TP Linux

Ce compte rendu de TP permet de résumer les points essentiels vus lors de la première fiche de TP Linux.

I. Connexion et déconnexion sous Linux

Pour se connecter, il suffit de rentrer son *login* et son *mot de passe* dans les champs correspondants. De plus, l'utilisateur a le choix de l'environnement graphique. Par défaut, on utilisera KDE.

II. Découverte du système de fichiers Linux via KDE

1) Voici l'ensemble des contrôles que l'on peut effectuer via la souris sur le bureau, un dossier et sur un fichier :

	Bouton gauche	*Bouton du centre*	*Bouton droit*
Bureau	affiche le contenu du bureau	permet d'ouvrir le bureau dans un nouveau onglet	affiche les options
Dossier	affiche le contenu du dossier	permet d'ouvrir le dossier dans un nouveau onglet	affiche les options
Fichier	permet d'ouvrir le fichier	ne fait rien	affiche les options

2) Le dossier contenant mon répertoire principal de travail est : */filer/etudiants/g4/*

3) Listing du répertoire *$HOME* : Fait.

4) Création d'un répertoire puis suppression du répertoire et un fichier dans $HOME : Fait.

5)Création de l'arborescence de fichier : Fait.

III. Observation des processus via KDE

1) On ouvre l'application qui se nomme : *Table des processus*, elle permet de lister les processus actifs et d'en surveiller le comportement.

2) Voici les colonnes les plus utiles :

-La colonne *Utilisateur* décrit à quel utilisateur appartient le processus.
-Les colonnes *Titre de la fenêtre* et *Nom* permettent de savoir à quelle application correspond un processus.

-La colonne *PID* permet de connaître l'identifiant PID de chaque processus (pour faire apparaître cette colonne, on fait un *clic droit* sur l'en-tête des colonnes du tableau et on choisit *Afficher la colonne PID*).

3) Pour afficher :
- L'arborescence de tous les processus, on sélectionne : *Tous les processus*, *en arbre*.

- Uniquement mes processus, on sélectionne : *Processus utilisateurs*.

4) Observation des processus créés lors de l'ouverture de différents programmes : Fait.

5) Description des signaux d'arrêt de processus :
- *SIGTERM* : arrête un processus et le ferme (ne peut pas fermer un terminal).

- *SIGKILL* : tue un processus, arrêt brutal et fermeture (peut fermer un terminal).

IV. Premier contact avec l'interpréteur de commande

1) Dans le menu, on recherche le terme *Terminal* et on lance le raccourci nommé *Konsole*.

2) La commande *pwd* correspond à l'affichage du répertoire courant :

```
$ pwd
/filer/etudiants/g4/d15019960
```

La console s'ouvre donc sur le répertoire principal de l'utilisateur.

3) La commande *whoami* affiche le nom d'utilisateur :

```
$ whoami
d15019960
```

La commande *ps* permet d'afficher la liste des processus en cours :
-L'option *–f* permet d'affichage sous forme de liste.

-L'option *–U login* permet d'afficher uniquement les processus exécutés par l'utilisateur du login spécifié.

4) Consultation de la documentation sur les commandes *pwd*, *whoami* et *ps* : Fait.

5)La commande *ls* permet de lister le contenu du répertoire courant.

6)Les options de la commande *ls* :
-L'option *-a* permet d'afficher tous les fichiers et dossiers, même s'ils sont cachés.
-L'option *-l* permet d'afficher le résultat sous forme de liste.
-L'option *-al* regroupe les effets de l'option *-a* et de l'option *-l*.

-L'option *--color=auto* permet de colorer les fichiers selon leurs types.

7) Pour créer un alias, il faut se servir de la commande *alias*.

8) Création de l'alias *llc* se comportant comme la commande *ls -la --color=auto* :

```
$ alias llc='ls -la --color=auto'
```

9) Vérification que la commande *llc* donne le résultat prévu : Fait.

10) La commande *exit* ferme le shell.

11) On ouvre un nouveau shell et on essaie à nouveau la commande *llc* :

```
$ llc
bash: llc : commande introuvable
```

La commande *llc* n'existe plus, l'alias n'a pas été mémorisé à la fermeture du shell.

V. Se déplacer dans une arborescence

1) Création du répertoire *TP2* : Fait.

2) Vérification avec la commande *pwd* que le terminal se situe dans le répertoire principal : Fait.

Vérification avec la commande *ls* que le répertoire principal contient les dossiers *Initiation_Informatique* et *Introduction_Programmation* : Fait.

3) La commande *cd* permet de se déplacer dans l'arborescence de dossiers :

```
$ cd Introduction_Programmation
$ pwd
/filer/etudiants/g4/d15019960/Introduction_Programmation
```

4) On sort du répertoire *Introduction_Programmation* :

```
$ cd ../
$ pwd
/filer/etudiants/g4/d15019960
```

5) On se place dans le répertoire *TP2* :

```
$ cd Initiation_Informatique/Linux/TP2
$ pwd

/filer/etudiants/g4/d15019960/Initiation_Informatique/Linux/TP2
```

VI. Création de fichiers

1) Création d'un fichier vide nommé *bonjour* à l'aide la commande *touch* :

```
$ touch bonjour
```

2) Vérification que le fichier *bonjour* créé est bien vide à l'aide de la commande *cat* : Fait.

3) Ecriture dans le fichier *bonjour* :

```
$ cat > bonjour
Hello World
Ctrl + D
```

4) Affichage du contenu du fichier *bonjour* à l'aide de la commande *cat* :

```
$ cat bonjour
Hello World
```

5) Ajoute de texte dans le fichier *bonjour* :

```
$ cat >> bonjour
How do you do ?
Ctrl + D
```

6) Affichage du contenu du fichier *bonjour* :

```
$ cat bonjour
Hello World
How do you do ?
```

7) Copie du fichier *bonjour* vers un fichier *salut* à l'aide de la commande *cp* :

```
$ cp bonjour salut
$ diff bonjour salut
```

8) Différence entre > et >> :

```
$ cat > salut
Hi ! My name is Tuyen
Ctrl + D
$ cat salut

Hi! My name is Tuyen
```

Le symbole de redirection > sert donc a créer ou à écraser le contenu du fichier existant lors de l'écriture alors que le symbole >> permet d'ajouter le texte à la fin du fichier existant.

VII. Notion de filtre de fichiers

1) Afficher tous les fichiers dont le nom commence par une chaîne de caractères quelconque, suivi d'un a ou d'un m, suivi de nouveau d'un a ou d'un m, et termine par une suite de caractères quelconque :

```
$ ls *[a,m][a,m]*
```

2) Vérification que ce filtre est correct : Fait.

3) Afficher différentes formes de noms de fichiers :

```
$ cd /bin
$ ls a*
ls: impossible d'accéder à a*: Aucun fichier ou dossier de ce type
```

```
$ ls [a,e,i,o,u,y]*
echo  ed  egrep  elvis-tiny  ip  umount  uname  uncompress  ypdomainname
$ ls *sh*
bash  bash3  csh  dash  fdflush  ksh  pdksh  rbash  sh  sh.distrib  tcsh
$ ls ?sh
csh  ksh
```

4) Afficher les noms de fichiers dont la quatrième lettre est un a, un b, un c ou un d :

```
$ cd /sbin
ls ???[a,b,c,d]*
badblocks       discover-pkginstall  getcap   iptables-multi   on_ac_power  shadowconfig
blockdev        e2label              hdparm   iptables-restore  pccardctl   sysctl
discover        findfs               ipmaddr  iptables-save     rtacct      update
discover-modprobe  fstab-decode      iptables  lspcmcia         setcap
```

5) Retour au repertoire *TP2* :

```
$ cd

$ cd Initiation_Informatique/Linux/TP2
```

VIII. Nom de fichiers et caractères spéciaux

1) Création de trois fichiers *a*b*, *acb* et *addb* : Fait.
Exécution de la commande *ls a*b* :

```
$ ls a*b

a*b  acb  addb
```

2) Exécution des commandes *ls a*b* et *ls a"*"b* :

```
$ ls a\*b
a*b
$ ls a"*"b
a*b
```

On peut en déduire que l'antislash \ ou les guillemets permettent d'échapper un caractère et d'éviter son interprétation par le shell.

3) Voici les trois façons de créer un répertoire nommé *cou\cou* avec *mkdir* :

```
$ mkdir cou\\cou
$ mkdir "cou\cou"
$ mkdir cou"\"cou
```

4) Création d'un fichier dont le nom est mon fichier : `$ touch mon\ fichier`

IX. Compression/Décompression de fichiers

1) On crée le répertoire *TP3* : Fait.

2) On se place dans le répertoire *TP3* : Fait.

3) Consultation des documentations des commandes *zip*, *unzip*, *gzip*, *gunzip* : Fait.

4) Téléchargement du *Mémo Unix / Linux* via *firefox* : Fait.

5) Le fichier *Memo_unix.pdf* fait *299884 octets*.

Compression et décompression du fichier *Memo_unix* au format :

Format	***ZIP***	***GZIP***
Commande de compression	$ zip Memo_unix.zip Memo_unix.pdf	$ gzip -c Memo_unix.pdf > Memo_unix.gz
Commande de décompression	$ unzip Memo_unix.zip	$ gunzip Memo_unix.gz
Taille de l'archive compressée	*290050 octets*	*289906 octets*

6) Consultation de la documentation de la commande *tar* : Fait.

7) Archiver le répertoire *TP3* au format *TAR* : $ tar -cf TP3.tar TP3
Lister le contenu de l'archive *TP3.tar* : $ tar -tvf TP3.tar

Restituer le répertoire *TP3* : $ tar -xf TP3.tar

X. Les droits d'accès

1) Création de l'arborescence demandée :

```
$ mkdir -m 777 RepertoireTestDroit
$ cd RepertoireTestDroit

$ cat > monFichierPerso
Mon Secret
Ctrl + D
$ chmod 666 monFichierPerso

$ mkdir Sous\ Repertoire\ 1
$ cat > Sous\ Repertoire\ 1/fic
Hello World
Ctrl + D

$ mkdir Sous\ Repertoire\ 2
$ cat > Sous\ Repertoire\ 2/fic
```

```
Salut l'ami
Ctrl + D

$ chmod 666 Sous\ Repertoire\ 1/fic
$ chmod 666 Sous\ Repertoire\ 1
$ chmod 666 Sous\ Repertoire\ 2/fic

$ chmod 111 Sous\ Repertoire\ 2
```

2) Commandes pouvant être exécutées depuis le répertoire *RepertoireTestDroit* :

-$ ls ./Sous Repertoire 1 : Ok
$ ls -l ./Sous Repertoire 1 : Erreur
$ cd ./Sous Repertoire 1 : Erreur
$ cat ./Sous Repertoire 1/fic : Erreur
$ touch ./Sous Repertoire 1/fic2 : Erreur
$ rm ./Sous Repertoire 1/fic : Erreur
$ ls ./Sous repertoire 2 : Erreur
$ cd ./Sous repertoire 2 : Ok
$ cat ./Sous Repertoire 2/fic : Ok
$ touch ./Sous Repertoire 2/fic2 : Erreur
$ rm ./Sous Repertoire 2/fic : Erreur
$ echo "Comment vas tu" >> ./Sous Repertoire 2/fic : Ok

3) Avec les droits --- --- ---, même lui n'aura plus accès à son ficher, il faut plutôt qu'il attribut à son répertoire personnelle, les droits *rwx* --- ---.

XI. Les alias permanets

1) Vérification de la présence des fichiers *.bash_profile* et *.bashrc* : Fait.

2) A la fin du fichier *.bashrc*, on met la commande suivante : *alias llc='ls -la --color=auto'*.
Vérification que l'alias est permanent : Fait.

XII. Gestion des processus

1) Lancement de la commande *emacs* au premier plan puis en arrière plan :

```
$ emacs
Ctrl + Z
[1]+ Stopped     emacs
$ bg
[1]+ emacs &
$ ps
```

2) La commande *ps -l* permet d'afficher sous forme de liste les caractéristiques de chaque processus (dont le PID et le PPID).

Le processus père au processus *emacs* est le processus *bash* (le shell).
On repasse le processus d'*emacs* au premier plan : `$ fg`

3) La commande *ps -Af* permet d'afficher tous les processus exécuté sur l'ordinateur sous forme de liste.
On tue le processus correspondant au *bash* précédent : `$ kill -9 2642`

Le *bash* précédent s'arrête ainsi qu'*emacs*.

4) On lance dans un nouveau terminal *emacs* en arrière plan : `$ emacs &`
On tue le processus de ce nouveau terminal : `$ kill -9 2642`
Lorsqu'on tue le processus du nouveau terminal, alors que des processus sont lancés en arrière plan dans ce terminal, les processus fils du terminal tué sont alors adoptés par le processus *init* (PID : 1).

XIII. Redirections

1) Stockage de la commande *ls -l* dans un fichier *toto* : `$ ls -l > toto`

2) Ajout de la commande *ps -A* au fichier *toto* à l'aide d'une redirection non écrasante :

```
$ ps -A >> toto
$ cat toto
```

3) Stockage de la liste des processus tournant sur le réseau dans le fichier *tata* : `$ ps -Af > tata`

4) -AfLa commande *wc* permet d'afficher le nombre d'octets, de mots et de lignes d'un fichier.
La commande *sort* permet de trier les lignes d'un fichier.wc -ww
La commande *head* permet d'afficher le début d'un fichier.

La commande *tail* permet d'afficher la dernière partie d'un fichier.

A l'aide de ces commandes, on peut réaliser diverses tâches :

- Compter le nombre de mots contenus dans le fichier *bonjour* : `$ wc -w bonjour`
- Compter le nombre de caractères contenus dans le fichier *bonjour* : `$ wc -m bonjour`
- Compter le nombre de processus dans *tata* : `$ wc -l tata`
- Trier les lignes du fichier *tata* dans l'ordre alphabétique : `$ sort -g tata`
- Extraire les dix premières lignes du fichier *toto* : `$ head toto`
- Extraire les cinq dernières lignes du fichier *toto* : `$ tail -n 5 toto`

5) Enregistrement dans un fichier *titi* les lignes 4 à 23 du fichier *tata* triées suivant le nom des propriétaires des processus : `$ sort tata | head -n 23 | tail -n 20 > titi`

XIV. Recherche d'une chaîne de caractères dans des fichiers textes

1) Documentation des options de *grep* :
- Donne seulement le nombre de lignes trouvées concernant le motif : *grep -c*
- Donne seulement le nom des fichiers dans lesquels le motif a été trouvé : *grep -l*
- Donne les lignes où le motif n'a pas été trouvé : *grep -v*
- Ne tient pas compte de la case : *grep -i*

- Impose que le motif corresponde à un mot entier sur la ligne du fichier : *grep –x*

2) Le concept d'expressions régulières est différent de celui du filtrage :
Le point . correspond à n'importe quel caractère.
Le point d'interrogation ? est utilisé pour indiquer que l'élément précédent est facultatif et peut être rencontré au plus une fois.

L'étoile * est utilisée pour indiquer que l'élément précédent peut être rencontré zéro ou plusieurs fois.

3) Explication de commandes *grep*
`$ grep "^.o" tata` : Recherche les chaînes de caractères commençant par n'importe quel caractère suivi d'un 'o' - affichage de toutes les lignes commençant par root
-`$ grep -ni bash tata` : Recherche les chaînes de caractères contenant "bash" en ignorant la case et en préfixant la ligne par son numéro.
-`$ grep -ni "td$" tata` : Recherche les chaînes de caractères terminant par "td" en ignorant la case et en préfixant la ligne par son numéro.
4) Sélection des lignes du fichier *tata* commençant par un "r" et terminant par un chiffre :

```
$ grep "^r.*[0-9]$" tata
```

5) Affichage uniquement des processus lancés par moi : `$ ps -Af | grep "d1106906"`

6) Nombre de processus lancés par moi : `$ ps -Af | grep "d1106906" | wc –l`

XV. Recherche de fichiers dans une arborescence

1) La commande *locate* recherche un fichier dans la base de données contenant l'ensemble des noms de fichiers du système, alors que la commande *find* parcoure l'arborescence afin de trouver le nom du fichier.

```
$ locate ls
$ find -name titi
```

2) Simulation grossière de la commande *find* : `$ ls -R -l | grep "titi$"`

XVI. Les variables d'environnement

1) Voici la signification des variables d'environnement principales :
- HOME : Répertoire principal
- USER : Login
- DISPLAY : Display
- TERM : Term
- HOSTNAME : Nom de la machine
- SHELL : Terminal par défaut

- PATH : Le chemin où sont cherchés les exécutables

2) Voici les actions que produisent les commandes suivantes :
$ touch fichier $USER : Crée un fichier nommé *fichier* et un autre dont le nom est *d1106906*.
$ touch "fichier $USER" : Crée un fichier dont le nom est *fichier d1106906*.
$ touch 'fichier $USER' : Crée un fichier dont le nom est *fichier $USER*.

Les apostrophes ' permettent de ne pas interpréter la chaîne de caractères alors que les guillemets " permettent de spécifier des espaces dans les noms de fichiers, mais le contenu est interprété.

3) Manipulations de quelques variables :

```
$ TEMP2="ma première variable"
$ TEMP3="$PATH"
$ echo "$TEMP3"
/bin:/usr/bin:/usr/local/bin:/usr/sbin:/sbin:/usr/local/sbin:/usr/X11R6/bin:$HOME/bin
$ echo "$PATH"
/bin:/usr/bin:/usr/local/bin:/usr/sbin:/sbin:/usr/local/sbin:/usr/X11R6/bin:$HOME/bin
$ PATH="salut"
$ ls
ls : commande introuvable
$ PATH="$TEMP3"

$ ls
```

On peut en déduire que le terminal utilise la variable d'environnement *PATH* afin de localiser les dossiers contenant les commandes utilisables.

XVII. Un premier script

1) On enregistre le script dans un fichier nommé *bonjour* et on l'exécute :

```
$ ./bonjour
Bonjour d15019960
```

```
Ceci est mon premier script
```

XVIII. Les arguments des scripts

1) Script qui affiche son nom, ses quatre premiers arguments et son nombre d'arguments :

```
#! /bin/bash
echo "Le nom du script est : $0"
echo "Les quatre premiers arguments sont : $1, $2, $3, $4"

echo "Le nombre d'arguments est : $#"
```

2) Script qui affiche le contenu du répertoire dont le chemin est placé en argument :

```
#! /bin/bash
ls -a $1
```

3) Script qui crée le répertoire dont le nom est le premier argument et qui ensuite crée dans ce dossier le fichier dont le nom est le deuxième argument en retirant les droits d'écriture et d'exécution à tous les utilisateurs :

```
#! /bin/bash
mkdir "$1"
touch "$1/$2"

chmod 444 "$1/$2"
```

XIX. Les conditionnelles dans les scripts

1) Exécution et modification du script pour bien le comprendre : Fait.

2) Script qui prend en argument le nom d'un fichier et s'il existe, il affiche son contenu sinon il renvoie une erreur :

```
#! /bin/bash
if [ -f "$1" ]
then
        cat "$1"
else
        echo "Erreur : fichier introuvable"

fi
```

XX. Les boucles dans les scripts

1) Exécution et modification des scripts pour bien les comprendre : Fait.

2) Script qui indique pour chaque fichier du répertoire courant si c'est un dossier ou non :

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

```
for fichier in ./*
do
        echo "$fichier : "
        if [ -d "$fichier" ]
        then
                echo "Oui"
        else
                echo "Non"
        fi
done
```

3) Script qui reproduit le fonctionnement de *ls -R* :

```
#! /bin/bash
for fichier in ./*
do
        if [ -d "$fichier" ]
        then
                echo "$fichier : "
                cd "$fichier"
                echo `ls`
                cd ../
        else
                echo "$fichier"
        fi
done
```