TP n°4
Système

1. **Faire** un script nommé `chpr` qui reçoit un argument et qui change le prompt (l'invite principale du shell) avec le contenu de l'argument reçu, suivi du caractère '$'. Penser à vérifier que la syntaxe est juste, c'est-à-dire que le script reçoit bien un et un seul argument. ***Cf. l'aide en page suivante.***

2. **La variable** d'environnement PATH est-elle positionnée ? Que vaut-elle ? (utiliser `printenv`)

3. **Ecrire** une commande (en une ligne) qui affiche le premier chemin indiqué dans PATH.

4. **Ecrire** un script `numstd` qui teste si l'argument passé est un nombre (utiliser `expr` et son code retour) et qui affiche son nombre de chiffres.

5. **Créer** un fichier appelé `notes.txt` contenant les 5 lignes suivantes :

```
Jean Dupond:M1MIASHS:8:10:11:13
Jean Aymar:M1MIASHS:5:10:2:3
Amine Derien:M1MIASHS:20:16:13:11
Carole Hemain:M1MIASHS:8:4:20:8
Sophie Font:M1MIASHS:10:10:10:10
```

   a. A partir de ce fichier, créer un nouveau fichier appelé `moyennes.txt` contenant la même chose que `notes.txt`, mais avec la **somme** puis la **moyenne** à la fin de chaque ligne. Il faut bien sûr calculer la somme et la moyenne par programme (avec `while read` et `(())` notamment). Que constatez-vous ?

   b. Modifier le script en intégrant la commande `bc` qui permet des calculs flottants.

   c. Créer ensuite, grâce à `sort`, un fichier `classement.txt` contenant la liste des étudiants (`Prénom Nom:groupe:moyenne`) classés, selon leur moyenne.

6. **Ecrire** un script `lasomme.sh` qui prend un nombre quelconque de paramètres et calcule leur somme totale. On utilisera `shift`.

7. **Question subsidiaire** : Reprendre **l'exercice 5** et toutes ses questions, mais, cette fois, ***on ne connaît pas à l'avance le nombre de notes*** des étudiants. Donc on travaillera sur le fichier suivant :

```
Anne Auraque:M1G2M:18:10:11:13:7:11:12
Edith Avuleur:M1G2M:6:2:3
Sarah Fraichy:M1G2M:20:16:13:11:3
Alain Dis:M1G2M:8:4:20:8:12
Elie Coptaire:M1G2M:10:10:10:10
```

Aide pour connaître le nombre de champs : https://linuxhint.com/20_awk_examples/#a8

Et aide pour le tri (`sort`) : regardez l'option `-g` (`general-numeric-sort`). Cette option permet de trier tous les nombres (flottants, nombres en hexadécimal, etc.). Mais, pour les flottants, il faut également ***vérifier la valeur de la locale*** `LC_NUMERIC`. *Si elle est positionnée à `fr_FR.UTF-8`, il faut la forcer (au moins pour l'exercice) à `en_US.UTF-8` car les calculs de `bc` renvoient des flottants en notation pointée => 12,5 de moyenne s'écrit `12.5`. Donc, il faut écrire :* `export LC_NUMERIC="en_US.UTF-8"` avant de lancer le script (et donc le `sort`)

Aide Exercice 1 – Prompt Statement variables *(source : http://ss64.com/bash/syntax-prompt.html)*

There are several variables that can be set to control the appearance of the bach command prompt: PS1, PS2, PS3, PS4 and PROMPT_COMMAND the contents are executed just as if they had been typed on the command line.

PS1 – Default interactive prompt (this is the variable most often customized)
PS2 – Continuation interactive prompt (when a long command is broken up with `\` at the end of the line) default=">"
PS3 – Prompt used by "select" loop inside a shell script
PS4 – Prompt used when a shell script is executed in debug mode ("set -x" will turn this on) default ="++"
PROMPT_COMMAND - If this variable is set and has a non-null value, then it will be executed just before the PS1 variable.

Set your prompt by changing the value of the PS1 environment variable, as follows:

```
$ export PS1="My simple prompt> "
>
```

This change can be made permanent by placing the "export" definition in your `~/.bashrc` file.

Special prompt variable characters:

```
   \d    The date, in "Weekday Month Date" format (e.g., "Tue May 26").
   \h    The hostname, up to the first . (e.g. deckard)
   \H    The hostname. (e.g. deckard.SS64.com)
   \j    The number of jobs currently managed by the shell.
   \l    The basename of the shell's terminal device name.
   \s    The name of the shell, the basename of $0 (the portion following the final slash).
   \t    The time, in 24-hour HH:MM:SS format.
   \T    The time, in 12-hour HH:MM:SS format.
   \@    The time, in 12-hour am/pm format.
   \u    The username of the current user.
   \v    The version of Bash (e.g., 2.00)
   \V    The release of Bash, version + patchlevel (e.g., 2.00.0)
   \w    The current working directory.
   \W    The basename of $PWD.
   \!    The history number of this command.
   \#    The command number of this command.
   \$    If you are not root, inserts a "$"; if you are root, you get a "#" (root uid = 0)
   \nnn   The character whose ASCII code is the octal value nnn.
   \n    A newline.
   \r    A carriage return.
   \e    An escape character (typically a color code).
   \a    A bell character.
   \\    A backslash.
   \[    Begin a sequence of non-printing characters.(like color escape sequences). This allows
bash to calculate word wrapping correctly.
   \]    End a sequence of non-printing characters.
```

Using **single** quotes instead of double quotes when exporting your PS variables is recommended, it makes the prompt a tiny bit faster to evaluate plus you can then do an `echo $PS1` to see the current prompt settings.

Command/History Numbers

The command number and the history number are usually different: the history number of a command is its position in the history list, which can include commands restored from the history file, while the command number is the position in the sequence of commands executed.

After the string is decoded, it is expanded via parameter expansion, command substitution, arithmetic expansion, and quote removal, subject to the value of the `promptvars` shell option.

Color Codes (ANSI Escape Sequences)
Foreground colors, Normal (non-bold) is the default, so the `0;` prefix is optional.
`\e[0;30m` = Dark Gray
`\e[1;30m` = Bold Dark Gray
`\e[0;31m` = Red
`\e[1;31m` = Bold Red
`\e[0;32m` = Green
`\e[1;32m` = Bold Green
`\e[0;33m` = Yellow
`\e[1;33m` = Bold Yellow
`\e[0;34m` = Blue
`\e[1;34m` = Bold Blue
`\e[0;35m` = Purple
`\e[1;35m` = Bold Purple
`\e[0;36m` = Turquoise
`\e[1;36m` = Bold Turquoise
`\e[0;37m` = Light Gray
`\e[1;37m` = Bold Light Gray

Background colors:
`\e[40m` = Dark Gray
`\e[41m` = Red
`\e[42m` = Green
`\e[43m` = Yellow
`\e[44m` = Blue
`\e[45m` = Purple
`\e[46m` = Turquoise
`\e[47m` = Light Gray

An alternative, more portable and more intuitive method of setting colors is to use tput, this will read the terminfo database with all the escape codes necessary for interacting with your terminal.
If you use tput instead of hard-coding the escape sequences, you can set the TERM variable to control whether color codes are output. This also makes the output portable to terminals other than xterm.

Examples

Set a prompt like: [username@hostname:~/CurrentWorkingDirectory]$
```
export PS1='[\u@\h:\w]\$ '
```

Set a prompt in color, note the escapes for the non printing characters [ ] , these ensure that readline can keep track of the cursor position correctly.
```
export PS1='\[\e[31m\]\u@\h:\w\[\e[0m\] '
```

Set a prompt with both foreground and background colors, set via tput:
```
magenta=$(tput setaf 5)
blue=$(tput setaf 4)
reset=$(tput sgr0)
export PS1='[\[$magenta\]\u\[$reset\]@\[$magenta\]\h\[$reset\]:\[$blue\]\w\[$reset\]]\\$ '
```