TP3_INF203

Groupe: INF2

Binôme: Shaghayegh HAJMOHAMMADKASHI

Kaiwen ZHENG

EXO1 :

Terminal : cd INF203

mkdir TP2

./TP1/scripts/installeTP.sh 3

le repertoire courant est

/home/h/hajmohas/INF203/TP3

il contient

total 24

-rwxr-xr-x 1 hajmohas ima-nogroup 247 18 févr. 16:00 description.sh

-rwxr-xr-x 1 hajmohas ima-nogroup 553 18 févr. 16:00 instant_suivant.sh

-rwxr-xr-x 1 hajmohas ima-nogroup 186 18 févr. 16:00 max2err.sh

-rwxr-xr-x 1 hajmohas ima-nogroup 178 18 févr. 16:00 max2.sh

-rwxr-xr-x 1 hajmohas ima-nogroup 181 18 févr. 16:00 prefixe.sh

-rwxr-xr-x 1 hajmohas ima-nogroup 198 18 févr. 16:00 test_elem.sh

cd TP3

[a]:

Les deux programmes ont pour objectif de déterminer le plus grand des deux nombres saisis. Cependant, ils diffèrent dans la manière dont les nombres sont saisis. Dans le premier programme (max2.sh de TP3), les deux nombres sont passés en tant qu'arguments du script, alors que dans le second programme (max2.sh de TP2), les nombres sont saisis à l'aide de la commande "read".

Le premier programme vérifie si le premier nombre est plus grand que le deuxième nombre. Si c'est le cas, il affiche que le premier nombre est plus grand. Si le premier nombre est plus petit que le deuxième nombre, il affiche que le deuxième nombre est plus grand. Si les deux nombres sont égaux, il affiche que les deux nombres sont égaux.

Le second programme compare simplement les deux nombres saisis à l'aide d'une instruction "if" et stocke le plus grand nombre dans une variable "max", qu'il affiche ensuite.

Pour vérifier que le programme est "correct" dans tous les cas de figure possibles, il est conseillé de tester les différentes situations suivantes :

1.Saisir deux nombres égaux : dans ce cas, les deux programmes doivent afficher que les deux nombres sont égaux.

hajmohas@im2ag-turing-01:[~/INF203/TP3]: ./max2.sh 5 5

les deux entiers 5 et 5 sont égaux

hajmohas@im2ag-turing-01:[~/INF203/TP2]: ./max2.sh

Saisissez deux entiers

5

5

Le plus grand des deux entiers 5 et 5 est 5

2.Saisir un premier nombre plus grand que le deuxième : dans ce cas, les deux programmes doivent afficher que le premier nombre est plus grand.

hajmohas@im2ag-turing-01:[~/INF203/TP3]: ./max2.sh 90 36

90 est plus grand que 36

hajmohas@im2ag-turing-01:[~/INF203/TP2]: ./max2.sh

Saisissez deux entiers

90

36

Le plus grand des deux entiers 90 et 36 est 90

3.Saisir un deuxième nombre plus grand que le premier : dans ce cas, les deux programmes doivent afficher que le deuxième nombre est plus grand.

hajmohas@im2ag-turing-01:[~/INF203/TP3]: ./max2.sh 2 4

4 est plus grand que 2

hajmohas@im2ag-turing-01:[~/INF203/TP2]: ./max2.sh

Saisissez deux entiers

2

4

Le plus grand des deux entiers 2 et 4 est 4

4.Saisir des nombres non entiers : dans ce cas, les deux programmes fonctionnes mal et afficheront son partie else.

hajmohas@im2ag-turing-01:[~/INF203/TP3]: ./max2.sh 2.4 5.8

./max2.sh: ligne 3 : [: 2.4 : nombre entier attendu comme expression

./max2.sh: ligne 6 : [: 2.4 : nombre entier attendu comme expression

les deux entiers 2.4 et 5.8 sont égaux

hajmohas@im2ag-turing-01:[~/INF203/TP2]: ./max2.sh

Saisissez deux entiers

5.8

2.4

./max2.sh: ligne 5 : [: 5.8 : nombre entier attendu comme expression

Le plus grand des deux entiers 5.8 et 2.4 est 2.4

5.Saisir moins de deux nombres : dans ce cas, les deux programmes fonctionnes mal et afficheront son partie else..

hajmohas@im2ag-turing-01:[~/INF203/TP3]: ./max2.sh 3

./max2.sh: ligne 3 : [: 3 : opérateur unaire attendu

./max2.sh: ligne 6 : [: 3 : opérateur unaire attendu

les deux entiers 3 et sont égaux

hajmohas@im2ag-turing-01:[~/INF203/TP2]: ./max2.sh

Saisissez deux entiers

3

./max2.sh: ligne 5 : [: 3 : opérateur unaire attendu

Le plus grand des deux entiers 3 et est

[b] :

Terminal :./max2err.sh 4 2

4 est plus grand que 2

./max2err.sh 2 4

./max2err.sh: ligne 6 : [: -ploumploum : opérateur binaire attendu

les deux entiers 2 et 4 sont egaux

./max2err.sh 5 5

./max2err.sh: ligne 6 : [: -ploumploum : opérateur binaire attendu

les deux entiers 5 et 5 sont egaux

Dans le programme max2err.sh, une erreur a été commise à la ligne 6 où l'opérateur binaire "ploumploum" est utilisé au lieu de l'opérateur "gt" pour vérifier si le deuxième argument est plus grand que le premier.

Cependant, le programme fonctionne bien pour certaines configurations des données, comme lorsque les deux arguments sont égaux, car dans ce cas l'exécution entre dans la condition "else" qui affiche que les deux entiers sont égaux ou quand on saisit un premier nombre plus grand que le deuxième.

Cela a un rapport avec la terminologie que nous utilisons pour désigner le shell ou l'interpréteur de commandes. Lorsque l'on exécute le script avec la commande "./max2err.sh 2 4", le shell va interpréter les instructions du script et tenter de les exécuter. Dans ce cas, une erreur de syntaxe est détectée dans le script, mais le shell continue de fonctionner et d'exécuter les autres instructions qui sont correctes. C'est un comportement typique d'un interpréteur de commandes qui va exécuter les commandes tant qu'elles sont valides et va s'arrêter à la première erreur détectée.

[c] :

```
#!/bin/bash

if [ $# -lt 2 ]
then
    echo "Erreur : ce script nécessite deux arguments entiers."
    exit 1
fi

if [ $1 -gt $2 ]
then
    echo $1 est plus grand que $2
elif [ $1 -lt $2 ]
then
    echo $2 est plus grand que $1
else
    echo les deux entiers $1 et $2 sont egaux
fi
```

```
#!/bin/bash

if [ -d "TP$1" ]; then
    echo "Erreur : le répertoire TP$1 existe déjà dans le répertoire courant."
    exit 1
fi

cd $HOME/INF203
cp -r /Public/203_INF_Public/TP$1 .
cd TP$1
echo le repertoire courant est
pwd
echo il contient
ls -l
```

[d] :

```bash
#!/bin/bash

# création du répertoire Exec s'il n'existe pas déjà
if [ ! -d "Exec" ]; then
    mkdir "Exec"
fi

for FILE in *
do
    if  [ -f $FILE ] && [ -x $FILE ] && [ "$FILE" != "description.sh" ]
    then
        # déplacer les fichiers exécutables dans le répertoire Exec
        mv "$FILE" "Exec"
        echo "$FILE est un fichier executable et a été déplacé dans le
répertoire Exec"
    elif [ -f $FILE ]
    then
        echo "$FILE est un fichier non executable"
    elif [ -d $FILE ]
    then
        echo $FILE est un repertoire
    fi
done
```

[e] :

```bash
#!/bin/bash

i=1
intervalle=2

while [ $i -le 10 ]
do
    heure=$(date +"%H:%M:%S")
    echo "Heure: $heure"
    sleep $intervalle
    intervalle=$((intervalle+2))
    i=$((i+1))
done
```

./des_heures.sh < instant_suivant10.sh

Heure: 19:09:52

Heure: 19:09:54

Heure: 19:09:58

Heure: 19:10:04

Heure: 19:10:12

Heure: 19:10:22

Heure: 19:10:34

Heure: 19:10:48

Heure: 19:11:04

Heure: 19:11:22

[f] :

```
#!/bin/bash

for ((i=1; i<=$1; i++))
do
    date +"Heure: %H, Minute: %M, Seconde: %S"
    sleep $2
done
```

hajmohas@im2ag-turing-01:[~/INF203/TP3]: ./des_heures.sh 4 2

Heure: 18, Minute: 48, Seconde: 32

Heure: 18, Minute: 48, Seconde: 34

Heure: 18, Minute: 48, Seconde: 36

Heure: 18, Minute: 48, Seconde: 38

[g] :

Terminal : ./des_heures.sh 5 2 | ./instant_suivant10.sh

Heure: Heure, Minute: 19, Seconde: 51

expr: argument non entier

heuresuivbrute: , minutesuivbrute: , secondesuivbrute:

expr: erreur de syntaxe: argument « 24 » inattendu

:19:52

Heure: Heure, Minute: 19, Seconde: 51

expr: argument non entier

heuresuivbrute: , minutesuivbrute: 19, secondesuivbrute: 52

expr: erreur de syntaxe: argument « 24 » inattendu

:19:52

Heure: Heure, Minute: 19, Seconde: 51

expr: argument non entier

heuresuivbrute: , minutesuivbrute: 19, secondesuivbrute: 52

expr: erreur de syntaxe: argument « 24 » inattendu

:19:52

Heure: Heure, Minute: 19, Seconde: 51

expr: argument non entier

heuresuivbrute: , minutesuivbrute: 19, secondesuivbrute: 52

expr: erreur de syntaxe: argument « 24 » inattendu

:19:52

Heure: Heure, Minute: 19, Seconde: 51

expr: argument non entier

heuresuivbrute: , minutesuivbrute: 19, secondesuivbrute: 52

expr: erreur de syntaxe: argument « 24 » inattendu

:19:52

Heure: Heure, Minute: 19, Seconde: 51

expr: argument non entier

heuresuivbrute: , minutesuivbrute: 19, secondesuivbrute: 52

expr: erreur de syntaxe: argument « 24 » inattendu

:19:52

Heure: Heure, Minute: 19, Seconde: 51

expr: argument non entier

heuresuivbrute: , minutesuivbrute: 19, secondesuivbrute: 52

expr: erreur de syntaxe: argument « 24 » inattendu

:19:52

Heure: Heure, Minute: 19, Seconde: 52

expr: argument non entier

heuresuivbrute: , minutesuivbrute: 19, secondesuivbrute: 52

expr: erreur de syntaxe: argument « 24 » inattendu

:19:53

Heure: Heure, Minute: 19, Seconde: 52

expr: argument non entier

heuresuivbrute: , minutesuivbrute: 19, secondesuivbrute: 53

expr: erreur de syntaxe: argument « 24 » inattendu

:19:53

Heure: Heure, Minute: 19, Seconde: 52

expr: argument non entier

heuresuivbrute: , minutesuivbrute: 19, secondesuivbrute: 53

expr: erreur de syntaxe: argument « 24 » inattendu

:19:53

./instant_suivant10.sh < ./des_heures.sh

Cette commande tente de rediriger l'entrée standard du script instant_suivant10.sh à partir du script des_heures.sh avec des arguments 5 et 2.

Cela ne fonctionnera pas correctement car instant_suivant10.sh ne lit pas directement les arguments passés à des_heures.sh, mais lit plutôt à partir de l'entrée standard. De plus, des_heures.sh ne produit pas de sortie que instant_suivant10.sh peut utiliser comme entrée

[h] :

```
#!/bin/bash

for f in *.entree
do
    file_name=$(basename $f .entree)
    ./instant_suivant.sh < $file_name.entree > $file_name.ma_sortie
    diff $file_name.sortie $file_name.ma_sortie
    if [ $? -eq 0 ]
    then
        echo tout va bien pour le test $X
    else
        echo tout va mal pour le test $X
    fi
done
```

hajmohas@im2ag-turing-01:[~/INF203/TP3/TEST_INSTANT_SUIVANT]: ./instant_suivant.sh < 144725.entree > 144725.ma_sortie

hajmohas@im2ag-turing-01:[~/INF203/TP3/TEST_INSTANT_SUIVANT]: diff 144725.sortie 144725.ma_sortie

hajmohas@im2ag-turing-01:[~/INF203/TP3/TEST_INSTANT_SUIVANT]: echo $?

0

hajmohas@im2ag-turing-01:[~/INF203/TP3/TEST_INSTANT_SUIVANT]: ./test_instant.sh

tout va bien pour le test

tout va bien pour le test