## TP1

### < cours >

**[1]** se connecter à un utilisateur différents : **su login-du-second-utilisateur.**

**[2]** établir une connexion distante à une autre machine : **ssh nom-de-la-machine.**

**[3]** **« whoami »** et **« id »** donnent des informations sur l’utilisateur courant (vous).

**[4]** **« who »** permet de connaître la liste des personnes connectées sur la machine.

**[5]** **« w »** (what) dit plus précisément qui fait quoi sur la machine.

**[6]** **« hostname »** donne le nom (complet) de la machine.

**[7]** Interrompre les connexions à l’aide de la commande **« exit »**.

**[8]** Sans option, **« cal »** (et **« ncal »**) permettent d’afficher le calendrier du mois courant.

**[9] « pwd »** sans argument, indique le répertoire courant.

**[11]** **« cd »**   
– avec en argument le nom d’un sous-répertoire : permet de se déplacer   
– avec l’argument « .. », remonte vers le répertoire parent  
– sans argument, ramène dans votre répertoire personnel.

**[12]** **« ls »** liste le contenu du répertoire courant – ses fichiers et ses sous-répertoires.

**[13]** « tar » sert à archiver plusieurs fichiers en un seul :   
– archiver : **tar -cvf toto.tar fic1 fic2**   
– extraire : **tar -xvf toto.tar**

**[14] « mkdir »** avec un paramètre rep, crée un répertoire vide de nom rep.

**[15] « touch »** avec un paramètre fic, permet de créer un fichier vierge de nom fic.

**[16] « mv »** permet de renommer et/ou de déplacer un fichier ou un répertoire.

**[17] « cp »** permet de dupliquer un fichier pour obtenir deux fichiers totalement autonomes.

**[18] « cat »** avec en un ou plusieurs noms de fichiers, permet d’afficher son ou leur contenu.

**[19] « rm »** avec en argument un nom de fichier fic, supprime le fichier fic.

**[20] « rmdir »** avec en argument un nom de répertoire supposé vide, supprime le répertoire.

### 

## TP2

### < cours >

**[1]** Un fichier (ordinaire/répertoire) peut être désigné en utilisant deux types de références (ou chemins) :   
– une référence absolue : depuis sa racine (« / ») jusqu’au fichier ;   
– une référence relative : depuis le répertoire de travail courant de l’utilisateur

Une référence relative ne commence jamais par « / ».   
À l’inverse, une référence absolue commence par « / », ou éventuellement par « ∼ » ou « ∼machin ».

**[2] « du »** affiche des informations concernant l’espace occupé sur le disque par une arborescence.

**[3] « diff »** compare les fichiers ou répertoires et affiche les différences fichier par fichier.

**[4]** les jokers ou wildcards :  
**«\*»** représente n’importe quel morceau de nom de fichier (sauf le «.» en début) ;   
**«?»** représente exactement un symbole quelconque (sauf le « . » en début) ;  
**«[]»** représente exactement un symbole parmi l’ensemble de symboles :   
 – soit par liste exhaustive de caractères, par exemple [abz] ;   
 – soit par intervalle : par exemple [a-f] ou [2-7] ;   
 – soit par union de listes et d’intervalles, par exemple [a-fA-FzZ] ;   
 – soit par complément avec «ˆ» ou «!», par exemple [^wx] ou [!a-z].

### < exo >

**Comment lister (exactement) les noms 1 de fichier dans le répertoire /usr/bin ...**

**1. qui commencent par un r ;**  ls r\*

**2. qui contiennent un r ;**  ls \*r\*

**3. qui terminent par un r ;**  ls \*r

**4. qui commencent par un p et terminent par un n ;**  ls p\*n

**5. dont la troisième lettre est un r ;**  ls ??r\*

**6. qui commencent par un r et contiennent exactement 5 caractères ;**  ls r????

**7. qui commencent par un r et contiennent au moins 5 caractères ;**  ls r????\*

**8. qui commencent par un r et contiennent entre 3 et 5 caractères ;**  ls r?? r??? r????

**9. qui contiennent un s ou un l ;**  ls \*1\* \*s\*

**10. qui contiennent un chiffre ;**  ls \*0\* \*1\* \*2\* \*3\* \*4\* \*5\* \*6\* \*7\* \*8\* \*9\*

**11. qui contiennent un caractère non alphabétique ;**  ls \*[!a-z]\*

**12. qui contiennent un r et un n ;**  ls \*r\*n \*n\*r\*

**13. qui contiennent un r et commencent par une lettre parmi l, m, n, o, p,q ;**  ls [l-q]\*r\*

**14. qui contiennent un r et ne commencent pas par un n ;**  ls [!n]\*r\*

**15. qui contiennent un r et ne commencent pas par une lettre parmi l, m, n, o, p.**  ls [!n]\*r\*

## TP3

### < cours >

« alias » – avec un argument de la forme ma\_commande=commande\_complète, définit (ou redéfinit) la commande « ma\_commande ». Attention, si commande\_complète contient des espaces (par exemple, s’il y a des options), il faut alors la délimiter avec des guillemets. – sans argument, liste tous les raccourcis qui ont été définis.

Liens et i-nœuds Un système UNIX identifie un fichier par son i-nœud (inode en anglais pour index node) et non par son (ou ses) nom(s). Un i-nœud contient toutes les caractéristiques concernant le fichier correspondant : son type (ordinaire ou répertoire), l’adresse du ou des emplacement(s) sur le disque où se trouve son contenu, les droits accordés, la date de dernière modification, etc. Les i-nœuds sont stockée dans une table et identifiée dans cette table par un numéro.

Un répertoire est un fichier qui contient une liste de couples (nom, numéro d’i-nœud), comme dans le tableau ci-contre. Lorsqu’on dit qu’un fichier fic « se trouve » dans un répertoire rep, cela signifie en fait que rep contient un couple associant le nom fic au numéro d’i-nœud d’un fichier. On appelle cela un lien de nom fic vers cet i-nœud.

« ln » (link) permet de créer un nouveau nom (lien) vers un i-nœud existant : si ancien est un nom valide et nouveau un nouveau nom, « ln ancien nouveau » crée un nouveau lien de nom nouveau vers le même i-nœud que ancien.

### < exo >

Exercice 1:

1. tfranc49@x15:/home2/tfranc49$ alias li="ls -1"

2. tfranc49@x15:/home2/tfranc49$ alias la="ls -a"

3. tfranc49@x15:/home2/tfranc49$ alias ls="ls -x"

4. tfranc49@x15:/home2/tfranc49$ alias rm="rm -i"

Exercice 3:

1. tfranc49@x15:/home2/tfranc49/Cours/2019/IS1$ ls -i TP3

2. tfranc49@x15:/home2/tfranc49/Cours/2019/IS1$ rm TP3/grosminet TP3/sylvestre

Le numéro d'i-nœud est le même pour sylvestre et grosminet.

3.tfranc49@x15:/home2/tfranc49/Cours/2019/IS1/TP3$ cp sylvestre gatto\_silvestro

Le numéro d'i-nœud est différent pour ces deux fichiers.

5.tfranc49@x15:/home2/tfranc49/Cours/2019/IS1/TP3/Canards$ ln FiFi loulou

6. tfranc49@x15:/home2/tfranc49/Cours/2019/IS1/TP3/Canards$ cp RiRi hihihi

tfranc49@x15:/home2/tfranc49/Cours/2019/IS1/TP3/Canards$ cat RiRi FiFi loulou hihihi

Ils ont tous le même contenu.

tfranc49@x15:/home2/tfranc49/Cours/2019/IS1/TP3/Canards$ ls -i RiRi FiFi loulou hihihi

RiRi FiFi et loulou ont le même numéro d'i-nœud mais pas hihihi.

7. tfranc49@x15:/home2/tfranc49/Cours/2019/IS1/TP3/Canards$ echo "Tout est en ordre, Donald." > hihihi

8. tfranc49@x15:/home2/tfranc49/Cours/2019/IS1/TP3/Canards$ cat RiRi FiFi loulou hihihi

Puisque les fichiers ont le même numéro d'i-nœud, éditer le contenu de l'un édite le contenu de tous les autres.

9.tfranc49@x15:/home2/tfranc49/Cours/2019/IS1/TP3/Canards$ ln loulou FiFi

Exercice 4:

tfranc49@x15:~$ id

tfranc49@x15:~$ id root

On appartient au groupe "Sc.Exactes\_Autres" et le super-utilisateur root appartient au groupe "root".

Exercice 5:

1. tfranc49@x15:~/Cours/2019/IS1/TP3$ ls -l

La taille du plus gros fichier ordinaire est 32 octets.

2. Le propriétaire et le groupe propriétaire sont les mêmes, cependant les droits des deux fichiers sont différents. On a le droit d'exécution sur le fichier "program".

3. On constate 3 liens qui pointent vers l'i-nœud du CoffreDePicsou. Cela s'explique car de base un répertoire a 2 liens qui pointent vers son i-nœud ( le . et lui-même étant dans son fichier parent) et comme CoffreDePicsou a un sous-répertoire il a également un troisième lien (..).

Exercice 6:

1. On en déduit qu'on ne peut pas modifier tata, qu'on ne peut que le lire.

2. Pour titi, on ne peut pas le lire, on ne peut que le modifier.

3. Pour toto, on a le droit de le lire et de l'exécuter tandis que pour tata nous n'avons que le droit de le lire et non pas de l'exécuter d'où une erreur de permission en faisant la commande ./tata .

4. Pour tutu, nous n'avons que le droit de l'exécuter. En l'exécutant on obtient la date et l'heure.

## TP4

### < exo >

Exercice 1:

2. Pour cet exercice on se place dans /home2/tfranc49/Cours/2019/IS1/TP4/A$

a. chmod a=+r+x,u+w titi

b. chmod a=+r,u+x titi

c. chmod a=+r titi

d. chmod a=-r,g+r titi

e. chmod u=+w+x,g=+x,o=+r titi

f. chmod u=w,g=x,o=rx titi

Exercice 2:

[effacé]

Exercice 3:

On reste dans TP4/B/

1.chmod 544 A/toto

2.chmod 624 A/tutu

3.chmod 216 B/toto

4.chmod 655 B/tutu

5.chmod 254 C/\*

6.chmod u+x \*/tutu

7.chmod 640 \*/toto

Exercice 4:

1. On constate, après avoir supprimé le droit de lecture, que l'exécution de tutu fonctionne correctement.

Tandis que celle de toto ne fonctionne pas "Permission non accordée".

Cela s'explique car l'exécution de toto requiert la lecture du fichier tandis que celle de tutu ne requiert que l'exécution.

2. On se place dans TP4/

On fait touch youpi

-> chmod 500 youpi

-> ./youpi et on peut l'utiliser comme commande.

Exercice 5:

1.On se place dans TP4/

Pour A: on a le droit de lecture,écriture et exécution.

Pour B: on a le droit de lecture et d'exécution

Pour C: on a le droit de lecture et d'écriture.

Pour D: on a le droit d'écriture et d'exécution.

On peut lister le contenu de ABC mais pas de D.

On en déduit donc que le droit de lecture pour les répertoires permet d'accéder au contenu d'un répertoire.

2. On ne peut pas supprimer un fichier, le renommer dans le répertoire B néanmoins on a pu modifier le contenu de titi.

On en déduit donc que le droit d'écriture pour les répertoires permet de modifier le contenu du répertoire.

(créer des nouveaux dossiers, ajouter des fichiers, remplacer des fichiers, renommer des fichiers)

3.On se place dans TP4

a. echo Lingots >> Gringotts/coffre

b. touch Gringotts/Caveaux/cav1 Gringotts/Caveaux/cav2

d. La commande cat ne marche pas, la commande ls ne marche que pour ls Gringotts et pas pour ls Gringotts/Caveaux.

Explication : il n'y a plus le droit d'exécution.

4.Le droit d'exécution pour les répertoires, permet d'afficher les droits, de supprimer, créer un ficher, de se déplacer dans celui-ci.

5.Les droits d'exécution et de lecture d'un répertoire permettent de voir le contenu de ses fichiers et de consulter ses droits.

6.chmod u=x+r C

7.chmod u+w C B

8.chmod u+r D

chmod u+r D/C

9.chmod 777 D/\*/\* D/\*

rm -R D

Exercice 6:

1.umask affiche les droits qu'il manque pour chaque catégorie

(utilisateur, groupe, others) en notation octal.

Tandis que umask -S indique les droits que les fichiers ont, en notation symbolique.

2. On observe que pour toto on a un droit de lecture et de modification.

Tandis que pour Rep on ces derniers en plus du droit d'exécution.

3.On se place dans test

On remarque pour Rep1 et titi, il manque des droits par rapport à Rep et toto.

On remarque donc, que le masque empeche d'avoir d'autres droits que ceux indiqués par ce dernier.

6.umask 077

7.umask 022

8.umask 026

9.en modifiant le fichier tp.bashrc de manière a faire un masque qui nous convient

Exercice 8:

2.Le voisin ne peut pas lire ce dernier mais peut le supprimer et renommer

3. Utilisateur courant : chmod 770 ./

Voisin : touch voisin

chmod 640 voisin

Utilisateur courant : chmod 740 ./

## TP5

### < exo >

Exercice 3

On se place dans Jokers

3.cat Boileau/\* >> poeme\_boileau

4.cat Ronsard/urgwe6 Ronsard/vawGl Ronsard/zuk4 >> poeme\_ronsard

Exercice 4

1.La profondeur de Vide est de 5 (en le comptant :

Vide => .Cachée => .Caradoc => A => A)

2. Il y en a 5 distincs.

Exercice 5

On se place dans TP5

mkdir Spirou

touch Spirou/champignac

mkdir Spirou/Fantasio

Pour afficher 5 dans le ls -ld Spirou

mkdir Spirou/.coucou

mkdir Spirou/.bonsoir

Pour afficher 8 dans le du -a Spirou | wc -

touch Spirou/.où.txt

touch Spirou/.là.txt

touch Spirou/.ici.txt

Exercice 6

On se place dans TP5

1.mkdir Poudlard Poudlard/CabaneHurlante Poudlard/CabaneHurlante/Honeydukes Poudlard/SauleCogneur

echo "Voici un animal fantastique" >> Poudlard/CabaneHurlante/fumseck

3.On ne peut pas modifier les droits de ls\_fumseck

4. On ne peut lire que lp\_fumseck.

Restreindre l'accès au fichier semble avoir un effet sur le lien symbolique.

5.Tout est normal pour lp\_fumseck tandis que pour ls\_fumseck, il y a la même erreur.

Le lien symbolique semble affecter par les changements de dossiers du fichier original.

6.lp\_fumseck reste relier au premier fumseck.

Tandis que pour ls\_fumseck, il lit le contenu du nouveau fichier.

le lien physique prend compte uniquement du fichier original.

Alors que le lien symbolique prend en compte le titre et son emplacement.

7.Déplacer le lien symbolique provoque une erreur.

8.Le lien symbolique marche.

Le déplacement du lien symbolique n'est donc pas important, mais celui du lien original oui.

10.

cd passageSecret

cd SauleCogneur/tunnel/Honeydukes/

cd CabaneHurlante/Honeydukes/

Avec la commande pwd, cela affiche différents chemins absolus.

Alors qu'avec pwd -P cela affiche la même : ... TP5/Poudlard/CabaneHurlante/Honeydukes

12. On se place dans Honeydukes

mkdir fred

ln -s fred george

mv george ..

mkdir george

rm -d fred

ln -s george fred

rm -d george

cd ..

mv george Honeydukes/

### TP6

### < exo >

Exercice 1

1.Sur le premier terminal, on y trouve les processus suivants : bash, xcalc, xclock, ps

Sur le second, on trouve que bash et ps

ps -ax

ps -g

On a comme information pour chaque processus, son identifiant, son "TTY", ce qui semble être son statut, une colonne "temps", et son nom

2.Avec ps -l on a : UID(21952540), PID (3306 et 3307), PPID(3283), PRI (80), SZ(20118 et 15300), WCHAN (poll\_s) et TTY(pts/1)

Avec ps -u on a : l'utilisateur, PID, VSZ (80472 et 61200), RSS (9352 et 5500), TTY, STAT, et l'heure de lancement

UID:Identifiant de l'utilisateur

PPID: Identifiant des processus parents

PRI:L'importance du processus

TT(Y):terminal associé au processus

SZ:taille en kb

S(TAT):statut du processus

3. ps a

4. ps -U machin

ps -U root

5.ps -p

6.Hormis bash (PPID : 1965), leur PPID est 3283

Le processus parent de xclock est bash (PPID : 1965)

Le processus parent de bash est xfce4-terminal

Le processus parent de xfce4-terminal est xfce4-session

Exercice 2

1. PID:identifiant du processus

UID:Identifiant de l'utilisateur

PRI:L'importance du processus

SIZE/RES : taille que ça représente dans la mémoire

COMMAND: nom de la commande/ligne de commande

le tri se fait par la colonne %CPU

2. Avec htop il suffit d'appuyer sur la colonne "time" pour trier en fonction du temps

D'appuyer sur "tree"/F5 pour afficher l'aborescence

D'appuyer "filter"/F4 pour rentrer notre nom d'utilisateur

Exercice 3

1.kill -l

15,4,2,19,18

2. Avec SIGTERM : Cela termine le processus

SIGKILL : Cela termine le processus

SIGINT : cela interrompt le processus

SIGSTOP : cela "met pause" sur le processus (la calculatrice n'affiche plus de bouton)

SIGCONT : cela remet en marche le processus

3. SIGTERM: ne marche pas sur le processus bash (et xcalc & est toujours là)

SIGKILL: arrive à terminer le processus bash (et xcalc & se termine aussi)

SIGINT : ne marche pas non plus sur le processus bash (et xcalc & est toujours là)

SIGSTOP : le processus bash est "mis en pause" (cela n'a pas d'impact sur xcalc &)

SIGCONT : le processus bash se remet en marche depuis un autre terminal

4. Sur trois générations, les ancêtres de xcalc & sont : xterm &, bash et xfce4-terminal

5.SIGTERM: le terminal se ferme, mais pas la calculatrice

SIGKILL: le terminal et la calculatrice se ferment

SIGINT : le terminal et la calculatrice se ferment

6."Opération non permise", il semble que l'on puisse arrêter que les processus qui nous appartiennent

7.Depuis htop, il faut sélectionner le processus, puis appuyer sur "kill"/F9 et "send"/entrer

8.Cela ferme tous les processus xclock

Exercice 4

1.Le processus n'existe plus

2.nohup xclock &

Le processus surviet bien à a fermeture du terminal depuis lequel il a été lancé

3.

Exercice 5

1. Etat : S2. Etat : Sl

Etat : Tl

3.Le signal envoyé est le n°20 : SIGTSTP

4.kill -18 2750

Exercice 6

1. Contrairement à ps, on a aucune sortie

2. Les numéros correspondent à l'ordre dans lequel on a exécuté les programmes.

jobs -l

Exercice 7

1.Un seul processus peut être en avant-plan

2.xcalc &

3.SIGINT

## TP7

### < exo >

Exercice 1

1. ">" permet d'écraser le contenu d'un fichier.

2. ">>" permet de mettre du contenu à la suite de ce qui est déjà écrit.

Exercice 2

1.caractère

2.date > /dev/pts/0

Cela exécute la commande sur le premier terminal

3.Permission non accordée. Puis ça marche

Exercice 3

3.Comme on a redirigé la commande "meuzobuga" dans "meuzobu".

Le terminal essaie d'exécuter "meuzobuga" même s'il n'existe pas, et nous affiche le message d'erreur

Exercice 4

1.ls LesCowboysFringants/\*/\*[A-Z]\* 2> /dev/null

2.ls -R LesCowboysFringants/ 2> /dev/pts/0

Exercice 5

1. Pour "cat", ça ne semble rien changer, tandis que pour "wc" cela enlève simplement "BlancheNeige"

Exercice 6

2. ctrl-c semble arrêter la commande "cat" à tout moment, alors que pour "ctrl-d" il faut un retour à la ligne (sinon ça fait un copie-coller)

3.Cela affiche le contenu avec le numero des lignes

4.cat -s Sorciere > Reine

Exercice 7

1.head -6 Reine

2. Ca affiche les 3 dernières lignes qu'on a tapé

3. Ca affiche tout ce qu'on marque juste après l'avoir tapé.

Et on ne peut écrire que trois lignes.

4.Elle affiche les n (ici 100) premiers caractères. head -c 100 Reine

5. tail -n +2 Reine .

Ca affiche les dernières lignes à partir de la deuxième.

6.tail -q BlancheNeige Prince

7.

## TP8

### < exo >

Exercice 1

1.ps ax | less

2.ps | tail -n +3 | wc -l

Exercice 2

2.echo tr 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ' 'bcdefghijklmnopqrstuvwxyzaBCDEFGHIJKLMNOPQRSTUVWXYZA' > crypte

ls | bash crypte

3.echo tr 'bcdefghijklmnopqrstuvwxyzaBCDEFGHIJKLMNOPQRSTUVWXYZA''abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ' > decrypte

4.ls | bash crypte | bash decrypte

5.ls | bash crypte | tee > message\_secret | bash decrypte

Exercice 3

1.cat /dev/zero | head -c 10000 > nul

2.cat /dev/zero | head -c 200 | tr '\0' 'a' > grosfichier

3.yes "Répétez après moi - \"après moi\"" | head -n 1000 | cat - >>grosfichier

Exercice 4

1. cat -n grosfichier | tail -3

2.cat -n grosfichier | head -7 | tail -1

3.cat grosfichier | tr " " "\n" |head -n 579 |tail -1

Exercice 5

1.ps ax | grep /user/sbin

2.ls -R | grep .java

3.ls -p -R ~ | grep -c -v "/[.]\*\|^$"

ls -p -R ~ | grep -v "/[.]\*\|^$" | wc -l

4.Il s'agit de l'option v.

5.

a.ls -p -R /lib | grep -c "s..s"

b.ls -p -R /lib | grep -c "s\{2,\}"

c.ls -p -R /lib | grep -c "s\{2\}"

d.ls -p -R /lib | grep -c "[^0-9]"

Exercice 6

3. od -i alea | tr -s '' | head -n1 | tr '''\n' |head -n2 | tail -n1

## TP9

### < exo >

Exercice 1

2.echo "Bonjour $USER, vous actuellement dans le répertoire $PWD"

3.On peut choisir là où la commande cd nous ramènera

4.OLDPWD indique le chemin du fichier où l'on se trouvait précédemment.

"cd -" nous permet d'y aller.

Exercice 2

3.

chmod 777 fic.sh

bash TP9/fic.sh (depuis IS1), cela l'exécute correctement

./fic.sh (depuis TP9), cela l'exécute correctement

fic.sh (depuis TP9), cela ne l'exécute pas correctement

la dernière façon d'exécuter ne marche pas car, le script ne se trouve pas dans le "PATH" par défaut.

4.PATH=$PATH:~/Cours/2019/IS1/TP9

5.PATH=$PATH:~/mybin

Exercice 3

3.NOM=Tanya

La variable NOM apparaît bien dans la liste

5.

contenu du script :

echo "Bonjour, $NOM"

NOM=Tartempion

echo "Bonjour, $NOM"

On obtient l'affichage suivant :

Bonjour,

Bonjour, Tartempion

la variable NOM n'a pas changé de valeur dans le shell initial

6.

Pour la question 4, on obtient le même résultant.

Pour la question 5, on obtient l'affichage :

Bonjour, tanya

Bonjour, Tartempion

7. une variable d'environnement à une portée touchant le terminal où l'on exécute le script et les scripts exécutés.

Tandis qu'une variable simple n'est utilisée que par le terminal utilisé.

Exercice 5

echo "Je dois saluer $# personnes"

echo "Bonjour "$@" "

Exercice 6

echo "Le chemin absolu de mon répertoire courant est $PWD

Le résultat de la commande whoami est $(whoami)

Le produit de 33 par 17 vaut $((33\*17))

Il a agi " anti{constitutionel,conventionnel,socia}lement

Exercice 8

echo "A B

L’ensemble {1, 2, 3} est inclus dans N.

Elle demanda d’une voix \*forte\* : \"Qui est-ce ?\""

echo '$HOME = ${HOME}'" = $HOME"

echo "19 \* 216 = $((19\*216))

Ajoutez un peu d’huile d’olive, d’humeur joyeuse, d’humus, et remuez bien !"

Exercice 9

TOTO=whoami

echo "J’aime bien utiliser la commande "$TOTO"."

echo "Ma variable" '$TOTO' "contient '$TOTO'."

echo "Mon login est $($TOTO)

La commande d’aujourd’hui est" '\'"$TOTO"'\'".

${TOTO:0:3}areyou, c’est une commande imaginaire !"

## TP10

### < exo >

Exercice 1

2.Un seul processus a été créé, le processus xclock se lance, cela lance le processus xclock

3.Deux processus ont été créé, on ne reprend pas le controle du terminal

4.pstree -s -p 5304

cela lance le processus xclock et on reprend le controle du terminal, il reste xclock

5.xclock ne se lance pas

Exercice 2

1.La valeur de retour est : 0

La valeur de retour est : 2

2.La valeur de retour est : 130

La valeur de retour (avec KILLINT) est : 147

Exercice 3

1.Dans le premier cas, seul sleep 5 est exécuté, alors que dans le deuxième cas, les deux le sont à la suite

2.Dans le premier cas, seul xclock est exécuté, alors que dans le second, aucun des des deux ne l'est

Exercice 4

1.Quand on compare pareil et même, la veuleur de retour est 0, dans les autres cas c'est 1

2. cmp pareil different 2> /dev/null on remarque que c'est le message affiché quand les messages sont différents

Donc cela correspond à la sortie erreur standard

4. cmp pareil meme 1>/dev/null && echo "les deux fichiers sont identiques"

5.cmp pareil different 1>/dev/null || echo "les deux fichiers sont différents"

6.(cmp pareil meme 1>/dev/null && echo "les deux fichiers sont identiques")&&(cmp pareil different 1>/dev/null || echo "les deux fichiers sont différents")

## TP11

### < exo >

Exercice 1 :

1.Réseau;Station;Trafic;Correspondance\_1;Correspondance\_2;Correspondance\_3;Correspondance\_4;Correspondance\_5;Ville;Arrondissement pour Paris

2.c.grep 'Paris;13' ratp.csv

d.grep '[0-9];13;' ratp.csv

e.grep 'Sceaux' ratp.csv

Exercice 2 :

1.

b. sort -d ratp.csv

c.sort -n ratp.csv | uniq -c | tr a-z A-Z

e.sort -t';' -k 9 ratp.csv | cut -d";" -f9 | sort -n | uniq -c | sort -r -k 1

f.(sort -t';' -k 10 ratp.csv | cut -d";" -f10 | sort -n | uniq -c | sort -r -k 1 )&&(sort -t';' -k 9 ratp.csv | cut -d";" -f9 | sort -n | uniq -c | sort -r -k 1)

2.

a.sort -t';' -k 3 -n -r ratp.csv | uniq -c

b.sort -t';' -k 3 -n -r ratp.csv | uniq -c | head -10

c.sort -t';' -k 2 ratp.csv | uniq -c |grep -v '(RER)'

Exercice 3 :

1.sort -t';' -k 2 ratp.csv | uniq -c | grep '(RER)'

2.sort -t';' -k 2 ratp.csv | uniq -c | grep -v '(RER)' | tr -s ';' | grep '[0-9];[0-9];[0-9];[0-9];[0-9]\*'

3.sort -t';' -k 2 ratp.csv | uniq -c | grep -v '(RER)' |grep '[0-9];[0-9];[0-9];;;;'

4.sort -t';' -k 10 ratp.csv | uniq -c | sort -r -k 1 | grep '[0-9];7;\*Paris' | cut -d';' -f10 | sort -n | uniq -c| sort -r -k 1

dans le 13e

5.grep 'Paris;13' ratp.csv | tr -s ';' |(pas fini)

Exercice 4 :

1.

a.find ~/ -name '\*.java'

d.find ~/ -path '\*/.\*/.\*'

2.find ~/ -path '\*/Documents/Nouveau\*/\*~' -exec rm '{}' ';'

3.find ~/ -name '\*.sh' -exec chmod 766 '{}' ';'

4.find ~/ -name '\*.java' -exec tar -rvf ~/Documents/java.tar '{}' ';'

Exercice 5 :

3.find ~/ -type d -empty

4.find ~/ -size +10M

7.find ~/ -inum -inserer inoeud-

8.find ~/ -perm 700

9.find ~/ ! -user tfranc49 -exec ls -l '{}' ';'