Partie 2 : Composition d'un système d'exploitation

C3

TD : bash sous Linux (branche Debian)

Pour ces exercices, nous utiliserons **deux outils** :

- **Le terminal** par défaut du système d'exploitation (xterm, gnome-terminal...)
- L'éditeur de texte **Gedit**

Partie 1 : Quelques commandes de bases

1. Ouvrir un terminal avec la combinaison de touches **ctrl+alt+T**
2. **Taper** `whoami` pour connaître votre nom d'utilisateur.
3. **Taper** `pwd` pour identifier votre position dans le système de fichier.
4. **Taper** `ls` pour connaître le contenu du répertoire dans lequel vous êtes.
5. **Taper** `ls -l` pour voir le contenu non-masqué, ses droits, son appartenance user:groupe, son poids en kb, sa date de dernière modification et son nom.
6. **Taper** `ll` pour voir le contenu global du répertoire, répertoire parent et fichiers cachés inclus.
7. **Créer un répertoire** et le nommer **en tapant** `mkdir arriere_grand_parent`

   > Notez que nous évitons d'ajouter l'accent obligatoire du mot « arrière » pour assurer la compatibilité. De même que nous évitons les espace entre les termes en les remplaçant par des "underscores", le fameux "tiret du 8". Autre exemple : mot_a_separer..

8. **Taper** `ls` pour voir le contenu global du répertoire de votre position et apercevoir votre dossier.
9. **Taper** `rmdir arriere_grand_parent` afin de supprimer le dossier.

`whoami`	**Connaître le nom d'utilisateur**
`pwd`	**Identifier la position dans un système de fichier**
`ls`	**Lister le contenu d'un répertoire**

`whoami`	**Connaître le nom d'utilisateur**
`ls -l`	**Lister le contenu non-masqué d'un répertoire et voir le détail**
`ll`	**Lister l'intégralité du contenu d'un répertoire et voir le détail**
`mkdir`	**Créer un répertoire**
`rmdir`	**Supprimer un répertoire**

Partie 2 : Automatiser les commandes via un script

10. Utilisons chaque commande l'une après l'autre en créant un script. Celui-ci sera exécuté de manière séquentielle, soit séquence après séquence. Pour ce faire, nous utiliserons un éditeur de texte standard. Nous devons mettre à jour la liste des paquets disponibles à la mise à jour et **nous installons l'éditeur GEDIT en tapant** `sudo apt update && apt install gedit -y`
11. Une fois **GEDIT** installé, **nous ouvrons un nouveau document en tapant** `sudo gedit script01.sh` et nous écrivons **le début de notre script :**

```
#!/bin/bash                             # On va chercher l'utilitaire bash
printf "\e[8;33;80t"                    # On formate le terminal
```

> **Il sera nécessaire d'enregistrer le script à chaque modification** avant de l'exécuter. De plus, l'exécution doit se faire dans un nouvel onglet ou une nouvelle fenêtre de terminal. Pour enregistrer vos modifications, préférez la combinaison de touches **Ctrl+S**

12. Nous pouvons ajouter du texte personnalisé afin de marquer la progression de notre programme :

```
echo ""                                 # On saute une ligne
echo "   Début de mon script   "        # On affiche un message
echo ""                                 # On saute une ligne
sleep 2                                 # On marque une pause
```

13. Nous pouvons à présent combiner nos méthodes :

```
#!/bin/bash
printf "\e[8;33;80t"

echo ""
echo " Début de mon script   "
echo ""
sleep 10
echo " Le nom d'utilisateur est : "
echo
whoami
echo ""
sleep 10
echo " Je suis dans le répertoire : "
echo ""
pwd
```

```
echo
sleep 20
```

14. Nous pouvons enregistrer notre avancée grâce à la combinaison de touches `Ctrl+S` , le document sera enregistré sous le nom `script01.sh`.
15. Nous devons passer les droits d'exécution à notre document `script01.sh` en ouvrant un nouveau terminal grâce à la combinaison de touches `Ctrl+Alt+T` et en tapant dedans :

```
sudo chmod +x script01.sh
```

16. Il ne reste plus qu'à tester notre programme via la commande :

```
sudo ./script01.sh
```

Le résultat doit être celui-ci. Si ce n'est pas le cas, veuillez reprendre la lecture de ce TD du début et vérifier que vous avez correctement appliqué chaque étape de manière rigoureuse.

Partie 3 : Optimisation d'un script via les variables

17. Nous allons transformer les occurrences dans le code en variables afin de réduire le nombre de caractères. Nous avons deux occurrences de `sleep` dans le code. Afin de transformé `sleep` en variable, il nous suffit de procéder de la même manière que pour `echo` qui devient `$e` :

```
#!/bin/bash
printf "\e[8;33;80t"
e='echo'
s='sleep'                               # On transforme sleep en $s

$e
$e " Début de mon script   "
$e
$s 10                                   # On utilise notre nouvelle variable $s
$e " Le nom d'utilisateur est : "
$e
whoami
$e
$s 10                                   # On réutilise notre nouvelle variable $s
$e " Je suis dans le répertoire : "
$e
pwd
$e
$s 20                                   # Nous allons modifier son unité de
mesure
```

La valeur qui suit `$s` n'est pas suivie d'une unité de mesure.
Par défaut, celui-ci est en secondes.
Voici la liste complète des unités à disposition :

- **s** : définit l'intervalle de temps en secondes
- **m** : définit l'intervalle de temps en minutes

- **h** : définit l'intervalle de temps en heures
- **d** : définit l'intervalle de temps en jours

18. Dans notre cas, nous pouvons ajouter l'unité `m` à notre valeur `20` dans la dernière ligne de notre script :

```
$s 20m
```

> *Si tout fonctionne correctement, **le terminal ne se fermera pas avant que 20 minutes se soient écoulées**. Pour tuer le processus en cours, il est possible d'utiliser **Ctrl+C** mais vous pouvez tout aussi bien **fermer le terminal** par son bouton de fermeture.*

Il sera nécessaire d'adapter l'unité de mesure ou la valeur afin qu'elle correspondent aux besoins. Dans notre cas, nous resterons en secondes et nous supprimons donc le `m` ainsi :

```
$s 20
```

19. Nous allons à présent enrichir notre code. Nos objectifs sont d'obtenir :
    - Un code lisible.
    - Un programme fonctionnel.
    - Un programme plus agréable à lire et voir s'animer sous nos yeux.

Pour cela, nous définissons 5 nouvelles variables en haut de notre script :

```
#!/bin/bash
###### AFFICHAGE #################################################
printf "\e[8;33;80t"
###### VARIABLES #################################################
e='echo'                # Permet d'écrire une ligne vide
s='sleep'               # Commande de pause
bt='tput setaf 7'       # Passer le texte en blanc
nt='tput setaf 0'       # Passer le texte en noir
b='tput setab 7'        # Passer la couleur de fond en blanc
n='tput setab 0'        # Passer la couleur de fond en noir
rm='tput sgr0'          # Permet de réinitialiser les couleurs
```

20. Par la suite, nous utilisons nos variables dans le script :

```
######## SCRIPT ##################################################
$e
$e "- $($bt)$($n) Début de mon script $($rm) -"
```

21. Nous enrichissons aussi notre script afin qu'il nous retourne la commande qui vient d'être utilisée :

```
######## SCRIPT ##################################################
$e
$e "- $($bt)$($n) Début de mon script $($rm) -"
$e
$s 2
$e "- Le nom d'utilisateur est : "
```

```
$e
$s 2
whoami
$e
$s 2
$e " $($nt)$($b) La commande était --> $($rm)$($bt)$($n) whoami $($rm) "
$e
$s 10
```

Voici ce que donne nos modifications :


Capture d’écran du 2023-12-06 17-45-52

22. Nous modifions le reste de notre script afin que nos modifications soient utilisées dans l'ensemble :

```
#!/bin/bash
###### AFFICHAGE #################################################
printf "\e[8;33;80t"
###### VARIABLES #################################################
e='echo'                # Permet d'écrire une ligne vide
s='sleep'               # Commande de pause
bt='tput setaf 7'       # Passer le texte en blanc
nt='tput setaf 0'       # Passer le texte en noir
b='tput setab 7'        # Passer la couleur de fond en blanc
n='tput setab 0'        # Passer la couleur de fond en noir
rm='tput sgr0'          # Permet de réinitialiser les couleurs
######## SCRIPT ##################################################
$e
$e "- $($bt)$($n) Début de mon script $($rm) -"
$e
$s 2
$e "- Le nom d'utilisateur est : "
$e
$s 2
whoami                  # Exécution de la commande whoami
$e
$s 2
$e " $($nt)$($b) La commande était --> $($rm)$($bt)$($n) whoami $($rm) "
$e
$s 2
$e "- Je suis dans le répertoire : "
$e
$s 2
pwd                     # Exécution de la commande pwd
$e
$s 2
$e " $($nt)$($b) La commande était --> $($rm)$($bt)$($n) pwd $($rm) "
$e
$s 10
```

Partie 4 : Ajout de l'ensemble des commandes et création d'un jeu

23. Nous pouvons ensuite ajouter l'ensemble de nos commandes en appliquant les variables définies :

```
#!/bin/bash
###### AFFICHAGE #################################################
printf "\e[8;33;80t"
###### VARIABLES #################################################
e='echo'                # Permet d'écrire une ligne vide
s='sleep'               # Commande de pause
bt='tput setaf 7'       # Passer le texte en blanc
nt='tput setaf 0'       # Passer le texte en noir
b='tput setab 7'        # Passer la couleur de fond en blanc
n='tput setab 0'        # Passer la couleur de fond en noir
rm='tput sgr0'          # Permet de réinitialiser les couleurs
######## SCRIPT ##################################################
$e
$e "- $($bt)$($n) Début de mon script $($rm) -"
$e
$s 2
$e "- Le nom d'utilisateur est : "
$e
$s 2
whoami                                          # Exécution de la commande whoami
$e
$s 2
$e " $($nt)$($b) La commande était --> $($rm)$($bt)$($n) whoami $($rm) "
$e
$s 2
$e "- Je suis dans le répertoire : "
$e
$s 2
pwd                                             # Exécution de la commande pwd
$e
$s 2
$e " $($nt)$($b) La commande était --> $($rm)$($bt)$($n) pwd $($rm) "
$e
$s 2
$e "- Le contenu du répertoire courant : "
$e
$s 2
ls                                              # Exécution de la commande ls
$e
$s 2
$e " $($nt)$($b) La commande était --> $($rm)$($bt)$($n) ls $($rm) "
$e
$s 10
$e "- Le contenu non masqué du répertoire courant : "
$e
$s 2
```

```
$e "$($nt)$($b)   Droits    Appartenance     Poids    Date         Nom
             $($rm)"
ls -l                                          # Exécution de la commande ls
-l
$e
$s 2
$e " $($nt)$($b) La commande était --> $($rm)$($bt)$($n) ls-l $($rm) "
$e
$s 10
$e "- Création du répertoire arriere_grand_parent : "
$e
$s 2
mkdir arriere_grand_parent                     # Exécution de la commande
mkdir
$e
$s 2
$e " $($nt)$($b) La commande était --> $($rm)$($bt)$($n) mkdir $($rm) "
$e
$s 10
$e "- Le contenu du répertoire courant : "
$e
$s 2
ls                                             # Exécution de la commande ls
$e
$s 2
$e " $($nt)$($b) La commande était --> $($rm)$($bt)$($n) ls $($rm) "
$e
$s 10
$e "- Suppression du répertoire arriere_grand_parent : "
$e
$s 2
rmdir arriere_grand_parent                     # Exécution de la commande
rmdir
$e
$s 2
$e " $($nt)$($b) La commande était --> $($rm)$($bt)$($n) mkdir $($rm) "
$e
$s 10
$e "- Le contenu du répertoire courant : "
$e
$s 2
ls                                             # Exécution de la commande ls
$e
$s 2
$e " $($nt)$($b) La commande était --> $($rm)$($bt)$($n) ls $($rm) "
$e
$s 10
```

24. Nous allons créer un nouveau script pour développer une fonction de question/réponse dans l'objectif de faire un jeu. Notre jeu permettra :

- De poser une question.
- De proposer d'y répondre.
- De comparer la réponse entrée à la réponse attendue.

- De féliciter l'utilisateur s'il entre la bonne réponse.
- De l'aider à la trouver s'il échoue.

25. Commençons par créer notre nouveau script en tapant dans un terminal :

```
sudo gedit script02.sh
```

> Pour rappel, on peut ouvrir un **nouveau terminal** en utilisant la combinaison de touches **Ctrl+Alt+T** mais il est aussi possible d'ouvrir une **nouvelle fenêtre** depuis un terminal en utilisant **Ctrl+Shift+N** ou encore un **nouvel onglet** grâce à **Ctrl+Shift+T**.

26. Encore une fois, nous déclarons l'utilisation de **bash** en tête de document :

```
#!/bin/bash
```

Et nous récupérons la configuration de notre affichage sur le `script01.sh` :

```
###### AFFICHAGE ################################################
printf "\e[8;33;80t"
```

27. Dans un second temps, nous allons créer le quizz. Pour ce faire, nous écrivons un énoncé, une commande et nous définissons la réponse attendue :

```
echo " Une commande vient d'être exécutée. Son résultat est le suivant : "
whoami
reponse_attendue="whoami"
```

28. Nous posons la question :

```
read -p " Quelle est la commande qui vient d'être exécutée ? " user_input
```

29. Nous comparons enfin l'entrée de l'utilisateur avec la réponse attendue :

```
if [[ "$user_input" == "$reponse_attendue" ]]; then
   echo "Votre réponse est correcte."
else
   echo "Votre réponse est incorrecte. La commande correcte était :
$reponse_attendue"
fi
```

30. Notre code complet :

```
#!/bin/bash
###### AFFICHAGE ##################################################
printf "\e[8;33;80t"
####### SCRIPT ####################################################
echo " Une commande vient d'être exécutée. Son résultat est le suivant : "
whoami
reponse_attendue="whoami"
read -p " Quelle est la commande qui vient d'être exécutée ? " user_input
if [[ "$user_input" == "$reponse_attendue" ]]; then
   echo "Votre réponse est correcte."
else
   echo "Votre réponse est incorrecte. La commande correcte était :
$reponse_attendue"
fi
```

Nous devons passer les droits d'exécution à notre document `script02.sh` en ouvrant un nouveau terminal grâce à la combinaison de touches `Ctrl+Alt+T` et en tapant dedans :

```
sudo chmod +x script02.sh
```

Il ne reste plus qu'à tester notre programme via la commande :

```
sudo ./script02.sh
```

Jusqu'ici, le script donne cette sortie à son exécution :

 Capture d'écran du 2023-12-06 20-46-49

Nous observons que le script est fonctionnel mais tout va trop vite et n'est pas correctement formaté.

31. Nous ajoutons donc certaines de nos variables du `script01.sh` :

```
#!/bin/bash
###### AFFICHAGE ##################################################
printf "\e[8;33;80t"
###### VARIABLES ##################################################
e='echo'                # Permet d'écrire une ligne vide
s='sleep'               # Commande de pause
####### SCRIPT ####################################################
$e
$e " Une commande vient d'être exécutée. Son résultat est le suivant : "
$e
$s 2
whoami
$e
$s 2
reponse_attendue="whoami"
read -p " Quelle est la commande qui vient d'être exécutée ? " user_input
$e
if [[ "$user_input" == "$reponse_attendue" ]]; then
   $e "Votre réponse est correcte"
else
```

```
    $e "Votre réponse est incorrecte. La commande correcte était :
$reponse_attendue"
fi
$e
$s 10
```

Vous devriez obligatoirement obtenir le résultat suivant :


Capture d'écran du 2023-12-06 20-41-43

32. Nous pouvons ajouter des **émojis** pour féliciter l'utilisateur grâce à **unicode** exemple :

```
if [[ "$user_input" == "$reponse_attendue" ]]; then
    $e "Votre réponse est correcte 🥳 🎉"
```

> **Unicode** est un standard informatique qui permet des échanges de textes dans différentes langues, à un niveau mondial.

33. Nous pouvons à présent nous amuser à écrire un quizz complet permettant à l'utilisateur de découvrir les différentes commandes, mais aussi et surtout, de les retenir en mémoire. Afin de rendre notre jeu plus attractif, nous pouvons **ajouter des couleurs**, comme sur notre précédent script. Les différentes couleurs en **bash** :


Capture d'écran du 2023-12-07 13-50-49

En pratique, nous pouvons utiliser ces couleurs en les déclarant dans nos variables :

```
###### VARIABLES #################################################
e='echo'                # Permet d'écrire une ligne vide
s='sleep'               # Commande de pause
bt='tput setaf 7'       # Passer le texte en blanc
nt='tput setaf 0'       # Passer le texte en noir
b='tput setab 7'        # Passer la couleur de fond en blanc
n='tput setab 0'        # Passer la couleur de fond en noir
rm='tput sgr0'          # Permet de réinitialiser les couleurs
```

Ceci pourrait donner :

```
#!/bin/bash
###### AFFICHAGE #################################################
printf "\e[8;33;80t"
###### VARIABLES #################################################
e='echo'                # Permet d'écrire une ligne vide
s='sleep'               # Commande de pause
bt='tput setaf 7'       # Passer le texte en blanc
nt='tput setaf 0'       # Passer le texte en noir
b='tput setab 7'        # Passer la couleur de fond en blanc
n='tput setab 0'        # Passer la couleur de fond en noir
rm='tput sgr0'          # Permet de réinitialiser les couleurs
####### SCRIPT ###################################################
$e
$e "$($nt)$($b) Une commande vient d'être exécutée. Son résultat est le suivant :
$($rm)"
$e
$s 2
```

```
whoami
$e
$s 2
reponse_attendue="whoami"
read -p "$($nt)$($b) Quelle est la commande qui vient d'être exécutée ? $($rm)"
user_input
$e
if [[ "$user_input" == "$reponse_attendue" ]]; then
   $e "$($nt)$($b) Votre réponse est correcte 🥳 🎉 $($rm)"
else
   $e "$($nt)$($b) Votre réponse est incorrecte. La commande correcte était :
$($rm)$($bt)$($n) $reponse_attendue $($rm)"
fi
$e
$s 10
```