DIU Enseigner l'Informatique au Lycée
Bloc 3

Systèmes d'exploitation

Consignes les exercices ou questions marqués d'un ⋆ devront être d'abord rédigés sur papier (afin de se préparer aux épreuves écrites du partiel et de l'examen). En particulier, il est recommendé d'être dans les même conditions qu'en examen : pas de document ni de calculatrice. Tous les TPs se font sous Linux.

1 Commandes de base

⋆ On considère la séquence de commandes ci-après, rentrées les unes après les autres dans un terminal. Décrire l'effet de chaque commande (création de répertoire, changement de répertoire, affichage dans le terminal, erreur, ...).

1. `cd ~`
2. `mkdir UnixProgWeb`
3. `mkdir UnixProgWeb/TP1`
4. `cd UnixProgWeb/TP1`
5. `cd ..`
6. `ls`
7. `chmod u+rwx,g-rwx,o-rwx TP1`

Réponse: Durée estimée de l'exercice : 10 minutes de recherche au brouillon puis donner la correction. On peut interroger les étudiants dans l'ordre de la salle, 1 réponse par étudiant. Leur laisser 5 à 10 minutes pour tester ces commandes dans le terminal après la correction.

1. `cd ~` : fait du répertoire utilisateur le répertoire courant
2. `mkdir UnixProgWeb` : crée un répertoire `UnixProgWeb` dans le répertoire courant (qui est le répertoire utilisateur). Une erreur peut se produire si le répertoire existe déjà.
3. `mkdir UnixProgWeb/TP1` : crée un répertoire `TP1` dans le répertoire `UnixProgWeb`. Une erreur peut se produire si le répertoire existe déjà.
4. `cd UnixProgWeb/TP1` : `UnixProgWeb/TP1` devient le répertoire courant.
5. `cd ..` : le répertoire parent de vient le répertoire courant (on se trouve donc dans `UnixProgWeb`).
6. `ls` : affiche le contenu du répertoire courant (donc `TP1`).
7. `chmod u+rwx,g-rwx,o-rwx TP1` : change les permissions de `TP1`. Il devient accessible en lecture écriture et "traversable" (on peut rentrer dedans) pour l'utilisateur et ni lisible, ni écrivable ni traversable par les autres.

Ouvrir un terminal et effectuer les commandes ci-dessus. Constatez que tout se déroule comme prévu.

2 Expressions régulières

⋆ pour chacune des expressions régulières ci-dessous, donner une suite de caractères de longueur au moins 1 reconnue par l'expression.

1. `*txt`
2. `+(txt)`
3. `[0-9]*`
4. `+([0-9])`
5. `@(a.txt | b.txt | c.txt)`
6. `+([^0-9])+([0-9])+([^0-9]).bak`
7. `????`
8. `?*?`

Réponse: Durée estimée de l'exercice : 15 20 minutes de recherche au brouillon puis donner la correction. On peut interroger les étudiants dans l'ordre de la salle, 1 réponse par étudiant. Leur laisser 5 à 10 minutes pour tester ces commandes dans le terminal après la correction.

1. `*txt` : L'expression reconnait n'importe quelle mot finissant par `txt` (ex : toto.txt, footxt, txt, ...). Le plus court est `txt`.
2. `+(txt)` : L'expression reconnait une répétition de `txt` (au moins une fois, donc `txttxttxttxt...`). Le mot le plus court est `txt`.
3. `[0-9]*` : reconnait n'importe quel mot qui commence par un chiffre. Les plus courts sont `0`, `1`, ..., `9`.
4. `+([0-9])` : reconnait un mot composé uniquement de chiffres, de longueur au moins 1.
5. `@(a.txt | b.txt | c.txt)` : reconnait exactement l'un des trois choix `a.txt`, `b.txt` ou `c.txt`.
6. `+([^0-9])+([0-9])+([^0-9]).bak` : reconnait un mot constitué d'une suite non-vide de non-chiffres, suivi d'une suite non-vide de chiffres, suivi d'une suite non-vide de non-chiffres, suivi de `.bak`. Par exemple : `abc1def.bak`
7. `????` : reconnait n'importe quel mot de 4 caractères (`toto`, ...)
8. `?*?` : reconnait n'importe quel mot d'au moins 2 caractères (un au début, un à la fin et une chaine éventuellement vide au milieu).

Tester les réponses dans le terminal. Pour cela, se placer dans le répertoire `TP1` créé lors de l'exercice précédent, créer des fichiers vides au moyen de la commande « `touch` *nomdefichier* » et tester l'expression au moyen de « `ls` *expression* ». Le fichier que vous venez de créer doit être listé (au moins lui). Exemple :

```
$ touch toto.txt
$ ls *txt
```

Affiche `toto.txt`.

3 Permissions

⋆ On suppose pour cet exercice que le répertoire courant est le répertoire personnel. De plus, on suppose que les répertoires `UnixProgWeb` et `UnixProgWeb/TP1` existent (cf. exercice 1).

Donner la commande permettant de mettre les permissions demandées. On utilisera des permissions numériques et on justifiera en montrant la représentation binaire.

1. le répertoire personnel possède tous les droits pour l'utilisateur et uniquement le droit d'exécution pour le groupe et les autres
2. les répetoire `UnixProgWeb` et `UnixProgWeb/TP1` possèdent tous les droits pour l'utilisateur et les droits de lecture et d'exécution pour le groupe et les autres (une commande par répertoire)
3. le fichier `lisible.txt` du répertoire `UnixProgWeb/TP1` possède les droits de lecture/écriture pour l'utilisateur et uniquement les droits de lectures pour le groupe et les autres
4. le fichier `secret.txt` du répertoire `UnixProgWeb/TP1` possède les droits de lecture/écriture pour l'utilisateur et aucun droit le groupe et les autres

Mettre les permissions sur tous les fichiers et répertoires comme indiqué ci-dessus (vous pouvez créer 2 fichier `lisible.txt` et `secret.txt` au moyen d'un éditeur de texte). Demander à un voisin (sur une autre machine) de tester les permissions de vos répertoires et fichiers (en essayant de rentrer dans les repertoires et de lire les fichiers). En particulier constater la différence entre droit en exécution et droit en lecture sur un répertoire. Enfin, supprimer les droits pour tout le monde et le groupe sur le répertoire personnel. Constater que plus personne n'a accès à ce répertoire à part le propriétaire.

Réponse: 10 à 15 minutes pour les permissions et idem pour tester les commandes.

1. On suppose que le répertoire utilisateur est le répertoire courant (comme dit dans l'énoncé).

   `chmod 711 .`

   On veut `rwx` pour l'utilisateur et uniquement `x` pour les autres : 111 001 001 (en binaire) donne 711 en octal (permissions numériques). Quelqu'un d'autre que l'utilisateur peut faire « `cd` » dans ce répertoire mais pas « `ls` » (pas de droit en lecture du répertoire). On peut cependant rentrer dans un sous-répertoire si on en connait le nom (par exemple `UnixProgWeb`).

2.

   ```
   chmod 755 UnixProgWeb
   chmod 755 UnixProgWeb/TP1
   ```

   On veut `rwx` pour l'utilisateur et `rx` pour les autres, donc 111 101 101 donne 755. On constate que n'importe qui peut faire `ls` dans ce répertoire (droit en lecture)

3.

   `chmod 644 UnixProgWeb/TP1/lisible.txt`

   Similaire au précédant, on veut `rw` pour l'utilisateur et `r` pour les autres : 110 100 100 donnent 644. Le contenu de ce fichier est lisible par d'autres (par exemple avec `cat lisible.txt`).

4.

   `chmod 600 UnixProgWeb/TP1/secret.txt`

   Similaire au précédant, on veut `rw` pour l'utilisateur et rien pour les autres : 110 000 000 donnent 600. Le contenu de ce fichier n'est pas lisible (par exemple `cat secret.txt` renvoie un message d'erreur pour quelqu'un autre que le propriétaire).

4 Manipulations de fichier textes

Afficher au moyen de la commande `cat` le fichier `/etc/passwd` (ce dernier contient la liste des utilisateurs « locaux » de la machine, *i.e.* pas les utilisateurs dont les comptes sont en réseaux comme les profs ou les étudiants). Pour chacune des actions suivantes, on se référera au cours 1 et éventuellement à la page de manuel de la commande en question si le cours ne donne pas de détail [1]

Remarque il est vivement conséillé d'ouvrir un fichier texte et d'y copier/coller les commandes que vous essayez afin de garder une trace de ce que vous avez fait.

1. afficher les 5 premières lignes du fichier `/etc/passwd`
2. afficher la page de manuel de la commande `tac`
3. utiliser la commande `tac` pour afficher le fichier `/etc/passwd` à l'envers
4. trier le fichier `/etc/passwd`. Quel est l'ordre utilisé ?
5. lire attentivement la page de manuel pour trier selon le troisième champs. On cherchera quelle option permet de spécifier le séparateur de champs et quelle option permet de donner un numéro de champ. Quel est l'ordre utilisé ?
6. trier selon le même champ que la question précédante mais en utilisant l'ordre numérique (chercher dans la page de manuel). En quoi l'ordre est-il modifié ? donner deux lignes dont l'ordre l'une par rapport à l'autre à changé.

Réponse:

1. `head -n 5 /etc/passwd`

1. si la page de manuel n'est pas disponible sur votre machine, vous pouvez consulter l'url suivante : `http://www.linux-france.org/article/man-fr/man1/Index-1.html`

2. `man tac` (touche « q » pour quitter)
3. `tac /etc/passwd`
4. `sort /etc/passwd`. L'ordre utilisé est l'ordre lexicographique (ordre du dictionaire).
5. `sort -t : -k 3 /etc/passwd`. C'est toujours l'ordre lexicographique qui est utilisé
6. `sort -t : -k 3 -n /etc/passwd`. C'est l'ordre numérique qui est utilisé. En particulier la ligne dont le troisième champ vaut 2 se trouve maintenant *avant* celle dont le troisième champ vaut 10, 100 ou 1000.

Récupérer le fichier se trouvant à l'adresse suivante :

`https://www.lri.fr/~kn/teaching/upw/td01/cdm.txt`

(en utilisant un navigateur Web) et le sauver dans le repértoire `TP1`. Afficher le contenu du fichier. Ce dernier contient des listes de matches de phase finale de coupe du monde, sous la forme :

`PAYS1:PAYS2:GROUPE`

(où `GROUPE` est une lettre entre `A` et `H`). On souhaite déterminer combien d'équipe distinctes sont dans le fichier.

1. lire la page de manuel de la commande `cut`. Déterminer commant extraire uniquement la première « colonne » du fichier `cdm.txt` (*i.e.* déterminer commant afficher uniquement le premier champ de chaque ligne, en spécifiant que le délimiteur de champ est «: »
2. Sauver le résultat de la première commande dans un fichier `liste1.txt` on peut sauver la sortie d'une ligne de commande en écrivant `commande arg1 ... argn > fichier`. Sauver de la même manière la deuxième colonne du fichier `cdm.txt` dans un fichier `liste2.txt`
3. sachant que le « | » permet d'enchaîner deux commandes en passant la sortie de la première comme entrée à la seconde, utilisez dans un premier temps la commande `cat` pour afficher les fichiers `liste1.txt` et `liste2.txt` l'un à la suite de l'autre puis enchainer le résultat avec la commande `sort`.
4. consulter (encore) la page de manuel de `sort` pour trouver l'option qui permet de supprimer les doublons. Afficher la liste des pays sans doublons
5. consulter la page de manuel de la commande `wc` (*word count*) pour savoir comment compter le nombre de lignes d'un fichier. En déduire rajouter la bonne invocation de la commande `wc` à votre enchainement pour déterminer le nombre de pays.

Réponse:

1. `cut -f 1 -d : cdm.txt`
2. `cut -f 1 -d : cdm.txt > liste1.txt` et `cut -f 2 -d : cdm.txt > liste2.txt`
3. `cat liste[1-2].txt | sort` (ou `cat liste1.txt liste2.txt | sort`)
4. `cat liste[1-2].txt | sort -u`
5. `cat liste[1-2].txt | sort -u | wc -l`

5 Recherche de fichiers

On suppose que le répertoire courant est le répertoire utilisateur et que les fichiers des exercices précédents n'ont pas été effacés (`liste1.txt`, `liste2.txt`, `secret.txt`, ...).

1. Utiliser la commande `find` pour trouver tous les fichiers dont l'extension est `.txt` et se trouvant quelque part dans le répertoire utilisateur (ou un sous-répertoire).
2. Lire la page de manuel de la commande `xargs`. Cette dernière prend la forme `xargs commande` et lit en plus *n* noms de fichiers sur son entrée standard qu'elle passe comme arguments à commande. Enchainer (au moyen de « | ») les commandes `find` et `xargs cat` pour afficher dans le terminal toutes les lignes de tous le fichiers texte.

3. Comment compter le nombre de lignes ainsi affichées ?

Réponse:

1. `find . -name '*.txt'`
   Les guillemets sont importants car sans ces derniers, l'étoile (*) est interprétée par le *shell*, qui la remplacera par les noms de fichiers en `.txt` se trouvant dans le répertoire courant.
2. `find . -name '*.txt' | xargs cat`
   Si des noms de fichiers ou de répertoire contiennent des espaces, cette commande ne fonctionnera pas bien. Il faudra alors utiliser la version :
   `find . -name '*.txt' -print0 | xargs -0 cat`
   qui utilise un caractère spécial pour séparer les arguments, plutot que des blancs (fonctionnement par défaut).
3. en réutilisant l'exercice précédent :
   `find . -name '*.txt' | xargs cat | wc -l`

6 Redirections

Exercice 1

⋆ Pour chacune des questions suivantes, donner les commandes permettant de réaliser les actions demandées. On cherchera d'abord les commandes en s'aidant du cours ou des pages de manuel.

1. Se placer dans le répertoire `UnixProgWeb` créé au TP1. Créer un sous-répertoire TP2 et se placer dedans.

Réponse: Le `$` représente le prompt du *shell* :

```
$ cd UnixProgWeb
$ mkdir TP2
$ cd TP2
```

2. Lister de façon détaillée le contenu du répertoire. Dire maintenant comment créer un fichier `liste.txt` contenant cette liste détaillée.

Réponse:

```
$ ls -l
$ ls -l > liste.txt
```

3. Lister de façon détaillée les fichiers `liste.txt` ainsi qu'un fichier `pasla.txt` (inexistant). Quel est l'affichage de la commande ?

Réponse:

```
$ ls -l liste.txt pasla.txt
ls: cannot access pasla.txt: No such file or directory
-rw-rw-r-- 1 kim kim 13834 Sep 20 09:50 liste.txt
```

(La ligne concernant le fichier `liste.txt` est bien sur différente pour chacun).

4. Effectuer la même commande en redirigeant les erreurs vers un fichier `erreur.txt` et la sortie standard vers un fichier `liste2.txt`. Afficher tour à tour (en 2 commandes) le contenu de ces fichiers.

Réponse:

```
$ ls -l liste.txt pasla.txt 2> erreur.txt > liste2.txt
$ cat erreur.txt
$ cat liste2.txt
```

Exercice 2

La commande dc implémente une calculatrice en notation *polonaise inversée* (les opérandes sont placées sur une pile et les opérations dépilent les opérandes et empilent le résultat). Elle attend des ordres de calcul sur son entrée standard et les exécute. Par exemple (les lignes en *italique* sont celles à taper au clavier, les lignes en **gras** sont celles affichées par le programme) :

```
$ dc
42
41
+
p
83
```

← invocation de la commande
← place l'entier 42 sur la pile
← place l'entier 41 sur la pile
← dépile deux nombres et place leur somme sur la pile
← affiche le sommet de pile

1. au moyen d'un éditeur de texte, créer un fichier `calculs.txt` contenant les lignes :

```
40
260
+
10
/
p
12
*
p
```

2. ⋆ Que serait l'affichage produit si cette série de commandes étaient données à dc. Donner *deux* lignes de commandes différentes permettant de passer ce fichier en entrée à dc.

Réponse: La suite de commande doit afficher 30 (résultat de (40 + 260)/10) puis 360 (résultat de 30 * 12).

```
$ cat calculs.txt | dc
$ dc < calculs.txt
```

7 Gestion des processus

Sous Unix, les processus peuvent avoir l'état suivant :

```
R       en cours d'exécution
D       en attente, non interruptible (généralement en train de faire une entrée/sortie)
S       en attente, interruptible
T       arrêté
Z     « zombie », processus défectueux, terminé mais dont les ressources ne sont pas encore libérées
```

Il peut de plus y avoir des informations complémentaires :

```
+       le processus est en avant plan
s       le processus est un leader de session (usuellement c'est le shell)
l       le processus est en fait un thread (sous-processus)
```

⋆ Fermez toutes vos fenêtres et ne conservez qu'une seule fenêtre de terminal. Tapez la commande `ps u`. Quels sont les programmes en cours d'exécution et quels sont leurs états.

Réponse: Normalement on doit voir la commande `ps` en cours d'exécution avec l'état `R+` et le shell avec l'état `Ss`. La commande `ps` étant celle qui collecte les informations sur les processus en cours d'exécution, c'est elle qui apparait comme « en cours d'exécution » dans l'affichage. Le `+` indique que pendant la (brève) période où la commande s'exécute, le shell n'a pas rendu la main et la commande est en avant plan.

La commande `bash` a le mode `Ss` car elle est en suspend pendant que `ps` s'exécute et le programme est leader de session.

8 Scripts *bash*

Exercice 4

On considère le script suivant :

```
#!/bin/bash
for i in $(seq 1 10)
do
  if test -d "TP$i"
  then
     echo "TP$i" "existe déjà"
  else
     echo "création de TP$i"
     mkdir "TP$i"
  fi
done
```

1. Créez un fichier (par exemple `script.sh`) dans le répertoire `UnixProgWeb`. Placez dans ce fichier le contenu du script ci-dessus (la première ligne du script doit être celle commençant par `#!...` sans ligne vide ou espace avant. Essayez d'exécuter le script en effectuant la commande : `./script.sh` (le répertoire courant dans votre terminal doit être `UnixProgWeb`). Que se passe-t-il ? Comment régler le problème ? Réglez le problème et exécutez la commande.

Réponse: Le script n'est pas *exécutable*. Il faut le rendre exécutable au moyen de la commande `chmod a+x script.sh` ou `chmod 755 script.sh`.

2. Commentez ce script (les commentaires en shell commencent par un # et finissent à la fin de la ligne), en décrivant **précisément** ce qui se passe à chaque ligne.

Réponse:

```
#!/bin/bash
               # la ligne ci-dessus indique au système quel
               # programme utiliser pour interpréter le script.
for i in $(seq 1 10)  # la commande seq 1 10 renvoie 10 chaines de
                      # caractères 1 2 3 ... 10.
               # le $( ... ) evalue la commande et la sortie
               # standard sous forme de chaine. La variable i
               # va prendre tour a tour les valeurs 1 ... 10
do
  if test -d "TP$i"   # "TP$i" est la chaine TP1 TP2 ...
               # test -d  teste si son argument est un répertoire
   then
      echo "TP$i" "existe déjà" # on affiche un message
   else
      echo "création de TP$i"
     mkdir "TP$i"            # mkdir crée un répertoire
   fi
done
```

Exercice 5

On souhaite écrire un script bash qui automatise le téléchargement de fichiers à une URL connue.

1. Placez vous dans le répertoire `UnixProgWeb/TP2`. Récupérez **sans utiliser de navigateur web** le fichier se trouvant à l'url « `https://www.lri.fr/~kn/php/cm/pays.txt` ». Pour ce faire, utilisez la commande `wget` de cette manière :

   ```
   $ wget -nd -nc -q "https://www.lri.fr/~kn/php/cm/pays.txt"
   ```

   L'option `-nd` récupère le fichier et ne crée pas les répertoires `kn/php/...` dans le répertoire courant. L'option `-nc` évite de retélécharger le fichier s'il est déjà dans le répertoire courant. L'option `-q` supprime les messages de débuggage de la commande. Constatez que le fichier a bien été rappatrié dans le répertoire courant et affichez son contenu avec `cat`.
2. Donner une commande permettant de récupérer le troisième champ (séparé par des « `|` ») du fichier `pays.txt` (attention le caractère « `|` » est un caractère spécial, il convient de l'échapper convenablement sur la ligne de commande).

Réponse:

```
$ cut -f 3 -d "|" pays.txt
```

3. Écrire un script shell qui télécharges tous les fichiers d'images listés dans le fichier `pays.txt` (le troisième champ donné par la question précédante), sachant qu'ils se trouvent à l'adresse : « `https://www.lri.fr/~kn/php/cm/`*fichier.png* ». Lancez votre script et vérifier que les images sont bien rappatriées. Si le fichier existe déjà, votre script doit le supprimer avant de le retélécharger.

Réponse:

```
#!/bin/bash

for i in $(cut -f 3 -d "|" pays.txt)
do
  if test -f "$i"
  then
     rm -f "$i"
  fi
  wget -nc -nd -q "https://www.lri.fr/~kn/php/cm/$i"
done
```

4. (facultatif) Modifier le script précédant pour qu'il affiche en fin d'exécution le nombre de fichier téléchargés ainsi que la somme des tailles des fichiers téléchargés (en octets).

Réponse:

```
#!/bin/bash
COUNT=0
SIZE=0
for i in $(cut -f 3 -d "|" pays.txt)
do
  if test -f "$i"
  then
     rm -f "$i"
  fi
  wget -nc -nd -q "https://www.lri.fr/~kn/php/cm/$i"
  COUNT=$(($COUNT + 1))
  T=$(wc -c "$i" | cut -f 1 -d ' ')
  SIZE=$(( $SIZE + $T))
done
echo "$COUNT fichier"
echo "$SIZE octets"
```