Exercice n°1

Taper les commandes suivantes ligne par ligne sans les mettre dans un script et regarder à chaque fois les résultats. Bash peut à la fois être programmé par des scripts et au clavier.

```
echo Salut a toi
echo Salut $LOGNAME dont le compte est dans $HOME
echo Tu fais ce TP dans $PWD, quel beau dossier '!'
```

$*mot* est compris par bash comme étant à remplacer par la valeur de la variable *mot*. Les deux variables sont prédéfinies et gérées automatiquement par bash. Attention, A est différent de a.

```
nom=Yakari
animal=Aigle
taille=Grand
echo le totem de $nom est $taille $animal
```

Attention : une affectation se fait par variable=valeur sans aucun espace.

```
totem=$taille$animal # sans espace pour commencer
echo le totem de $nom est $totem
totem=$taille $animal
echo le totem de $nom est $totem
totem=$taille_$animal_noir
echo le totem de $nom est $totem
totem=${taille}_${animal}_noir
echo le totem de $nom est $totem
totem='$taille $animal noir'
echo le totem de $nom est $totem
totem="$taille $animal noir"
echo le totem de $nom est $totem
echo 'le totem de $nom est $totem'
echo "le totem de $nom est $totem"
echo 'le totem de '$nom" est $totem" # on peut juxtaposer des chaînes
```

Bash n'accepte plusieurs mots que s'ils sont encadrés par des délimiteurs de chaînes. Les délimiteurs de chaînes ' et " sont spécialisés, chacun son rôle.
On peut juxtaposer des chaînes délimitées différemment. Si on colle un mot juste après le nom d'une variable, bash ne peut pas savoir où s'arrête la variable.
On doit mettre des {} pour isoler le nom de la variable du reste :

```
totem=${taille}${animal}noir ou ${taille}_${animal}_noir
echo mon totem est $totem
```

Exercice n°2

Corriger les lignes suivantes pour qu'elles affichent les textes sans faire d'erreurs avec hero et totem remplacés par leurs valeurs mais en laissant $USD tel quel, et avec les " ' et ! affichés à l'écran :

```
hero=Yakari
echo l'animal fetiche de $hero est $totem, il vaut 3000€, c'est cher !
echo "vole, petit $totem !" lui-dit $hero
```

D'autres exemples à essayer toujours au clavier :

```
nombre=`ls $HOME | wc -l` # syntaxe obsolète
echo "il y a $nombre éléments (fichiers ou dossiers) dans $HOME"

nombre=$(ls $HOME | wc -l) # syntaxe recommandée
echo il y a toujours $nombre éléments dans $HOME
```

Les délimiteurs `...` ou $(...) doivent encadrer une commande Unix ; ce qu'elle affiche est redirigé dans la variable. Il est recommandé d'employer la syntaxe $( ).
Compléter les lignes suivantes pour qu'elles affichent le nombre total de processus qui tournent actuellement sur la machine :

```
nombre= ... à vous de mettre ce qu'il faut...
echo il y a $nombre processus sur $HOSTNAME
```

Voici comment on fait des calculs *entiers* en bash :

```
x=3
y=$(( 11 - (-x * -7) ))
echo $x $y $(( y / x )) $((-y % x))
```

Attention à ne pas confondre les syntaxes $(commande) et $((expression)).

Exercice n°3
Ecrire un script bash. Ce sont des fichiers texte non compilés mais exécutables, qui contiennent des commandes Unix et des instructions Bash.
Il faut placer #!/bin/bash dans la toute première ligne du fichier et rajouter le droit d'exécution dessus.

Ouvrir le terminal et lancer la commande nano exo1.sh. Taper les lignes suivantes dedans et enregistrer.

```
#!/bin/bash
echo mon premier script bash fonctionne parfaitement du premier coup
```

Ensuite rajouter le droit d'exécution par `chmod u+x exo1.sh`. Puis lancer le script en tapant `sh exo1.sh`.

Essayer maintenant le script suivant :

```
#!/bin/bash
echo 'Quel est ton prénom ? '
read prenom
echo "Bonjour, cher(e) $prenom, comment vas-tu ? "
read etat
echo ah, tiens moi aussi, je vais $etat
```

Essayer avec différentes valeurs y compris des réponses vides ou très longues...
Maintenant, on rajoute un test pour refuser les réponses vides :

```
#!/bin/bash
read -p 'quel est ton prénom : ' prenom
if test "$prenom" = ''
then
      echo "désolé, je ne parle pas aux inconnus."
      exit 1 # sortie immédiate du programme : pas besoin de else
fi
```

L'option -p de la commande read permet d'afficher la question et de lire la réponse en une seule commande.
Les " " autour des variables et les ' ' qui créent une chaîne vide sont nécessaires puisque ces variables peuvent contenir aucun ou plusieurs mots. Grâce à ces délimiteurs, les mots restent ensemble.

Modifier le script pour qu'il vérifie aussi que l'état n'est pas vide.

Exercice n°4

Créer un script exo2.sh. Lancer ce script et tester tous les cas (fichier et dossier existant, absent...).

```
#!/bin/bash
read -p 'donnez un nom de fichier ou de répertoire : ' nomfich
if test -f "$nomfich"
then
      echo "$nomfich" est un fichier
else
      if test -d "$nomfich"
      then
            echo "$nomfich" est un dossier
      else
            echo "je ne vois ni fichier ni dossier qui s'appelle $nomfich"
      fi
fi
```

Tester le script suivant en le lançant avec ou sans paramètres sur la ligne de commande :

```
#!/bin/bash
if test $# -eq 0
then
      echo il faudrait fournir un paramètre sur la ligne de commande
      echo par exemple, lancez : $0 bonjour mon beau script
      exit 1
fi
echo le premier paramètre est : $1
if test "$2" = ''
then
      echo "il n'y a pas d'autres paramètres"
else
      echo vous avez aussi fourni : $2
fi
echo vous avez fourni $# paramètres en tout, voici la liste : $@
```

Les paramètres sont le moyen normal de spécifier les objets (fichiers, utilisateurs...) traités par le script.
$0 est le nom du script lui-même, $# est le nombre de paramètres fournis.

Exercice n°5

Créer un script exo3.sh.

```
#!/bin/bash
echo "Il faut deviner le nombre auquel je pense."
while read -p "quel nombre proposez-vous ? " proposition
      do
      if test "$proposition" -eq $$
      then
            echo "Bravo"
            break
      else
            echo "Perdu"
      fi
done
echo "fin du jeu"
```

La comparaison entre la proposition et la bonne réponse, $$ est faite par -eq car ce sont des nombres. Quand ce sont des chaînes, on les compare par = ou !=. Il y a d'autres comparateurs, comme -lt, -ge.
Le nombre caché est $$, c'est à dire le numéro de processus de ce script (qui change à chaque lancement). Utiliser ps pour le connaître (mettre le jeu en arrière-plan, le temps de regarder son PID).

Exercice n°6

Créer un script exo4.sh avec une boucle for :

```
#!/bin/bash
echo "Voici ce que je vois dans ce dossier :"
for nom in *
do
      if test -f "$nom"
      then
            if test ${nom#*.} = 'sh'
            then
                  echo "Il y a $nom et c'est probablement un script bash"
            else
                  echo "Il y a le fichier $nom"
            fi
      else
            echo "Il y a $nom mais ce n'est pas un fichier"
      fi
done
```

La boucle for parcourt tous les fichiers et dossiers du répertoire courant (ceux qui sont désignés par *). Pour chacun, il y a un premier test qui signifie : est-ce un fichier ? Et il y a un second test qui signifie : est-ce que la fin de son nom est « sh » ?
La notation ${variable#motif} extrait le reste de la variable si son début correspond au motif.

Exercice n°7

Pour programmer une nouvelle commande très utile pour les utilisateurs : pouvoir supprimer un fichier comme la commande rm mais avec la possibilité de le récupérer dans une corbeille. Le principe est de déplacer le fichier à supprimer dans un répertoire ~/corbeille au lieu de le supprimer. Cette corbeille est dans le compte, donc facilement accessible de partout. Vider la corbeille = simplement supprimer ce dossier.

Voici l'algorithme du script « supprim » à programmer :

vérifier qu'il y a un paramètre, sinon : erreur
vérifier que ce paramètre est un fichier, sinon : erreur
vérifier que le répertoire ~/corbeille existe, sinon le créer
déplacer le fichier dans ~/corbeille

Créer ce script et vérifier qu'il marche correctement en le testant sur plusieurs fichiers.

Exercice n°8

Écrire maintenant un script appelé recup qui remet un fichier effacé avec supprim. Voici son algorithme.

vérifier qu'il y a un paramètre, sinon : erreur
vérifier que ~/corbeille/ *le paramètre* est un fichier, sinon erreur : fichier pas trouvé
déplacer le fichier de ~/corbeille vers le répertoire courant