INF203 - Partiel
Année 2024-2025

Durée : 2h.
Une feuille A4 manuscrite recto/verso autorisée, dictionnaire papier (non annoté) autorisé pour les étudiants étrangers uniquement.
Tout autre document, calculatrices et appareils électroniques interdits.
Pour chaque question, une partie des points peut tenir compte de la présentation.
Le barème est indicatif.

Toute réponse même incomplète sera valorisée à partir du moment où elle réalise au moins une partie de ce qui est demandé. Les questions sont relativement indépendantes, et dans tous les cas vous pouvez utiliser les scripts et les fonctions demandées dans les questions précédentes même si vous n'avez pas réussi à les écrire.

Par ailleurs, tout code ou algorithme *élégant*, en plus d'être correct, sera bonifié.

1 Programmation Shell : Oh la vache ! (13 points)

La commande `cowsay` permet d'afficher une image ASCII d'une vache disant ce que souhaite l'utilisateur. Par exemple, la commande suivante donne :

```
cowsay bonjour
 ---------
< bonjour >
 ---------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Cette commande peut prendre en option `-f nom_de_fichier` pour afficher un animal différent selon le fichier spécifié par son chemin absolu.

Une collection d'animaux sont fournis dans `/usr/share/cowsay/cows`. Par exemple :

```
cowsay -f /usr/share/cowsay/cows/duck.cow bonjour
 ---------
< bonjour >
 ---------
  \
   \
    \ >()_
        (__)__ _
```

Le but de cet exercice est d'écrire quelques scripts Shell pour aider l'utilisateur à connaître et utiliser les animaux disponibles.

On suppose que le répertoire `/usr/share/cowsay/cows` ne contient que des fichiers avec l'extension `.cow` et que ce sont tous des fichiers valides pour l'option `-f` de `cowsay`.

Questions

1. (1 point) Écrivez une commande ou une suite de commandes qui affiche le nombre d'animaux disponibles dans le répertoire `/usr/share/cowsay/cows`.

Solution:

```
ls /usr/share/cowsay/cows | wc -w
# ou wc -l car les fichiers sont de toute façon affichés ligne par ligne
```

Barème
- ½ : utilisation de wc avec au moins la bonne option
- ½ : utilisation correcte du pipe

2. (1 point) Certains fichiers de `/usr/share/cowsay/cows` portent simplement un nom d'animal, comme `duck.cow` donné en exemple plus haut. D'autres peuvent lister plusieurs animaux, comme par exemple `dragon-and-cow.cow` qui contient un dessin plus élaboré.
   Écrivez une commande ou une suite de commandes qui affiche la liste des fichiers du répertoire `/usr/share/cowsay/cows` dans lesquels figure au moins une vache (dont le nom contient la chaîne `cow` en dehors de l'extension).

Solution:

```
ls /usr/share/cowsay/cows/*cow*.cow
```

Attention à ne pas oublier de préciser l'extension sinon on affiche tout le contenu du répertoire ! `grep` est inopérant pour les mêmes raisons.

Barème
- ½ : utilisation du joker *
- ½ : le motif est suffisamment spécifique (par exemple *cow* ne convient pas)

3. (2 points) Les animaux listés dans le répertoire `/usr/share/cowsay/cows` ont été ajoutés au fur et à mesure du développement du logiciel `cowsay`.
   On peut connaître leur date de création à l'aide de la commande `stat`, par exemple :

```
stat -c %y /usr/share/cowsay/cows/sheep.cow
1999-08-14 09:17:58.000000000 +0200
```

   Écrivez un script `animalwho.sh` qui :
   — prend comme seul argument un nom d'animal `XXX` ;
   — calcule l'âge `AGE` de cet animal (à l'année près, on ne tiendra pas compte du mois ni du jour) ;
   — et fait dire à cet animal `Je suis un XXX qui a AGE ans`.
   Pour cette question on ne s'occupe pas de vérifier que l'animal `XXX` existe.

Solution:

```
#!/bin/bash
cow_file_path="/usr/share/cowsay/cows/"

animal=$cow_file_path$1.cow

year=$(stat -c %y $animal | cut -c 1-4)
age=$(expr 2025 - $year)

cowsay -f $animal "Je suis un $1 qui a $age ans"
```

Barème
- ½ : cut correct pour récupérer l'année
- ½ : expr pour calculer l'âge
- ½ : substitution de commande $() pour récupérer les résultats
- ½ : utilisation correcte des variables (ça m'étonnerait que quelqu'un écrive un one-liner avec les substitutions de commandes dedans directement)

4. (2 points) Écrivez un script `animallist.sh` qui affiche la liste des animaux disponibles dans `/usr/share/cowsay/cows/`, sans leurs chemins d'accès ni leurs extensions :

```
./animallist.sh
apt
bud-frogs
bunny
calvin
cheese
...
```

Solution:

```
#!/bin/bash
cow_file_path="/usr/share/cowsay/cows/"

for A in $cow_file_path/*
do
    basename $A .cow
done
```

Une solution plus courte mais plus experte (la regexp doit tenir compte des cas comme `dragon-and-cow`) :

```
ls /usr/share/cowsay/cows/ | sed s/\.cow$//
```

Barème

— 1 : présence d'un for et construction correcte de la boucle
— 1 : utilisation de basename avec les bons arguments

On peut donner 1 point sur les 2 si tentative imparfaite mais convaincante d'utilisation de sed.

5. (3 points) Écrivez un script `animalsay.sh` pour pouvoir utiliser cowsay avec des animaux différents plus facilement, sans avoir à préciser le chemin d'accès complet. Par exemple, obtenir le canard donné en exemple en première page pourra également se faire avec :

```
./animalsay.sh -a duck bonjour
```

Plus en détails, votre script devra :

— Si l'option `-a XXX` est donnée, faire appel à `cowsay` avec comme fichier d'animal `/usr/share/cowsay/cows/XXX.cow`.
  On suppose pour l'instant que tous les animaux `XXX` choisis existent.
— Si l'option `-r` est donnée, faire appel à `cowsay` en utilisant un animal choisi aléatoirement dans le répertoire `/usr/share/cowsay/cows`.
  (Pensez à consulter le rappel des commandes données en fin d'énoncé.)

Pour simplifier le traitement des options, on supposera qu'une seule option peut être donnée à notre script et que ce sera toujours le premier argument. Il admet donc exclusivement deux utilisations distinctes :

```
./animalsay.sh -a ANIMAL MESSAGE
./animalsay.sh -r MESSAGE
```

Solution: Remarque : cette solution utilise un traitement simpliste des options – on peut utiliser la commande `getopt` dans un cadre plus général, mais ce n'est pas le but de l'exercice ici et cela sort du cadre de ce cours.

```
#!/bin/bash
cow_file_path="/usr/share/cowsay/cows/"
```

```
if [ $# -ge 3 -a "$1" = "-a" ]
then
    cow_file=$2.cow
    message=$3
elif [ $# -ge 2 -a "$1" = "-r" ]
then
    cow_file=$(ls $cow_file_path | sort -R | head -n 1)
    message=$2
else
    echo "Option ou nombre d'arguments incorrect" > /dev/stderr
    exit 1
fi

cowsay -f $cow_file_path$cow_file $message
```

Barème
- ½ : variables automatiques $# et $1, $2...
- 1 : utilisation de if/then/else et structure correcte du code
- 1 : conditions booléennes : comparaisons appropriées, opérateur -a
- ½ : utilisation de sort et head pour choisir une vache au hasard

On ne pénalise pas si l'écriture sur la sortie d'erreur n'a pas été faite correctement.

6. (1 point) Pour améliorer le script précédent, on veut détecter si l'animal `XXX` donné en argument avec l'option `-a` existe réellement dans le répertoire `usr/share/cowsay/cows/`.

   Si le fichier correspondant n'existe pas :
   - afficher le message d'erreur `Animal XXX inexistant`, en utilisant la commande `echo` avec la redirection `> /dev/stderr`
   - et utiliser la vache par défaut (`/usr/share/cowsay/cows/default.cow`).

   Vous indiquerez quelles lignes vous ajoutez à votre script précédent et à quel(s) endroit(s).

Solution: On ajoute dans le premier cas du `if` les lignes

```
if [ ! -f $cow_file_path$cow_file ]
then
echo "Animal $2 inexistant" > /dev/stderr
cow_file="default.cow"
fi
```

Barème
- ½ : test correct sur l'existence du fichier
- ½ : insertion correcte dans le code précédent (par exemple avec une valeur par défaut pour la variable $cow_file)

7. (2 points) Voici le contenu des deux fichiers `.cow` donnés en exemple au début du sujet :

```
cat /usr/share/cowsay/cows/default.cow
          $thoughts   ^__^
         $thoughts  ($eyes)\\_______
            (__)\\       )\\/\\
             $tongue ||----w |
                ||     ||
cat /usr/share/cowsay/cows/duck.cow
   $thoughts
  $thoughts
   $thoughts >()_
      (__)__ _
```

On remarque la présence de certains mots-clés :
— `thoughts` qui permet à `cowsay` de positionner la bulle de texte
— `eyes` qui détermine la position des yeux de la vache lorsqu'on utilise l'option `-e`
— `tongue` qui permet de même de placer la langue avec l'option `-T`
En particulier, il n'est pas possible de modifier les yeux du canard puisque son fichier ne contient pas le mot-clé `eyes`.

Écrivez un script `animaleyes.sh` qui fait la liste des animaux du répertoire `/usr/share/cowsay/cows` dont on peut modifier les yeux.

Solution:

```
#!/bin/bash
cow_file_path="/usr/share/cowsay/cows/"

for A in $cow_file_path/*
do
    if grep eyes $A > /dev/null
    then
    echo $A
    fi
done
```

Remarque : `grep` dispose d'une option qui fait exactement ça :

```
grep -l eyes /usr/share/cowsay/cows/*
```

Barème
- ½ : utilisation de grep avec les bons arguments
- ½ : structure du code : if dans for
- 1 : récupération du résultat de grep (directement dans le if, ou avec un test sur $?, ou avec l'option -c, etc.)

On peut donner ½ point de bonus pour la suppression de la sortie de grep (/dev/null, -q, etc.).

8. (1 point) Les développeurs de `cowsay` essayent de recenser tous les animaux que les utilisateurs souhaient afficher mais qui ne sont pas encore disponibles.

Comment exécutez-vous le script `animalsay.sh` de façon à collecter tous les messages d'erreurs dans un unique fichier `vaches.log` (qui se complète à chaque nouvelle exécution) ?

Solution:

```
./animalsay.sh ... 2>> vaches.log
```

Barème
- 1 : utilisation de `2>>`
  (½ si `>>` ou `2>` est utilisé)

2 Programmation C : Calendrier perpétuel (7 points)

L'objectif de l'exercice est d'écrire un programme en C pour calculer la date qu'il sera dans n jours. Pour cela, on représentera la date par un tableau de 3 entiers (`int`) : jour, mois et année.

Questions

1. (1 point) Définissez un tableau de 12 entiers `jours_dans_mois` tel que le i-ème élément de `jours_dans_mois` contient le nombre de jours du mois i (dans une année non bissextile).

Solution:

```
int jours_dans_mois[12]={31,28,31,30,31,30,31,31,30,31,30,31};
```

Barème
- ½ : déclaration du tableau
- ½ : initialisation

2. (3 points) Écrivez une fonction `lendemain` qui prend une date (un tableau de 3 entiers) en argument et qui modifie ce tableau pour qu'il contienne la date du lendemain. La définition du tableau `jours_dans_mois` sera intégrée à cette fonction. Pour cela, on contrôlera si le jour actuel est le dernier du mois (et si c'est le cas, si le mois actuel est le dernier de l'année). Attention à l'indexation des tableaux qui débute par l'indice 0 (`jours_dans_mois[0]` correspond au nombre du jour du mois de janvier).

   On pourra pour le moment ne pas prendre compte des années bissextiles.

Solution:

```
 void lendemain(int * d){

     int jours_dans_mois[12]={31,28,31,30,31,30,31,31,30,31,30,31};

     int mois = d[1]-1;

     if (d[0]<jours_dans_mois[mois])
// ou déjà   if (d[0]<jours_dans_mois[mois] || (d[0]==28 && d[1]==2 && (d[2] % 4 == 0))
     d[0]++;
     else{

         d[0]=1;
         if (d[1]<12)
         d[1]++;
         else {
             d[2]++;
             d[1]=1;
         }
     }
 }
```

Barème
- — ½ : en-tête de la fonction
- — ½ : manipulation correcte du tableau reçu en argument
- — ½ : test sur la fin du mois
- — 1 : écriture et structure des if/else
- — ½ : remise à 1 du jour et/ou du mois en cas de dépassement du mois/de l'année

On ne pénalise pas si la définition de jours_dans_mois n'est pas réécrite.

3. (2 points) Écrivez une fonction `main` dans laquelle vous définirez un tableau `date` contenant les trois entiers de la date du jour de ce partiel ainsi qu'un entier `shift` qui contient le nombre de jours à incrémenter. On incrémentera alors `date` de `shift` jours et on imprimera le jour de fin sous le format :

`Dans n jours nous serons le dd/mm/yyyy.`

Solution:

```
int main(){

        int date[3]={28,2,2025};
        printf("Aujourd'hui, nous sommes le %d/%d/%d\n",date[0],date[1],date[2]);

        int shift = 2;
        for (int i=1;i<=shift;i++)
                lendemain(date);

        printf("Dans %d jours, nous serons le %d/%d/%d\n",shift,date[0],date[1],date[2])

        return 0;
}
```

Barème
- — ½ : toutes les variables doivent être déclarées
- — ½ : construction correcte de la boucle for
- — ½ : appel correct à la fonction lendemain
- — ½ : affichage correct avec printf

On ne pénalise pas l'initialisation de date (déjà évaluée à la première question).

4. (1 point) Comment faut-il modifier le plus simplement possible la fonction `lendemain` pour intégrer les 29 février des années bissextiles ?
   Si vous avez déjà traité ce cas plus haut, mentionner simplement "déja fait !".
   Rappel : L'opérateur % effectue l'opération "modulo" (reste de la division euclidienne).
   Les années bissextiles (ayant 29 jours en février) sont les années divisibles par 4 (du moins, on simplifiera ainsi, en supposant que nous ne chercherons pas des dates au-delà de l'année 2100).

Solution: La condition sur `d[0]` devient :

```
if (d[0]<jours_dans_mois[mois] || (d[0]==28 && d[1]==2 && (d[2] % 4 == 0)) )
```

Barème
- — ½ : utilisation de %
- — ½ : opérateurs booléens et parenthésage

A Annexe : rappels

On rappelle ici différentes commandes et options qui peuvent être utiles.

basename

`basename NOM` affiche le `NOM` de fichier donné en argument, en supprimant tous les répertoires du chemin donné (s'il y en a)

`basename NOM SUFFIXE` réalise la même opération, et de plus supprime le `SUFFIXE` donné s'il est présent à la fin du `NOM`

cut affiche une partie seulement de chaque ligne de ses entrées.

`-c debut-fin` permet de choisir les caractères affichés ;

`-f debut-fin` permet de choisir des colonnes ;

`-d` permet de choisir le délimiteur.

diff détermine si les contenus de 2 fichiers sont identiques :

— aucune réponse : les 2 fichiers sont identiques.

— la réponse est seulement que les 2 fichiers sont différents : l'un au moins n'est pas un fichier texte.

— la liste des différences entre les 2 fichiers : les 2 fichiers sont des fichiers texte.

expr calcule l'expression décrite par ses arguments.

grep n'affiche que les lignes d'un fichier contenant une chaîne de caractères donnée.

`-c` n'affiche que le nombre total de lignes contenant la chaîne recherchée.

`-v` n'affiche que les lignes ne contenant pas la chaîne donnée.

head et tail

`head` affiche le début de ses entrées, selon l'option choisie :

`-n N :` les `N` premières lignes

`-n -N :` toutes les lignes sauf les `N` dernières

De même la commande `tail` affiche la fin de ses entrées :

`-n N :` les `N` dernières lignes

`-n +N :` les dernières lignes à partir de la N^e

sed affiche les lignes de ses entrées en les filtrant et/ou modifiant selon les instructions données :

`sed /Candide/d` affiche celles ne contenant pas `Candide`

`sed s/Candide/Toto/` remplace le premier `Candide` par `Toto` sur chaque ligne

`sed s/Candide/Toto/g` remplace tout les `Candide` par `Toto`

sort affiche les lignes de ses entrées triées par ordre alphabétique croissant

`sort -n` affiche les lignes triées selon l'ordre numérique

`sort -R` trie les lignes de son entrée selon un ordre aléatoire

tr copie l'entrée standard sur la sortie standard en substituant

`tr chaîne1 chaîne2 :` chaque caractère présent dans `chaîne1` par le caractère de position correspondante dans `chaîne2`

`tr -s chaîne :` les répétitions des caractères présents dans `chaîne` par une occurrence unique

wc compte les caractères, les mots ou les lignes lus sur son entrée standard :

`-c` compte les caractères

`-w` compte les mots

`-l` compte les lignes