

Compte-rendu du projet Cowsay

Mathias Rivet

26 avril 2023

Les codes sources sont aussi disponible à l'adresse : <https://github.com/programindfr/INF203-Projet>

Table des matières

| | | |
|----------|----------------------|----------|
| 1 | Préliminaires | 1 |
| 2 | Bash | 4 |
| 2.1 | cow kindergarten | 4 |
| 2.2 | cow primaryschool | 5 |
| 2.3 | cow highschool | 5 |
| 2.4 | cow college | 6 |
| 2.5 | cow university | 7 |
| 2.6 | smart cow | 8 |
| 2.7 | crazy cow | 9 |

1 Préliminaires

| Argument | Usage | Exemple |
|----------|---|--|
| -n | Permet d'afficher le texte sans qu'il ne soit formaté. | <pre>mathias@pc:~\$ cowsay -n Hello World ! Ctrl+D ----- / Hello \ \ World ! / ----- \ ^--^ (oo)_____) (__) \)\\/ ----w </pre> |
| -W | Permet de choisir le nombre de colonnes avant un retour à la ligne. | <pre>mathias@pc:~\$ cowsay -W 5 Hello World ! ----- / Hell \ o Worl \ d ! / ----- \ ^--^ (oo)_____) (__) \)\\/ ----w </pre> |

| | | |
|----|---|--|
| -b | La vache se transforme en cyborg. | <pre>mathias@pc:~\$ cowsay -b Hello World !</pre> <pre> ----- < Hello World ! > ----- ^__^ (==)___ (__)\/)\/\ ----w </pre> |
| -d | La vache semble morte. | <pre>mathias@pc:~\$ cowsay -d Hello World !</pre> <pre> ----- < Hello World ! > ----- ^__^ (xx)___ (__)\/)\/\ U ----w </pre> |
| -g | La vache a des yeux avides. | <pre>mathias@pc:~\$ cowsay -g Hello World !</pre> <pre> ----- < Hello World ! > ----- ^__^ (\$\$)___ (__)\/)\/\ ----w </pre> |
| -p | La vache est devenue folle. | <pre>mathias@pc:~\$ cowsay -p Hello World !</pre> <pre> ----- < Hello World ! > ----- ^__^ (@@)___ (__)\/)\/\ ----w </pre> |
| -s | La vache est sous l'emprise de stupéfiants. | <pre>mathias@pc:~\$ cowsay -s Hello World !</pre> <pre> ----- < Hello World ! > ----- ^__^ (**)___ (__)\/)\/\ U ----w </pre> |

| | | |
|----|---|---|
| -t | Affiche une vache fatiguée. | <pre>mathias@pc:~\$ cowsay -t Hello World !</pre> <pre> ----- < Hello World ! > ----- /\ /\ /\ /\ /\ /\ /\ </pre> |
| -w | La vache est surexcitée. | <pre>mathias@pc:~\$ cowsay -w Hello World !</pre> <pre> ----- < Hello World ! > ----- /\ /\ /\ /\ /\ /\ /\ </pre> |
| -y | Affiche une jeune vache. | <pre>mathias@pc:~\$ cowsay -y Hello World !</pre> <pre> ----- < Hello World ! > ----- /\ /\ /\ /\ /\ /\ /\ </pre> |
| -e | L'utilisateur peut choisir lui-même deux caractères pour les yeux de la vache. | <pre>mathias@pc:~\$ cowsay -e "><" Hello World !</pre> <pre> ----- < Hello World ! > ----- /\ /\ /\ /\ /\ /\ /\ </pre> |
| -T | L'utilisateur peut choisir lui-même deux caractères pour la langue de la vache. | <pre>mathias@pc:~\$ cowsay -T "\" Hello World !</pre> <pre> ----- < Hello World ! > ----- /\ /\ /\ /\ /\ /\ /\ </pre> |

Exemple

```
mathias@pc:~$ chmod +x cow_kindergarten
mathias@pc:~$ ./cow_kindergarten

-----
< 10 >
-----

      \      ^__^
      (oo)\_______
           (__)\\       )\/\
               U     ||----w |
                   ||     ||
```

2.2 cow primaryschool

Script

```
#!/bin/bash

for i in $(seq $1)
do
    clear
    if [ $i -eq $1 ]          # test si i est égal au premier argument
    then
        cowsay -T "U " $i    # si oui la vache tire la langue
    else
        cowsay $i
        sleep 1
    fi
done
```

Exemple

```
mathias@pc:~$ chmod +x cow_primaryschool
mathias@pc:~$ ./cow_primaryschool 20

-----
< 20 >
-----

      \      ^__^
      (oo)\_______
           (__)\\       )\/\
               U     ||----w |
                   ||     ||
```

2.3 cow highschool

Script

```
#!/bin/bash

for i in $(seq $1)
do
    clear
    if [ $i -eq $1 ]          # test si i est égal au premier argument
    then
        cowsay -T "U " $(( $i * $i )) # si oui la vache tire la langue
    else
        cowsay $(( $i * $i ))          # on affiche le carré de i
        sleep 1
    fi
done
```

Example

[illegible]

2.4 cow college

Script Pour une raison non élucidée, `cowsay 0` attend un input depuis `stdin`, ce qui bloque l'exécution. Il faut donc rajouter un espace pour corriger le problème.

```
#!/bin/bash

f0=0
f1=1

if [ $1 -eq 1 ]
then
    cowsay -T "U " "$f0 "
elif [ $1 -ge 2 ]
then
    cowsay "$f0 "
    sleep 1
    clear
    cowsay $f1
    sleep 1
    while
        fn=$((f0 + f1))
        [ $fn -lt $1 ]
    do
        clear
        f0=$f1
        f1=$fn
        cowsay $f1
        if [ $((f0 + f1)) -lt $1 ]
        then
            sleep 1
        fi
    done
    clear
    cowsay -T "U " $f1
fi

# On définit f1 et f0.

# Pour n égal à 1, c'est déjà la fin
# et la vache tire la langue.

# On ajoute un espace sinon cowsay
# reste bloqué pour afficher 0.
# Pour n supérieur ou égal à 2, f0
# et f1 sont affichés.

# On ajoute aussi un espace pour 0.

# Est-ce que fn+1 est supérieur à n ?
# Non alors on peut l'afficher.

# On incrémente d'un cran f0 et f1.

# Si ce n'est pas le dernier
# affichage on attend.

# Oui alors la vache tire la langue
# en affichant fn.
```

Exemple

```
mathias@pc:~$ chmod +x cow_college
mathias@pc:~$ ./cow_college 0
mathias@pc:~$ ./cow_college 1
```

```

-----
< 0 >
-----

      \      ^__^
      (oo)\_____.
      (__)\\       )\/\
      U      ||----w |
              ||      ||

mathias@pc:~$ ./cow_college 2

-----
< 1 >
-----

      \      ^__^
      (oo)\_____.
      (__)\\       )\/\
      U      ||----w |
              ||      ||

mathias@pc:~$ ./cow_college 60

-----
< 55 >
-----

      \      ^__^
      (oo)\_____.
      (__)\\       )\/\
      U      ||----w |
              ||      ||

```

2.5 cow university

Script On utilise ici la méthode du crible d'Ératosthène.

```
#!/bin/bash

i=2
while [ $((i * i)) -lt $1 ]
do
    i=$((i + 1))
done

lastPrime=2

for k in $(seq 2 $1)
do
    isPrime=0

    for j in $(seq 2 $i)
    do
        if [ $((k % j)) -eq 0 -a $k -ne $j ]
        then
            isPrime=1
        fi
    done
    if [ $isPrime -eq 0 ]
    then
        if [ $k -gt 2 ]
        then
            sleep 1
        fi
    fi
done

# 0 et 1 ne sont pas des nombres premiers.
# On cherche la racine de n.

# On mémorise le dernier nombre premier
# pour l'afficher en tirant la langue.

# Booleen pour savoir si le nombre est
# premier.

# Si k modulo j est égal à 0 et que k est
# différent de j alors k n'est pas premier
# (pour j quelconque fixé entre 2 et
# racine de n).

# Booleen indiquant que k n'est pas
# premier.

# Si k est premier on l'affiche.

# Pas de pause avant le premier affichage.
```

```

        fi
        lastPrime=$k                                # On mémorise le nombre premier actuel.
        clear
        cowsay $lastPrime                           # On affiche le nombre premier actuel.
    fi
done

if [ $1 -ge 2 ]
then
    clear
    cowsay -T "U " $lastPrime                       # La vache tire la langue pour le dernier
                                                    # nombre premier.
fi

```

Exemple

```

mathias@pc:~$ chmod +x cow_university
mathias@pc:~$ ./cow_university 0
mathias@pc:~$ ./cow_university 1
mathias@pc:~$ ./cow_university 100

```

```

-----
< 97 >
-----
      \  ^__^
       (oo)\_______
            (__)\       )\/\
                U     ||----w |
                   ||     ||

```

2.6 smart cow

Script Les deux points clés de ce script sont les fonctions wc et cut. Le fait que bash interprète les variables selon le contexte est utile pour réduire la taille du code et les répétitions. Ainsi le caractère opérateur est interprété selon le contexte ce qui réduit la taille du code par 5.

BONUS : la vache a aussi appris à calculer le modulo de deux nombres !

```

#!/bin/bash

length=$(( (echo "$1" | wc -c) - 1 )) # On récupère la taille de la chaîne de
                                       # caractère sans compter le caractère \n de
                                       # fin.

for i in $(seq $length)
do
    char=$(echo "$1" | cut -c $i)      # La variable char contient le caractère de la
                                       # chaîne qui se situe à la i-ème position.

    if [ "$char" = '+' -o "$char" = '-' -o "$char" = '*' -o "$char" = '/' -o "$char" = '%'
    ↪ ]

                                       # Si char est un opérateur alors on sait comment
                                       # découper la chaîne puisqu'on connaît sa
                                       # position.

    then
        deb=$(echo "$1" | cut -c 1-$(($i - 1)))
                                       # deb contient les caractères de la position 1 à
                                       # i-1 (la position juste avant l'opérateur).

        fin=$(echo "$1" | cut -c $(($i + 1))-$length)
                                       # fin contient les caractères de la position i+1
                                       # à length (position juste après l'opérateur
                                       # à la position du dernier caractère).

        cowsay -e "$(($deb $char $fin))" "$deb $char $fin"
                                       # Selon le contexte, char est interprété comme un

```



```

                                # opérateur arithmétique ou un caractère.
fi
done

```

Exemple

```

mathias@pc:~$ chmod +x smart_cow
mathias@pc:~$ ./smart_cow '3 + 11'

-----
< 3 + 11 >
-----
      ^--^
      (14)\-----
      (__) \         )\\
           ||-----w |
           ||         ||
mathias@pc:~$ ./smart_cow 3-11

-----
< 3 - 11 >
-----
      ^--^
      (-8)\-----
      (__) \         )\\
           ||-----w |
           ||         ||

```

2.7 crazy cow

Script La crazy_cow affiche les termes de la [suite de Conway](#) jusqu'au terme de rang n avec n un nombre donné en argument du script. La vache marque une pause entre chaque terme et tire la langue pour afficher le dernier terme. Le script de crazy_cow est inspiré de cow_college pour l'aspect de suite définie par récurrence et de smart_cow pour l'aspect d'opérations sur des chaînes de caractères. Attention, il se pourrait que la vache soit surexcitée par le calcul des différents termes.

```

#!/bin/bash

suivant(){
    # On définit une fonction pour le calcul du
    # terme suivant.

    length=$(( (echo "$1" | wc -c) - 1 )) # Calcule la taille du terme actuel.
    xSuivant=""
    nb=0
    j=1
    while [ $j -le $length ] # Itération sur tous les caractères.
    do
        chiffre=$((echo "$1" | cut -c $j)) # Caractère du chiffre à compter.
        nb=1
        while [ "$chiffre" = "$(echo "$1" | cut -c $((j + $nb)))" ]
        # Tant que le caractère suivant correspond au
        # même chiffre on compte le nombre
        # d'occurrences.
        do
            nb=$((nb + 1))
        done
        xSuivant="$xSuivant$nb$chiffre" # On ajoute le nombre d'occurrences du chiffre
        # et le chiffre lui-même.
        j=$((j + $nb)) # On passe au chiffre suivant en ignorant les
        # occurrences du chiffre actuel.
    done
    echo $xSuivant # Équivalent à return.
}

```

```

xn=1                                     # Initialisation de la suite.
for i in $(seq 0 $1)                   # On affiche jusqu'au rang n.
do
    clear
    if [ $i -lt $1 ]
    then
        cowsay -w $xn
        sleep 1
        xn=$(suivant $xn)
    else
        cowsay -T "U " $xn
    fi
done

```

Exemple

```

mathias@pc:~$ chmod +x crazy_cow
mathias@pc:~$ ./crazy_cow 0
---
< 1 >
---
      \  ^--^
       \ (oo)\_-----
        (__) \       )\ \
          U  ||----w |
             ||      ||

mathias@pc:~$ ./crazy_cow 3
-----
< 1211 >
-----
      \  ^--^
       \ (oo)\_-----
        (__) \       )\ \
          U  ||----w |
             ||      ||

mathias@pc:~$ ./crazy_cow 10
-----
< 11131221133112132113212221 >
-----
      \  ^--^
       \ (oo)\_-----
        (__) \       )\ \
          U  ||----w |
             ||      ||

```