

TP n° 4 - Boucles - Compléments sur les scripts

Pour cette séance, comme pour toutes les séances de TP, vous commencez par créer un répertoire TP4 dans votre arborescence personnelle.

Ensuite, pour chaque exercice, créez un sous-répertoire dans TP4.

Pour chaque exercice, vous devrez écrire un ou plusieurs scripts. Si nécessaire, n'hésitez pas à :

- utiliser des fichiers temporaires pour stocker des données intermédiaires
- utiliser des variables pour stocker le résultat d'une commande avec la syntaxe `var=$(commande)`
- faire des opérations arithmétiques avec la syntaxe `=$((i + 1))` ou `$(($i + 1))`
- utiliser des boucles `for` (avec `seq`)
- utiliser des boucles `while` avec des conditions
- utiliser des instructions conditionnelles `if` ou `case`
- écrire les commandes dans un script...

Remarque : des commandes utiles sont parfois données en indication, on prendra le temps d'en lire le manuel grâce à la commande `man <nom de la commande>`.

Exercice 1. While, break et case

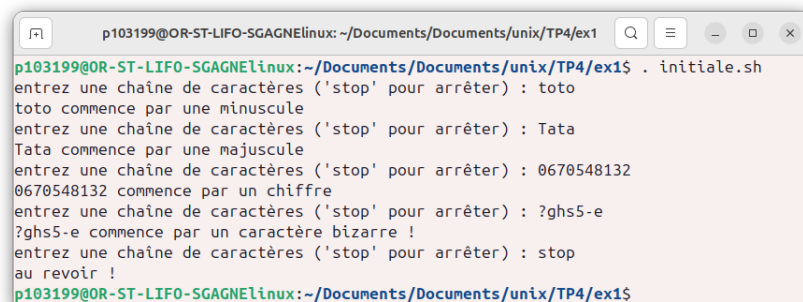
Créez dans TP4 un sous-répertoire `ex1`.

Point technique 1 : la commande `true` est une commande qui retourne immédiatement le code de retour 0, c'est donc l'équivalent du booléen classique `true` des divers langages.

Point technique 2 : on peut créer une boucle d'apparence "infinie" avec `while true`, mais on en contrôle la sortie par l'instruction `break` si un test est réalisé. C'est une façon parmi d'autres d'écrire les boucles `while`.

1. En utilisant une boucle `while true` et un `break`, écrivez un script `initiale.sh` qui :

- demande une chaîne de caractères à l'utilisateur,
- tant que la chaîne n'est pas "stop", affiche un message qui est adapté à l'initiale de la chaîne, comme ci-dessous,
- quand la chaîne "stop" est entrée, affiche un message de fin.



```
p103199@OR-ST-LIFO-SGAGNElinux: ~/Documents/Documents/unix/TP4/ex1
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex1$ . initiale.sh
entrez une chaîne de caractères ('stop' pour arrêter) : toto
toto commence par une minuscule
entrez une chaîne de caractères ('stop' pour arrêter) : Tata
Tata commence par une majuscule
entrez une chaîne de caractères ('stop' pour arrêter) : 0670548132
0670548132 commence par un chiffre
entrez une chaîne de caractères ('stop' pour arrêter) : ?ghs5-e
?ghs5-e commence par un caractère bizarre !
entrez une chaîne de caractères ('stop' pour arrêter) : stop
au revoir !
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex1$
```

2. Écrivez une nouvelle version du script, `initialev2.sh` qui utilise cette fois une boucle `while` avec une condition, et aucun `break`.

Exercice 2. Suite de Syracuse

Créez dans TP4 un sous-répertoire **ex2**.

Point culture : Une suite de Syracuse est une suite d'entiers qui illustre la conjecture de Syracuse.

On la définit en donnant son premier terme u_0 et la relation de récurrence suivante :

$$u_{n+1} = \begin{cases} \frac{u_n}{2}, & \text{si } u_n \text{ est pair,} \\ 3u_n + 1, & \text{si } u_n \text{ est impair.} \end{cases}$$

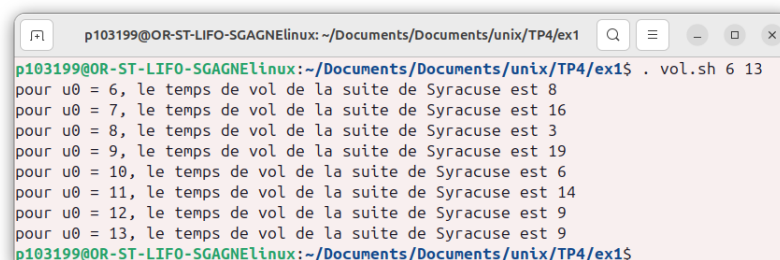
La conjecture de Syracuse affirme que, quelle que soit la valeur de u_0 , la suite finit par arriver à la valeur 1 pour un certain n .

1. Calculez à la main les termes de la suite de Syracuse jusqu'à 1 pour les valeurs $u_0 = 5$, $u_0 = 6$ et $u_0 = 7$.
2. Écrivez un script **suite.sh** qui prend en argument un entier strictement positif et qui affiche la suite de Syracuse commençant par la valeur fournie, jusqu'à obtenir le 1 attendu.



```
p103199@OR-ST-LIFO-SGAGNElinux: ~/Documents/Documents/unix/TP4/ex1
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex1$ . suite.sh 13
u(0) = 13
u(1) = 40
u(2) = 20
u(3) = 10
u(4) = 5
u(5) = 16
u(6) = 8
u(7) = 4
u(8) = 2
u(9) = 1
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex1$
```

3. Le "temps de vol" est le premier indice N pour lequel $u_N = 1$. Écrivez un script **vol.sh** qui prend en argument deux entiers strictement positifs (supposés donnés dans l'ordre croissant) et qui affiche le temps de vol des suites de syracuse correspondant aux valeurs initiales u_0 comprises entre ces deux arguments. Ce script a la même base que **suite.sh**.



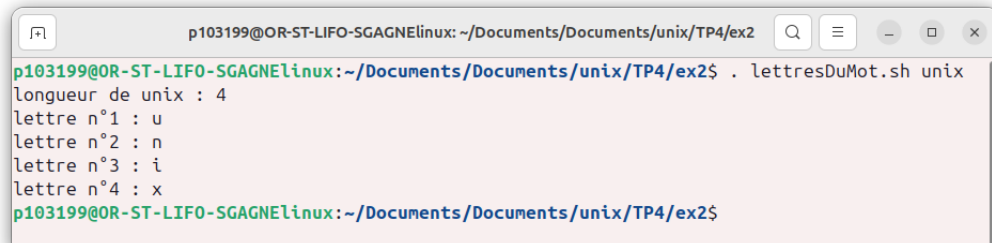
```
p103199@OR-ST-LIFO-SGAGNElinux: ~/Documents/Documents/unix/TP4/ex1
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex1$ . vol.sh 6 13
pour u0 = 6, le temps de vol de la suite de Syracuse est 8
pour u0 = 7, le temps de vol de la suite de Syracuse est 16
pour u0 = 8, le temps de vol de la suite de Syracuse est 3
pour u0 = 9, le temps de vol de la suite de Syracuse est 19
pour u0 = 10, le temps de vol de la suite de Syracuse est 6
pour u0 = 11, le temps de vol de la suite de Syracuse est 14
pour u0 = 12, le temps de vol de la suite de Syracuse est 9
pour u0 = 13, le temps de vol de la suite de Syracuse est 9
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex1$
```

Exercice 3. Travaux sur les mots

Créez dans TP4 un sous-répertoire **ex3**.

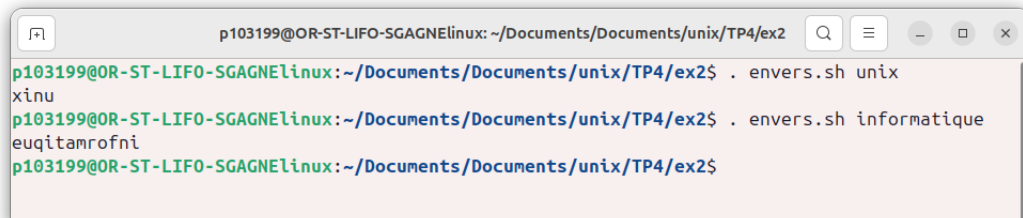
Point technique : la longueur d'une chaîne de caractères **mot** peut être obtenu par `${#mot}`

1. Écrivez un script **lettresDuMot.sh** qui prend en argument une chaîne de caractères et qui affiche la longueur de la chaîne ainsi que chaque lettre, comme ci-dessous :



```
p103199@OR-ST-LIFO-SGAGNElinux: ~/Documents/Documents/unix/TP4/ex2
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex2$ . lettresDuMot.sh unix
longueur de unix : 4
lettre n°1 : u
lettre n°2 : n
lettre n°3 : i
lettre n°4 : x
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex2$
```

2. Écrivez un script **nbOccurrences.sh** qui prend en arguments une chaîne de caractères et une lettre et qui affiche dans un message adapté combien de fois la lettre apparaît dans la chaîne.
3. (plus difficile) Écrivez un script **lettreLaPlusFrequente.sh** qui prend en argument une chaîne de caractères et qui affiche dans un message adapté quelle est la lettre la plus fréquente dans la chaîne, et combien de fois elle apparaît. S'il y a plusieurs lettres répondant à la question, on n'en affiche qu'une.
4. Écrivez un script **envers.sh** qui prend en argument une chaîne de caractères et qui affiche la chaîne "à l'envers", comme ci-dessous :



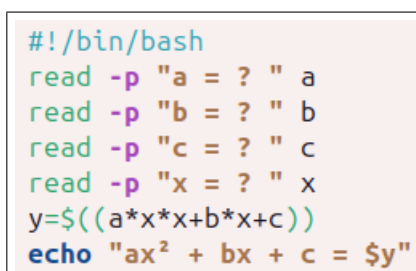
```
p103199@OR-ST-LIFO-SGAGNElinux: ~/Documents/Documents/unix/TP4/ex2
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex2$ . envers.sh unix
xinu
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex2$ . envers.sh informatique
euqitamrofni
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex2$
```

5. Copiez **envers.sh** en **palindrome.sh** pour avoir la même base. Modifiez ensuite **palindrome.sh** pour qu'il dise si le mot passé en argument est un palindrome ou non.

Exercice 4. La commande bc

Créez dans TP4 un sous-répertoire **ex4**.

Point technique : la commande **bc** vient compléter ce que nous savons sur l'évaluation des opérations arithmétiques comme par exemple



```
#!/bin/bash
read -p "a = ? " a
read -p "b = ? " b
read -p "c = ? " c
read -p "x = ? " x
y=$((a*x*x+b*x+c))
echo "ax2 + bx + c = $y"
```

Ce type de calcul fonctionne pour des **valeurs entières** des variables. Mais **pas pour des valeurs décimales** !

Pour étendre les résultats aux valeurs décimales, il y a la commande `bc` dont voici quelques exemples d'utilisations avec un `pipe` :

```
p103199@OR-ST-LIFO-SGAGNElinux: ~  
p103199@OR-ST-LIFO-SGAGNElinux:~$ echo "a=5; b=3; 2*a+b^2" | bc  
19  
p103199@OR-ST-LIFO-SGAGNElinux:~$ echo "x=1.58; y=3.71; x^3+y^2" | bc  
17.70  
p103199@OR-ST-LIFO-SGAGNElinux:~$ echo "scale=4; x=1.58; y=3.71; x^3+y^2" | bc  
17.7084  
p103199@OR-ST-LIFO-SGAGNElinux:~$ echo "scale=5; x=1.58; y=3.71; x^3+y^2" | bc  
17.70841  
p103199@OR-ST-LIFO-SGAGNElinux:~$
```

1. Adaptez le script donné en exemple en début d'exercice en écrivant un script `secondDegre.sh` qui fonctionnera avec des valeurs décimales pour `a`, `b`, `c`, `x`. Vous pourrez créer une variable `expr_y` qui contiendra l'expression de `y` (chaîne de caractères) et l'utiliser avec `bc`.

```
p103199@OR-ST-LIFO-SGAGNElinux: ~/Documents/Documents/unix/TP4/ex3  
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex3$ . secondDegre.sh  
a ? 2.19  
b ? -6.57  
c ? 3.18  
x ? 6.13  
nombre de décimales ? 5  
45.19931  
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex3$ . secondDegre.sh  
a ? 2.19  
b ? -6.57  
c ? 3.18  
x ? 1.14  
nombre de décimales ? 3  
-1.465  
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex3$
```

2. Écrivez un script `fonction.sh` qui demande à l'utilisateur d'entrer :

- l'expression algébrique correcte d'une fonction de la variable `x`. L'utilisateur entre une chaîne qui a un sens algébrique exact : bien parenthésée et avec des opérations explicites : pas $2x$ mais bien $2 * x$ par exemple,
- la valeur de `x` pour laquelle l'image sera calculée,
- le nombre de décimales souhaitées pour le résultat.

Voici deux exemples pour $f(x) = 3.1x^2 - 2.7x + \frac{3.4}{x}$ et pour $f(x) = 2x + 1 - \frac{1.5}{2x-7}$

```
p103199@OR-ST-LIFO-SGAGNElinux: ~/Documents/Documents/unix/TP4/ex3  
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex3$ . fonction.sh  
entrez l'expression de f(x) : 3.1*x^2-2.7*x+3.4/x  
entrez la valeur de x : 1.72  
entrez le nombre de décimales du résultat : 4  
f(1.72) = 6.5037  
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex3$ . fonction.sh  
entrez l'expression de f(x) : 2*x+1-1.5/(2*x-7)  
entrez la valeur de x : 3.568  
entrez le nombre de décimales du résultat : 6  
f(3.568) = -2.893411  
p103199@OR-ST-LIFO-SGAGNElinux:~/Documents/Documents/unix/TP4/ex3$
```