Compte Rendu du Projet Cowsay

Aleksandr Shmigelskii, Gabriel Mella, Daniel Bass
IMA-05

Enseignant de TD/TP : Jean-Loup Haberbusch

Table des matières

1. Introduction

2. Préliminaires

Option	Signification / Effet	Exemple d'usage
`-b`	Borg mode : la vache aura un aspect "cyborg".	`cowsay -b "Je suis un Borg"`
`-d`	Dead mode : la vache a des yeux « XX ».	`cowsay -d "Aïe. Je ne me sens pas bien"`
`-g`	Greedy mode : la vache a des yeux « $$ ».	`cowsay -g "J'adore l'argent"`
`-p`	Paranoïd mode : la vache a des yeux « @@ ».	`cowsay -p "Je suis surveillé..."`
`-s`	Stoned mode : la vache a des yeux « ** ».	`cowsay -s "Coucou..."`
`-t`	Tired mode : la vache a des yeux « – ».	`cowsay -t "Je suis épuisée..."`
`-w`	Wired mode : la vache a des yeux « OO ».	`cowsay -w "Je ne tiens plus en place"`
`-y`	Youthful mode : la vache a des yeux « .. ».	`cowsay -y "Je suis toute jeune"`
`-e` *eyes*	Personnalise les yeux (2 caractères).	`cowsay -e ^o "Regarde mes yeux"`
`-T` *tongue*	Personnalise la langue (1 ou 2 caractères).	`cowsay -T "U" "Ma langue est sortie"`
`-f` *cowfile*	Utilise un autre dessin ASCII (fichier .cow).	`cowsay -f small "Vraiment petite!"`
`-r`	Choisit une vache au hasard (fichier .cow).	`cowsay -r "Je suis une vache aléatoire."`
`-l`	Liste les vaches définies dans le chemin **COWPATH**	`cowsay -l`

Table 1 – Principales options de `cowsay`

3. Bash

3.1. cow_kindergarten

Fonctionnalité Ce script fait « dire » à la vache les nombres de 1 à 10 de manière animée :
- `clear` efface l'écran avant chaque itération pour simuler une animation.
- `cowsay $i` affiche le chiffre courant (`i` parcourant {1...10}).
- `sleep 1` introduit une pause d'une seconde entre chaque affichage.
- À la fin, `cowsay -T U "J'ai terminé!"` fait tirer la langue à la vache.

Exemple d'exécution

```
$ ./cow_kindergarten.sh
 ___
<  >
 ---
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
...
 ___
< 9 >
 ---
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
 _____________
<J'ai terminé!>
 -------------
     \  ^__^
      \ (oo)\_______
        (__)\\       )\/\
         U  ||----w |
            ||     ||
```

Commentaires
- Utilisation d'une boucle `for i in {1..10}` appelant `cowsay` à chaque itération.
- **Effet d'animation** : `clear` + `sleep 1` suffisent, pas besoin d'outils externes.

Le code source est fourni dans l'archive (`scripts/cow_kindergarten.sh`).

3.2. cow_primaryschool

Différences principales par rapport à `cow_kindergarten`

- Le nombre d'itérations est désormais fixé par l'argument `$1`.
- Vérification qu'un seul argument est fourni et qu'il est strictement positif :
  - si `$# -ne 1`, on affiche un message d'usage et on quitte ;
  - si `$1 -le 0`, on affiche via `cowsay` « Veuillez fournir un nombre entier positif supérieur à 0 » puis on quitte.
- Boucle `while [ $CMPT -le $N ]` remplace la boucle fixe `{1..10}`.

Exemples d'exécution

```
$ ./cow_primaryschool.sh 5
 ---
< 5 >
 ---
    \   ^__^
     \  (oo)\_______
        (__)\       )\/\
            ||----w |
            ||     ||

$ ./cow_primaryschool.sh -5
 ------------------------------------
/ Veuillez fournir un nombre entier \
\ positif supérieur à 0.            /
 ------------------------------------
    \   ^__^
     \  (oo)\_______
        (__)\       )\/\
            ||----w |
            ||     ||

$ ./cow_primaryschool.sh 5 7
Usage: ./cow_primaryschool.sh <nombre n>
```

Commentaires

- Le test `[ $1 -le 0 ]` intercepte les valeurs non-positives et utilise `cowsay` pour un message d'erreur plus lisible.
- En cas d'erreur, le script quitte immédiatement sans exécuter la boucle principale.
- Le test `[ $1 -le 0 ]` couvre les entiers non positifs, mais si `$1` n'est pas numérique, Bash renvoie une erreur de syntaxe (operand expected), mais nous avons supposé que seuls des chiffres seraient transmis.

Le code complet est disponible dans l'archive (`scripts/cow_primaryschool.sh`).

3.3. cow_highschool

Différences principales par rapport à `cow_primaryschool`

— Au lieu d'énoncer simplement i, la vache énonce son carré i^2 grâce à :
`cowsay $((CMPT * CMPT))`

— La structure générale (vérification d'argument, boucle, `clear`, `sleep`) reste identique.

Exemples d'exécution

```
$ ./cow_highschool.sh 10
 ____
< 36 >
----
    \   ^__^
     \  (oo)\_______
        (__)\       )\/\
            ||----w |
            ||     ||

$ ./cow_highschool.sh -5
 ___________________________________
/ Veuillez fournir un nombre entier \
\ positif supérieur à 0.            /
 -----------------------------------
    \   ^__^
     \  (oo)\_______
        (__)\       )\/\
            ||----w |
            ||     ||

$ ./cow_highschool.sh
Usage: ./cow_highschool.sh <nombre n>
```

Commentaires

— Le calcul du carré utilise l'arithmétique intégrée de Bash (`$((...))`).

— La validation écrite `[ $1 -le 0 ]` couvre les valeurs non-positives, mais un argument non numérique génère une erreur Shell (« operand expected ») non gérée.

Le script complet se trouve dans l'archive (`scripts/cow_highschool.sh`).

3.4. cow_college

Fonctionnalité Ce script énonce les termes de la suite de Fibonacci strictement inférieurs à n :
- Vérification de l'argument : un entier > 1 est requis (`[ $1 -le 1 ]`).
- Initialisation de deux variables `FIB1=1`, `FIB2=1`.
- Boucle `while [ $FIB1 -lt $N ]` :
  - Affichage de `cowsay $FIB1`.
  - Calcul du terme suivant via `NEW=$((FIB1+FIB2))`, décalage `FIB1=$FIB2`, `FIB2=$NEW`.
  - `sleep 1` + `clear` pour l'animation.
- Fin marquée par `cowsay -T U "J'ai terminé!"`.

Exemples d'exécution

```
$ ./cow_college.sh 10
 ___
< 1 >
 ---
    \   ^__^
     \  (oo)\_______
        (__)\       )\/\
            ||----w |
            ||     ||
 ...
 ___
< 8 >
 ---
    \   ^__^
     \  (oo)\_______
        (__)\       )\/\
         U  ||----w |
            ||     ||

$ ./cow_college.sh 1
 ___________________________________
/ Veuillez fournir un nombre entier \
\ positif supérieur à 1.            /
 -----------------------------------
    \   ^__^
     \  (oo)\_______
        (__)\       )\/\
            ||----w |
            ||     ||

$ ./cow_college.sh
Usage: ./cow_college.sh <nombre n>
```

Commentaires
- On ne stocke que deux variables `FIB1`, `FIB2`.
- La condition `while [ $FIB1 -lt $N ]` garantit de n'afficher que les termes strictement inférieurs à N, et stoppe avant le premier terme $\geq N$.
- Validation minimale : un argument non numérique déclenchera une erreur de shell non gérée.

Le code complet est disponible dans l'archive (`scripts/cow_college.sh`).

3.5. cow_university

Fonctionnalité Ce script énonce tous les nombres premiers strictement inférieurs à n :
- Vérification de la présence d'un argument unique et de sa positivité (`[ $1 -le 1 ]`).
- Boucle `CMPT=2` à `CMPT<$N` :
  - Initialisation de `isPrime=1` (on suppose premier).
  - Boucle interne `while [ $i -lt $CMPT ]` testant `$CMPT % $i`.
  - Si un diviseur est trouvé, `isPrime=0` et `break`.
  - Si `isPrime==1`, appel de `cowsay $CMPT`, `sleep 1`, `clear`.
- Fin de l'exercice marquée par `cowsay -T U "J'ai terminé!"`.

Exemples d'exécution

```
$ ./cow_university.sh 20
 ___
< 2 >
 ---
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||

3 ... 19
    \   ^__^
     \  (oo)\_______
        (__)\       )\/\
            ||----w |
            ||     ||

$ ./cow_university.sh -5
 ------------------------------------
/ Veuillez fournir un nombre entier \
\ positif supérieur à 1.            /
 ------------------------------------
    \   ^__^
     \  (oo)\_______
        (__)\       )\/\
            ||----w |
            ||     ||
```

Commentaires
- Algorithme naïf de test de primalité en $O(n^2)$, testant tous les diviseurs jusqu'à `CMPT-1`.
- Interruption précoce dès qu'un diviseur est trouvé (`break`) pour limiter les calculs.
- Validation minimale : un argument non numérique déclenche une erreur de shell non gérée.

Le script complet est disponible dans l'archive (`scripts/cow_university.sh`).

3.6. smart_cow

Fonctionnalité Le script évalue une expression arithmétique simple (addition, soustraction, multiplication, division) passée en argument et affiche le résultat dans les yeux de la vache :

- Vérification qu'un seul argument (la chaîne d'expression) est fourni.
- **Subshell silencieux + redirection** (`( res=$((expr)) ) 2>/dev/null`) pour tester la validité de l'expression sans polluer l'écran.
- Inspection du code de retour (`$?`) : si $\neq 0$, message d'erreur `cowsay "Expression invalide : $expr"` et sortie.
- Re-calcul du résultat hors subshell (`res=$((expr))`) pour récupérer la valeur.
- Détermination de la forme des yeux selon la longueur du résultat :
  - 1 chiffre → `eyes="$res-"`
  - 2 chiffres → `eyes="$res"`
  - >2 chiffres → `eyes="??"` + message d'excuse.
- Affichage final `cowsay -e "$eyes" "$msg"`.

Exemples d'exécution

```
$ ./smart_cow.sh "3+11"
 --------------------------
< Le résultat de 3+11 est 14 >
 --------------------------
    \   ^__^
     \  (14)\_______
        (__)\       )\/\
            ||----w |
            ||     ||

$ ./smart_cow.sh "3+11+"
 -------------------------------
< Expression invalide : 3+11+ >
 -------------------------------
    \   ^__^
     \  (oo)\_______
        (__)\       )\/\
            ||----w |
            ||     ||

$ ./smart_cow.sh 3 + 11
Usage: ./smart_cow.sh "<expression>"
```

Commentaires et difficultés

- Capturer l'erreur d'arithmétique Bash nécessite un subshell et `2>/dev/null`, car `$((...))` renvoie un code $\neq 0$ mais affiche aussi un message sur stderr.
- Refaire le calcul hors subshell est la solution la plus simple pour récupérer `$res`.
- Pas de test explicite sur les caractères de l'expression : si on entre une chaîne non arithmétique, elle est évaluée à 0 sans message d'erreur.
- Gestion des cas « yeux trop petits » et format dynamique des yeux selon la longueur du résultat.

Le script complet est disponible dans l'archive (`scripts/smart_cow.sh`).

3.7. crazy_cow

Fonctionnalité Ce script applique une suite d'opérations arithmétiques successives, de gauche à droite, à partir d'un nombre initial. Il affiche à chaque étape une vache avec le résultat intermédiaire dans les yeux et change son comportement si un seuil est dépassé.

- Le script attend un argument initial suivi d'un ou plusieurs couples `<opérateur> <valeur>`.
- Il utilise `shift` pour traiter dynamiquement les arguments deux par deux.
- À chaque itération :
  - vérification de l'opérateur (`+`, `-`, `*`, `/`, `%`) ;
  - vérification que la valeur est bien un entier ;
  - tentative de calcul (`result $op val`) avec gestion d'erreur comme dans `smart_cow`.
- En cas de dépassement du seuil (`THRESHOLD=100`), la vache devient folle :
  - affichage d'un message de délire (`eyes = ??`, `@@`, etc.),
  - puis mort finale avec l'option `-d`.
- Si le résultat est :
  - négatif → yeux `XX`,
  - nul → yeux `??`,
  - normal → yeux `oo`.

Exemple d'exécution

```
$ ./crazy_cow.sh 5 + 2 - 100 + 3 \* 10
...
< Résultat après -93 + 3 : -90 >
        \   ^__^
         \  (XX)\_______

$ ./crazy_cow.sh 50 + 40 + 30
...
< Aaaaarg, c'en est trop ! Je meurs... >
        \   ^__^
         \  (xx)\_______


$ ./crazy_cow.sh 5 + 2 - 100 = + 3 "*" 10
...
< Opérateur invalide : = >
        \   ^__^
         \  (OO)\_______

$ ./crazy_cow.sh 5 + 2 - 100 + + 3 "*" 10
< Oups, impossible de calculer : -93 + + >
        \   ^__^
         \  (oo)\_______
```

Commentaires et difficultés

- Cette version visait à proposer un script plus original, avec un traitement dynamique des arguments via `shift` dans une boucle.
- Nécessité d'échapper l'astérisque `*` en ligne de commande (`\*` ou `"*"`), car sinon le shell tente de l'expanser (globbing) en cherchant les fichiers du répertoire courant.
- Vérification élémentaire des erreurs d'entrée via un sous-shell et redirection `2>/dev/null` pour ne pas afficher les messages Bash.

Le script complet est fourni dans l'archive (`scripts/crazy_cow.sh`).