom
```

Il faut donner le chemin d'accès en plus

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ bin_perso/./fic
ceci est le résultat de l'exécution du fichier fic.
```

5. Ajoutez maintenant à votre variable PATH le chemin <home>/bin_perso (où <home> est le chemin de votre home directory). Vous devrez pour celà affecter à la variable PATH son ancienne valeur à laquelle vous ajouterez la chaîne de caractères:<home>/bin_perso

```
aryouko@aryouko-IdeaPad-1-14ALC7: $ PATH=$PATH:/home/aryouko/bin_perso
aryouko@aryouko-IdeaPad-1-14ALC7: $ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin:/home/aryouko/bin_perso
```

6. Placez-vous maintenant dans votre répertoire `home`, et tapez « fic ». Que se passe-t-il ? Placez vous dans le répertoire /tmp et tapez encore « fic ». Qu'en déduisez-vous sur l'utilité de la variable PATH ?

```
aryouko@aryouko-IdeaPad-1-14ALC7: $ fic
ceci est le résultat de l'exécution du fichier fic.
aryouko@aryouko-IdeaPad-1-14ALC7: $ cd /tmp
aryouko@aryouko-IdeaPad-1-14ALC7:/tmp$ fic
ceci est le résultat de l'exécution du fichier fic.
```

La variable PATH permet de donner les chemins d'accès où des fichiers exécutables sont situés.

7. Comme vous avez dû le remarquer dans l'exercice précédent, les valeurs des variables d'environnement sont accessibles à tous les processus fils du shell courant. Comment faire pour que le chemin <home>/bin_perso/ soit inclus dans la valeur de la variable PATH, dès qu'on lance un nouveau terminal ?

Il faudrait exporter la variable PATH dans le bashrc. Il faut faire « nano ~/.bashrc » et rajouter la ligne « export PATH=$PATH:/home/aryouko/bin_perso

Exercice 4 – *La commande alias*

1. `alias` est une commande qui permet de construire des raccourcis vers des commandes complexes (utilisation en bash : `alias [nom]=[commande]`).

```
aryouko@aryouko-IdeaPad-1-14ALC7: $ alias
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal || echo error)"
"$(history|tail -n1|sed -e '\''s/^\s*[0-9]\+\s*//;s/[;&|]\s*alert$//'\'')"'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l='ls -CF'
alias la='ls -A'
alias ll='ls -alF'
alias ls='ls --color=auto'
```

2. Lancez `alias` et observez. Vous pouvez constater entre autres que dans votre environnement la commande `ls` que vous utilisez couramment est en fait un alias qui est automatiquement créé lors du lancement d'un shell (grâce au fichier de configuration `.bashrc`). Donnez la commande réelle correspondant à `ls`. Lancez la commande de base : `/bin/ls`. Comparez les résultats, qu'apporte l'alias ?

L'alias permet de mettre en couleur les répertoires et fichiers cachés (ça ne se voit pas sur ce screen), la commande réelle est : « ls --color=auto »

```
aryouko@aryouko-IdeaPad-1-14ALC7: $ ls
arendre.css   Bureau      images   Musique   Téléchargements
arendre.html  Documents   Images   Public    Vidéos
bin_perso     fic6.txt    Modèles  snap      wordpress.sql
aryouko@aryouko-IdeaPad-1-14ALC7: $ /bin/ls
arendre.css   Bureau      images   Musique   Téléchargements
arendre.html  Documents   Images   Public    Vidéos
bin_perso     fic6.txt    Modèles  snap      wordpress.sql
```

3. En utilisant son nommage absolu `/bin/ls`, créez un nouvel alias de la commande `ls` que vous nommerez `monls`

```
aryouko@aryouko-IdeaPad-1-14ALC7: $ alias monls=/bin/ls
aryouko@aryouko-IdeaPad-1-14ALC7: $ monls
arendre.css   Bureau      images   Musique   Téléchargements
arendre.html  Documents   Images   Public    Vidéos
bin_perso     fic6.txt    Modèles  snap      wordpress.sql
```

4. Sauvegardez le contenu de `PATH` dans `OLDPATH`

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ OLDPATH=$PATH
aryouko@aryouko-IdeaPad-1-14ALC7:~$ echo $OLDPATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:
/snap/bin:/snap/bin:/home/aryouko/bin_perso
```

5. Tapez la suite de commandes `PATH=.. && pwd && clear`
Décrivez ce qu'il se passe. Expliquez d'où vient le problème

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ PATH=.. && pwd && clear
/home/aryouko
La commande « clear » est disponible aux emplacements suivants
 * /bin/clear
 * /usr/bin/clear
La commande n'a pas pu être trouvée car « /bin:/usr/bin » n'est pas incluse dans la varia
ble d'environnement PATH.
clear : commande introuvable
```

La commande n'a pas pu être trouvé car elle n'existe pas dans le PATH car le PATH est modifié en même temps, on ne peut donc plus y accéder.

6. Lancez la commande `ls`. Que se passe-t-il ? Lancez la commande `monls`. Essayez d'expliquer pourquoi cette dernière fonctionne.

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ ls
La commande « ls » est disponible aux emplacements suivants
 * /bin/ls
 * /usr/bin/ls
La commande n'a pas pu être trouvée car « /bin:/usr/bin » n'est pas incluse dans la varia
ble d'environnement PATH.
ls : commande introuvable
aryouko@aryouko-IdeaPad-1-14ALC7:~$ monls
arendre.css   Bureau     images   Musique  Téléchargements
arendre.html  Documents  Images   Public   Vidéos
bin_perso     fic6.txt   Modèles  snap     wordpress.sql
```

monls ne passe pas par le PATH pour être exécuté alors que le ls lui passe par et donc vu que le PATH n'est plus utilisable c'est impossible de l'utiliser.

7. Restaurez l'ancienne valeur de `PATH` que vous aviez sauvegardée dans `OLDPATH`.

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ PATH=$OLDPATH
aryouko@aryouko-IdeaPad-1-14ALC7:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:
/snap/bin:/snap/bin:/home/aryouko/bin_perso
```

Exercice 5: Les redirections

Une commande UNIX est une "boîte noire" à laquelle on fournit des arguments sur un fichier logique nommé *entrée standard* et qui renvoie deux types de réponses éventuelles sur des fichiers logiques appelés respectivement *sortie standard* et *sortie erreur standard*, comme illustré à la fi-gure 1.

FIG. 1 – commande UNIX

Quelques exemples :

- La commande cat écrit sur la sortie standard ce qu'elle reçoit sur l'entrée standard.
- La commande date ne prend rien sur le flot d'entrée et renvoie quelque chose sur le flot de sortie.
- La commande cd ne prend rien en entrée et ne renvoie rien en sortie.

Toutes ces commandes peuvent évidemment renvoyer quelques choses sur la sortie erreur standard pour signaler un problème quelconque (mauvaise utilisation de la commande, arguments inexistants, problèmes de droits, etc.).

Par commodité, il est souvent nécessaire de sauver les données fournies par un programme dans un fichier pour pouvoir ensuite les voir en totalité ou les traiter ou inversement de réutiliser des données qui sont déjà dans un fichier. Par défaut, le canal d'entrée est le clavier et les canaux de sortie et sortie erreur sont l'écran. Mais il est très facile de leur substituer un fichier.

Redirection de la sortie : les symboles > et >>

question 1 (*Redirection de sortie)*

- tapez ps dans une fenêtre de terminal.

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ ps
    PID TTY          TIME CMD
   8426 pts/1    00:00:00 bash
   8580 pts/1    00:00:00 ps
```

- tapez ensuite ps > fic. Un nouveau fichier nommé fic a été créé.

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ ps > fic
aryouko@aryouko-IdeaPad-1-14ALC7:~$ ls
arendre.css   Bureau      fic6.txt  Modèles  snap                wordpress.sql
arendre.html  Documents   images    Musique  Téléchargements
bin_perso     fic         Images    Public   Vidéos
```

- Regardez son contenu.

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ cat fic
    PID TTY          TIME CMD
   8426 pts/1    00:00:00 bash
   8810 pts/1    00:00:00 ps
```

- maintenant faites date > fic.
- regardez à nouveau le contenu du fichier fic.

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ date > fic
aryouko@aryouko-IdeaPad-1-14ALC7:~$ cat fic
jeu. 28 nov. 2024 19:21:39 CET
```

- concluez sur le rôle de >.

> permet d'enregistrer le résultat d'une commande donner avant le > dans un fichier créé avec le nom donné après le >. Le > permet donc l'enregistrement d'une info dans un fichier.

question 2 (*Redirection de sortie)*

- faites who >> fic.
- regardez à nouveau le contenu du fichier fic.

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ who >> fic
aryouko@aryouko-IdeaPad-1-14ALC7:~$ cat fic
jeu. 28 nov. 2024 19:27:20 CET
aryouko  seat0        2024-11-28 14:12 (login screen)
aryouko  tty2         2024-11-28 14:12 (tty2)
```

- concluez sur le rôle de >>.

>> permet d'enregistrer comme celui d'avant (>) mais en rajoutant dans le fichier ci il existe, c'est-à-dire qu'il ne supprime pas les lignes déjà écrite mais en rajoute.

Redirection de l' entrée : le symbole <

Le symbole < sert à rediriger le flot d'entrée.

Question 3 (Redirection d'entrée)

La commande cat prend ce qui lui arrive dans le flux d'entrée et le réécrit en sortie.

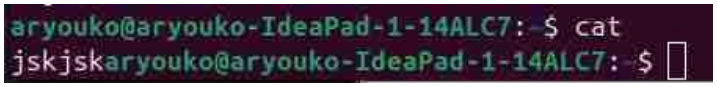
- lancez cat sans argument.
- l'invite change de forme pour vous permettre de fournir des données.
- frappez quelques caractères puis frappez la touche entrée.
- Observez le résultat.

```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ cat
Bonjour
Bonjour
```

- frappez encore quelques lignes, puis pour indiquer au processus la fin des données à traiter, pressez la combinaison de touches Ctrl-D (aussi appelé EOF, ou *end of file*).

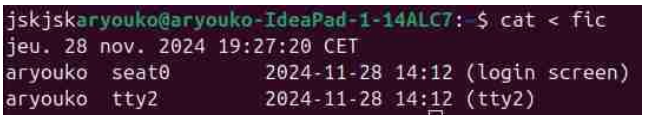
```
aryouko@aryouko-IdeaPad-1-14ALC7:~$ cat
Bonjour
Bonjour
hdgd
hdgd
uej
uej
aryouko@aryouko-IdeaPad-1-14ALC7:~$ cat
```

- que se passe-t-il si à la fin d'une ligne contenant des caractères, vous pressez Ctrl-C au lieu de Ctrl-D ? Expliquez.



Quand on presse Ctrl-D alors qu'on a écrit quelque chose cela renvoie déjà la chose comme pour quand on presse Entrée mais sur la même ligne et si on represse Ctrl-D alors là ça remettra la ligne normal mais sur la même ligne aussi.

- lancez maintenant la commande cat < fic.



- comment expliquez-vous ce qui se passe ?

Le < permet de dire quel fichier ouvrir, entre autre c'est comme juste écrire le cat sans le < car le < demande juste l'entrée comme le fait la commande cat de base. Ici le < est inutile donc.

Exercice 6: (bonus, plus de copie d'écran nécessaire)

Question 1

Indiquez pour chaque variable ci-dessous leur valeur et leur signification :

echo $variable pour avoir la valeur

- USER **Valeur : aryouko et signification : le nom de l'utilisateur actuelle**
- HOME **Valeur : /home/aryouko et signification : chemin du répertoire de l'utilisateur**
- EDITOR **Valeur : la valeur est vide et signification : éditeur de texte par défaut utilisé dans le terminal**
- SHELL **Valeur : /bin/bash et signification : shell par défaut**
- LOGNAME **Valeur : aryouko et signification : nom de connexion de l'utilisateur**
- PATH **Valeur : /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin:/home/aryouko/bin_perso et signification : liste des chemins de répertoires où sont situés les commandes**
- LANG **Valeur : fr_FR.UTF-8 et signification : langue et encodage du système**

- TERM **Valeur : xterm-256color et signification : type du terminal utilisé par le shell**
- MAIL **Valeur : la valeur est vide et signification : chemin du répertoire où les courriels de l'utilisateur sont stockés**

Question 2

Indiquez comment supprimer une variable d'environnement ?

Montrez que cela fonctionne.

```
aryouko@aryouko-IdeaPad-1-14ALC7: $ test="Bonjour"
aryouko@aryouko-IdeaPad-1-14ALC7: $ export test
aryouko@aryouko-IdeaPad-1-14ALC7: $ echo $test
Bonjour
aryouko@aryouko-IdeaPad-1-14ALC7: $ unset test
aryouko@aryouko-IdeaPad-1-14ALC7: $ echo $test
```

Question 3

Récupérez le jeu tetris à l'adresse suivante :

http://gitlab.iu-vannes.org/root/tetris

C'est le fichier tetris.sh qui est à récupérer.

Il y a plusieurs erreurs dans ce jeu, corrigez les pour qu'il fonctionne parfaitement, voici quelques erreurs possibles indicatives :

- Les pièces sont toutes colorées en blanc au lieu d'avoir plusieurs couleurs.

```
85    colors=($RED $GREEN $YELLOW $BLUE $FUCHSIA $CYAN $WHITE)
```

Tout était WHITE il faut donc changer avec les couleurs existantes.

- L'aide en cours de jeu ne s'affiche pas en haut a droite de l'aire de jeu, celle pour indiquer comment faire une rotation, etc.

```
s: rotate
```

Cela est déjà écrit rien à changer

- La prochaine pièce doit s'afficher en bas a gauche, mais elle ne s'affiche pas.

Cela marche déjà aussi

- La flèche de droite ou gauche ne fonctionne pas

```
EFT=2
OTATE=3
```

Les deux était à 2, il fallait mettre à 3 ROTATE.

Autres erreurs :

- (l. 286) for ((i = 0; i < 4 : i++)) { il faut mettre : for ((i = 0; i < 4; i++)) {
- (l. 226) draw_piese() { il faut mettre : draw_piece() {

Premier fichier est celui modifié

```
aryouko@aryouko-IdeaPad-1-14ALC7: $ diff bin_perso/tetris.sh Téléchargements/tetris.sh
38c38
< ROTATE=3
---
> ROTATE=2
85c85
< colors=($RED $GREEN $YELLOW $BLUE $FUCHSIA $CYAN $WHITE)
---
> colors=($WHITE $WHITE $WHITE $WHITE $WHITE $WHITE $WHITE)
127c127
<     ((use_color)) && puts "\033[0m"
---
>     puts "\033[0m"
162c162
<     if (( score/10 > LEVEL_UP * level)) ; then          # if level should be increased
---
>     if (( score > LEVEL_UP * level)) ; then          # if level should be increased
226c226
< draw_piece() {
---
> draw_piese() {
286c286
<     for ((i = 0; i < 4; i++)) {
---
>     for ((i = 0; i < 4: i++)) {
```