TP0 : Découverte du Shell Linux/Unix

1. Objectifs

Le shell bash contient de nombreuses fonctionnalités et dispose donc de nombreuses commandes disponibles. Ce TP couvre les bases de la navigation dans le système de fichiers Linux et du travail avec des fichiers et des répertoires. La gestion des fichiers et des répertoires est une fonctionnalité majeure du shell Linux. Ainsi ce TP explore les commandes de gestion du système Linux, vous montrant comment jeter un coup d'œil à l'intérieur de votre système Linux à l'aide de ligne de commande. Après cela, nous vous montrons quelques commandes pratiques que vous pouvez utiliser pour travailler avec des fichiers de données sur le système

Prenez le temps de bien lire le sujet et d'effectuer les manipulations demandées ou d'explorer un peu plus par vous-même. N'hésitez pas à poser des questions à l'enseignant si vous souhaitez approfondir certains points.

> **Rédiger un aide-mémoire ou un compte rendu pour chaque séance de TP, il vous sera utile pour vos révisions et lors du contrôle de synthèse.**

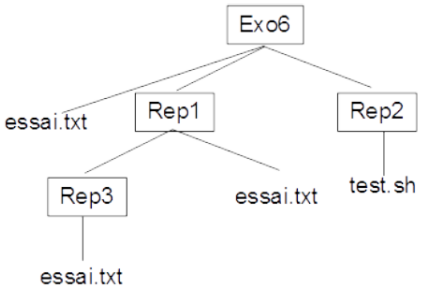
2. Arborescence

La plupart des systèmes Linux sont organisés selon la même structure. Le système de fichiers est une structure arborescente. À la *racine* se trouve le répertoire «/» . Ce répertoire contient un ensemble de sous-répertoires, comme `bin`, `lib`, `etc`, `tmp`, qui peuvent contenir eux-mêmes d'autres sous-répertoires. Pour désigner un fichier ou un répertoire, on utilise son *chemin d'accès*. Pour former un chemin, on liste les répertoires à « traverser » en utilisant comme séparateur le caractère `/`
Un chemin peut être *absolu* s'il part de la racine `/` de l'arborescence (exemple : `/usr/local/bin`) ou relatif s'il part de la position courante dans l'arborescence (exemple : `bin` si l'on est positionné dans le répertoire `/usr/local`).

1. Ouvrez un Terminal et tapez les commandes **`ls`** , **`ls`** `-l`, **`ls`** `-a`, **`ls`** `-al`, **`ls`** `-ld`, **`ls`** `-R` Interprétez.
2. Taper la commande **`id`**, en consultant le manuel de cette commande interpréter le résultat affiché.
3. Créez un répertoire `SMARTICT1` s'il n'est pas créé, puis un sous-répertoire `Tp0` au moyen de la commande `mkdir`
4. Placez-vous dans ce dernier répertoire avec la commande `cd` puis tapez `pwd` pour vérifier votre position dans l'arborescence.
5. Tapez **`cd`** `..` puis de nouveau **`pwd`** Que représente le répertoire `..` ?
6. Tapez **`cd`** `~` puis de nouveau **`pwd`** Que représente le répertoire `~` ?
7. Tapez **`cd`** `/usr/bin` puis de nouveau **`pwd`** Que représente le répertoire `/usr/bin` ?
8. Tapez **`cd`** `../../usr/bin` puis de nouveau **`pwd`** Que représente le répertoire `../../usr/bin` ?
9. Placez-vous dans le répertoire `SMARTICT1` et tapez la commande **`mkdir`** `../SMARTICT1/Test1` quelle différence avec la commande **`mkdir`** `Test1` ?
10. Placez-vous dans le répertoire `SMARTICT1` et tapez la commande **`ls -l`** `../../SMARTICT1/bin` quelle différence avec la commande **`ls -l`** `/bin` ?

3. Manipulations élémentaires sur les fichiers

1. Repositionnez-vous sous `SMARTICT1/Tp0` et tapez `gedit` à l'invite du Shell : la fenêtre d'un *éditeur de texte* s'ouvre alors. Saisissez une quarantaine de lignes, enregistrez le résultat dans un fichier, puis fermez `gedit`.
2. Affichez le contenu de ce fichier, puis du fichier `/etc/passwd` au moyen des commandes **`cat`**, **`less`** et **`more`**
3. Dupliquez ce fichier sous un autre nom en utilisant la commande **`cp`**. Observez le résultat au moyen de **`ls`**.
4. Supprimez la copie au moyen de la commande **`rm`**
5. Déplacez (renommez) le fichier initial au moyen de la commande `mv` et contrôlez le résultat.
6. En se plaçant dans le répertoire `SMARTICT1/Tp0` et n'utilisant que les commandes `touch`, `mkdir`, et la notion de chemin relatif, créer l'arborescence ci-contre :



7. Les alias sont utilisés pour créer des raccourcis de certaines commandes longues. La syntaxe de base est : **`alias`** `votreRacourcis='Commande à remplacer'`
   Exemple : `alias` `affiche='ls -l'`
   Créer un alias qui affiche tous les fichiers d'un répertoire même ceux qui sont cachés.

4. Droits d'accès par défaut (travailler sur /tmp)

1. Pour la suite des questions, travaillez dans le répertoire `/tmp`
2. Tapez **`umask`** pour voir votre masque courant. Quels sont donc les droits par défaut ?
3. Tapez **`umask`** `0124 ; mkdir T ; ls -l`.
4. Décomposer `124` sous la forme `rwx/rwx/rwx`, quel est le lien avec les droits du répertoire créé ?
5. Créez et sauvegardez un nouveau fichier texte avec un éditeur de texte lancé depuis ce Shell. Observez ses droits.
6. Créez et sauvegardez un nouveau texte, mais avec un éditeur lancé depuis un autre Shell. Quels sont ses droits ?
7. Restaurez votre ancien masque.
8. Proposez deux méthodes permettant de créer dans un répertoire, 5 fichiers qui ne peuvent être lus et modifiés que par l'utilisateur ?

5. Utilisateurs et droits d'accès

Linux/Unix est un système multiutilisateur. Pour assurer l'intégrité du système et une certaine confidentialité des données, chaque fichier appartient à l'utilisateur qui l'a créé et il est possible de définir si les autres utilisateurs du système ont le droit de le lire, de le modifier ou de l'exécuter dans le cas où il s'agit d'un programme.
La commande **`ls`** `-l` vous permet de visualiser les droits d'accès associés à chaque fichier ainsi que son possesseur.

Chaque utilisateur possède un répertoire dont il propriétaire. Il s'agit de son *répertoire maison* (*home directory*) qui est ici accessible à `/Utilisateurs/nom` où `nom` est le *login* de l'utilisateur.

Le reste du système appartient en général à un *super-utilisateur* : `root`. C'est *l'administrateur* du système. Parfois il peut être intéressant de cacher un fichier à la majorité des utilisateurs sauf à une certaine catégorie de personnes. C'est pourquoi les utilisateurs peuvent appartenir à des groupes.

Chaque fichier appartient aussi à un groupe et il est possible de définir pour chaque fichier des permissions relatives à son possesseur, à son groupe et au « reste du monde ».

1. Toujours sous `SMARTICT1/Tp0`, tapez la commande `ls -l` ; observez le résultat. Tapez aussi **`ls`** `-l /usr` Identifiez les colonnes indiquant le possesseur du fichier, le groupe du fichier et les permissions du fichier.
2. Supprimez votre autorisation en lecture sur le fichier texte créé précédemment au moyen de la commande `chmod`
3. Essayez de lire le contenu du fichier au moyen de la commande **`cat`** ou à l'aide de **`gedit`**.
4. Restaurez l'autorisation de lecture. Réessayez de l'ouvrir avec **`gedit`**.
5. Essayez de supprimer la permission d'exécution pour vous sur le répertoire `SMARTICT1` créé précédemment.
6. Essayez ensuite de vous y positionner au moyen de la commande `cd`. Rétablissez les permissions initiales et repositionnez-vous dans `SMARTICT1/Tp0`.

6. Entrées/sorties

- Pour modifier l'entrée standard (clavier de descripteur 0), on utilise l'opérateur < . Par exemple, pour utiliser le fichier `entrees.txt` à la place du clavier dans la commande `maCommande`, on utilisera la syntaxe suivante : `maCommande < entrees.txt`
- Pour modifier la sortie standard (terminal de descripteur 1), on utilise l'opérateur >. Par exemple, pour stocker les résultats de la commande `maCommande` dans le fichier `sortie.txt` on utilisera la syntaxe : `maCommande > sortie.txt`
- Pour rediriger la sortie d'erreur standard (descripteur 2), on utilise l'opérateur 2>. Par exemple, pour stocker les messages d'erreurs dans un fichier au lieu de la sortie standard, on utilisera la syntaxe : `maCommande 2> erreurs.txt`

Un autre opérateur permet de faire communiquer deux commandes en redirigeant la sortie de l'une sur l'entrée de l'autre.

- Il s'agit de l'opérateur **`|`** (*tube* ou *pipe* en anglais), sous la forme `commande1` **`|`** `commande2`

Le *remplacement de commandes* consiste à substituer, dans la ligne de commande, le texte de la commande par les résultats qu'elle a produits sur la sortie standard. Ce mécanisme permet d'utiliser les sorties d'une commande comme argument d'une autre commande.

- Syntaxe : **`$(`**`commande`**`)`** ou `` `commande` `` (*backquotes*)

Toute commande Linux/Unix émet en se terminant un code de retour numérique (*`exit status`*) compris entre `0` et `255`. La valeur `0` correspond à une exécution « réussie » de la commande. Les autres valeurs (de `1` à `255`) correspondent à différents cas d'erreurs d'exécution possibles. Ce mécanisme sera surtout utile lors de l'écriture de scripts, pour enchainer correctement des actions ou effectuer des tests.

1. Analysez et expliquez les commandes suivantes (remplacer `anne` par votre `login`) :

```
who
who | grep anne
who | grep anne > ficSortie
cat ficSortie
who | grep anne > /dev/null
cat /dev/null
who | grep anne > /dev/null ; echo $?
! who | grep anne > /dev/null ; echo $?
```

2. Écrire une commande composée qui affiche `1` si le fichier `fich` existe dans le répertoire en cours et `0` sinon.

7. Lien physique et lien symbolique

Le chemin d'accès à un fichier permet au système Linux/UNIX de l'identifier et de le localiser sans équivoque. Les noms de fichiers sont mémorisés dans des répertoires et, dans Linux/UNIX, s'appellent des *liens matériels.* À chaque lien d'un répertoire est associé une valeur numérique appelée *numéro de l'inode* qui identifie le fichier associé à ce lien. L'*inode* contient le type du fichier, les droits, le propriétaire, le groupe, la taille en octet du fichier, le nombre de liens matériels, etc. Il est possible qu'il existe dans l'arborescence des fichiers d'un disque plusieurs liens distincts qui désignent le même fichier.
Il suffit qu'on leur ait associé le même numéro d'inode. Les liens matériels permettent aux utilisateurs de créer des noms, mémorisés dans leur répertoire de travail, qui leur évite d'avoir à utiliser des chemins complexes. Créer un lien, c'est créer un nouveau nom pour un fichier déjà existant.

- La commande `ln`, qui a la même syntaxe que `cp`, se contente de créer des liens plutôt que de créer des fichiers par copie.
- Syntaxe : **`ln`** `fichierPointé nomDuLien` ou **`ln`** `repertoirePointié nomDuLien`
- La commande `rm` détruit un lien. Elle ne détruit le fichier, c'est-à-dire l'*inode*, que si le nombre de liens matériels, inscrits dans l'*inode*, devient nul, ce qui signifie que l'on vient de supprimer le dernier lien sur le fichier.
- La commande `mv` change le nom d'un fichier si les arguments sont dans un même répertoire, elle les transfère sinon.
- Les noms (liens) sont supprimés du répertoire origine pour être inscrits dans le répertoire destination.
- La commande `mv` permet de transférer des fichiers « normaux » aussi bien que des répertoires.
- L'option `-i` de la commande `ls` permet d'afficher le numéro de mode d'un fichier.
- L'option `-l` affiche le nombre de liens matériels. C'est le nombre qui vient immédiatement après les droits.

Il existe cependant deux restrictions à l'usage des liens matériels :
  - Il est impossible de créer un lien matériel sur un répertoire.
  - Il est impossible de créer un lien matériel d'un disque à un autre.

Le *lien symbolique* remédie à cet inconvénient parce qu'un *lien symbolique* est un fichier spécial à part. Il possède son propre mode et contient, comme uniques données, le chemin d'accès au fichier lié.

La commande **`ln`** `-s` qui crée un lien symbolique, ne vérifie pas que le fichier lié existe, puisque celui-ci peut se trouver sur un disque amovible non monté à cet instant.

1. Exécuter dans le répertoire `SMARTICT1/Tp0` les commandes

```
cp /etc/passwd passwd
ln passwd passwd2
```

Que se passe-t-il si l'on tente de modifier `passwd2` ?

2. Quel effet a la commande suivante ? **`ln`** `passwd passwd3`
Comment peut-on retrouver tous les liens de `passwd` ?

3. Exécuter ensuite la commande **`rm`** `passwd`
Que se passe-t-il ? Le fichier associé est-il encore accessible ? Peut-on recréer le lien `passwd` et comment ?

4. Exécuter maintenant les commandes :

```
ln -s passwd passwd4
ln -s passwd4 passwd5
ls -il passwd*
```

Décrire le résultat de l'affichage de la commande `ls`

5. Taper enfin

```
rm passwd4
cat passwd5
```

Quel genre de message d'erreur recevez-vous, et pourquoi ?

6. En utilisant les commandes `touch`, `mkdir`, `cp`, `ln`, créer la structure arborescente suivante dans votre répertoire de travail :

```
ls -liR RepExo6/
RepExo6/ :
total 16
981332 -rw-r--r-- 3 ... ... 15 nov 8 10:54 fich1
981332 -rw-r--r-- 3 ... ... 15 nov 8 10:54 fich2
981338 lrwxrwxrwx 1 ... ... 8 nov 8 11:00 essai1 -> Rep/essai
981334 drwxr-xr-x 2 ... ... 4096 nov 8 10:59 Rep
Rep/ :
total 12
981336 -rw-r--r-- 1 ... ... 20 nov 8 10:57 fich1
981332 -rw-r--r-- 3 ... ... 15 nov 8 10:54 fich3
981335 -rw-r--r-- 1 ... ... 30 nov 8 10:56 essai
```

8. Expressions régulières et commande **grep**

1. En lisant le manuel de la commande `grep`, expliquer les grandes lignes de l'utilisation de cette commande.
2. Quelle option permet d'afficher les numéros de lignes trouvées ?
3. Quelle option permet d'ignorer la case (majuscule ou minuscule) ?
4. Quelle option permet de n'afficher que le nombre de lignes trouvées ?

L'expression à rechercher est décrite sous la forme d'une *expression régulière*. Les expressions régulières permettent de décrire des motifs formés de caractères. Ce mécanisme est similaire à celui de génération de noms de fichiers par le Shell (jokers), mais pas exactement identique.

Exemple : Recherche toutes les chaînes de caractères alphabétiques dans tous les fichiers dont les noms commencent par une majuscule et se terminent par `.f`

```
grep  '[a-zA-Z]'          [A-Z]*.f
```

expression régulière — génération de nom de fichier

Les jokers décrivent de façon générique des **noms** de fichiers, alors que les expressions régulières représentent une description générique de chaînes de caractères **contenues dans** un fichier (ou un flot de caractères).

Différences d'interprétation des caractères spéciaux

	Expressions régulières	**Jokers (noms de fichier)**
`?`	le caractère `?`	un caractère quelconque, sauf `<new-line>`
`.`	un caractère quelconque sauf `<new-line>`	Le caractère point, sauf s'il est en début de ligne en Shell
`*`	remplace `0` fois ou `n` fois le caractère qui précède `*`	zéro ou un nombre quelconque de caractères
`[a-i]`	un caractère entre `a` et `i`	un caractère entre `a` et `i`
`[!a-i]`	un caractère entre `a` et `i`, ou `!`	un caractère qui n'est pas entre `a` et `i`
`[^a-i]`	un caractère qui n'est pas entre `a` et `i`	un caractère entre `a` et `i`, ou `^`
`\`	banalise le caractère spécial qui suit	banalise le caractère spécial qui suit
`^`	ce qui suit est en début de ligne	le caractère `^`
`$`	ce qui précède est en fin de ligne	référence une variable

Exemples d'expressions régulières :

`toto`	contient la chaîne de caractère `toto`
`^toto toto`	en début de ligne
`toto$ toto`	en fin de ligne
`^toto$`	une ligne ne contenant que `toto`
`t[aeiouy]to`	la seconde lettre est une voyelle
`t[^aeiouy]to`	la seconde lettre n'est pas une voyelle
`t.to`	la seconde lettre est n'importe quel caractère
`^…$`	une ligne avec exactement 3 caractères
`^\.`	ligne débutant par un point

`\.[a-z][a-z]`	ligne débutant par un point suivi de deux minuscules
`^[^.]`	ligne ne débutant pas par un point.
`totos*`	ligne contenant `toto` ou `totos`, `totoss`, `totosss`, …
`"*toto"*`	ligne avec `"toto"` avec ou sans guillemets
`[A-Z] [A-Z]*`	ligne avec une ou plusieurs majuscules

L'expression régulière peut être détectée un certain nombre de fois dans la ligne :

`\{nombre\}`	nombre exact
`\{nombre,\}`	nombre minimum
`\{nombre1,nombre2\}`	entre `nombre1` et `nombre2`

Exemple : rechercher toutes les lignes se terminant par deux chiffres dans les fichiers du répertoire courant

```
grep '[0-9]\{2\}$' *
```

Pour que le rôle des caractères spéciaux soit ignoré par le Shell, il faut les « protéger ». Il existe trois mécanismes qui le permettent, aussi bien pour la génération des noms de fichiers que pour les expressions régulières :

- Le caractère `\` protège le caractère qui le suit immédiatement.
- Tous les caractères encadrés par `'` sont protégés, sauf le `'` lui-même qui sert de délimiteur.
- Tous les caractères encadrés par `"` sont protégés, sauf les caractères `$`, `` ` `` et `\` qui conservent leur signification pour le Shell ainsi que le caractère `"` qui sert de délimiteur. Ainsi pour éviter que les caractères spéciaux de l'expression régulière du `grep` ne soient traités par le Shell et non le `grep` lui-même, on encadre généralement cette expression régulière par des `'` .

5. Trouvez dans le fichier `/etc/passwd` la ou les lignes contenant le motif `mail`.
6. Trouvez dans le fichier `/etc/passwd` les numéros des lignes correspondant au motif `mail`.
7. À quoi sert la commande composée suivante : `ls -lR | grep '^d' | wc -l`
8. Affichez le nombre de commandes externes commençant par `l`.
9. Affichez la liste que des fichiers cachés de votre répertoire maison (fichiers commençants par **.**)
10. Trouvez les lignes de votre fichier historique qui contiennent la commande **`ls`** ainsi que leur numéro. (le fichier historique est le fichier `.bash_history` de votre répertoire maison)
11. Listez tous les fichiers du répertoire /`tmp` qui ont des droits d'accès en écriture pour tous les utilisateurs.
12. Exécuter la commande suivante :

```
echo -e "Bonjour.gif\n Bonjour,gif\n Bonjourgif\n" > testTP
```

13. Interpréter les commandes suivantes :

```
grep "\.gif" testTP
grep ".gif" testTP
```

14. Chercher dans un fichier les mots *Monsieur* et *Madame* et afficher le numéro de ligne où se trouvant ces mots recherchés
15. Chercher dans un fichier les numéros de téléphone au format `xx xx xx xx xx`

- **Caractères de remplacement (jokers) :**

Le Shell permet d'utiliser des caractères spéciaux dans les noms de fichiers (ou de répertoires), pour pouvoir effectuer une commande portant sur une sélection de fichiers (ou de répertoires) dont le nom vérifie un critère. On utilise par exemple :

`*`	pour remplacer n'importe quelle suite de caractères
`?`	pour remplacer un seul caractère
`[abc]`	pour remplacer soit `a`, soit `b`, soit c
`[!abc]`	pour remplacer n'importe quels caractères sauf `a`, `b` ou `c`
`[A-Z]`	pour remplacer toutes les lettres majuscules

On peut combiner ces « jokers ». Par exemple `ls *a?.[co]` listera tous les fichiers du répertoire courant, dont le nom, contient la lettre `a` en avant dernière lettre et qui se terminent par l'extension `.c` ou bien `.o`
1. Listez dans le répertoire `/bin` les différents Shells disponibles sur votre système (ce sont les fichiers dont les noms comportent les deux lettres successives `sh` au début, en fin ou au milieu)

2. Listez dans le répertoire `/dev` les fichiers dont le nom comporte exactement 4 caractères, le dernier étant un chiffre.

9. Gestion des processus

Les *processus* représentent les programmes actuellement en cours d'exécution sur votre système. Comme les fichiers, ces processus appartiennent à un utilisateur (celui qui a demandé l'exécution du programme).

1. Tapez la commande **`ps`** et donnez le numéro du processus `ps`.
- Tapez la commande **`ps`** `l` et donnez le numéro du processus père de `ps`.
- Tapez la commande **`ps`** `aux` et donnez les noms des processus appartenant à l'utilisateur.
- Tapez la commande **`pstree`** et interprétez.

2. Affichez le nombre de processus qui vous appartiennent (**utilisez un tube et un filtre**).

```
#!/bin/bash
i=1
while [ $i -le 100 ]
do
   echo "i = $i"
   i=$(($i + 1))
   sleep 1
done
```

3. Lancer une nouvelle console de façon à disposer de deux consoles.
- Placez-vous dans le répertoire `/tmp` et lancez `gedit`
- Tapez le texte ci-contre, enregistrez le sous le nom `boucle`, puis quitter `gedit`, (Les explications des diverses instructions seront vues dans les prochaines TPs)
- Tapez `./boucle` que se passe-t-il ?
- Positionnez les droits en exécution sur ce fichier puis recommencez
- Interrompez le programme avec `Ctrl-c`.

4. Relancez le programme et interrompez-le par `Ctrl-z`. Tapez `ps` Que lit-on ?
- Relancez le programme par `fg` (*foreground*) puis interrompez-le à nouveau par `Ctrl-z`
- Tapez `bg` (*background*). Maintenant, le programme s'exécute en *tâche de fond*.

5. Tuez le programme par la commande **`kill`** `-9` *`pid`* que signifie l'option `-9` ? (le *pid* est le numéro de processus et peut être obtenu par la commande `ps`)

6. Lancez le programme en tâche de fond par la commande `./boucle &` puis tapez **`ps`**. Interprétez.

7. Lancez la commande **`top`**. Que fait cette commande ?
- appuyer sur la touche `h` pour accéder à l'aide,
- tuez le processus en tâche de fond précédent à l'aide de `top`. Quittez `top` (touche `q`).

8. Lancez un nouveau terminal,
- récupérer le PID du deuxième terminal, en utilisant la commande **`ps`**, récupérer son numéro `PID`,
- avec la commande **`kill`**, envoyer le signal `9` à ce processus.

9. Sur le même poste de travail, lancez une nouvelle session utilisateur,
- en utilisant la commande **`ps`**, récupérer son numéro `PID`,
- à partir de la première session, utiliser la commande **`kill`**, pour vous déconnecter de la nouvelle session.

10. Demander à votre voisin d'ouvrir une nouvelle session à son nom sur votre machine,
- lister les processus qui appartiennent à votre voisin,
- récupérer le PID de son shell,
- avec la commande **`kill`**, envoyer le signal `9` à ce processus, interpréter le message obtenu.

10. Les filtres

Un *filtre* est une commande qui effectue un **traitement sur du texte**. Le résultat du filtre, également du texte, est affiché sur la sortie standard (le terminal par défaut). Si on précise en argument un nom de fichier, le filtre traitera son contenu, sinon il utilisera l'entrée standard (clavier par défaut).

Voici quelques filtres simples :

- **`cat`** affiche un texte directement sur la sortie standard
- **`more`** affiche un texte avec déplacement de haut en bas
- **`less`** affiche un texte avec navigation possible dans les deux sens
- **`sort`** effectue un tri ou une fusion
- **`wc`** compte le nombre de lignes, de mots et/ou de caractères
- **`head`** affiche les premières lignes
- **`tail`** affiche les dernières lignes
- **`nl`** numérote les lignes
- **`tr`** remplace ou élimine des caractères
- **`cut`** permets d'afficher des zones spécifiques d'un fichier

1. Utilisez la documentation (`man`) pour obtenir des détails sur la syntaxe, la sémantique et les options de ces commandes.
La commande `cat` est un filtre élémentaire. Elle peut également être utilisée sans argument :
2. Taper `cat`
Tapez quelques lignes, puis appuyez sur `CTRL+D`. Que se passe-t-il ?

3. En utilisant une de ces commandes, donner la commande qui :

- ✓ affiche les 6 premières lignes, puis les 10 dernières lignes du fichier `/etc/passwd`
- ✓ affiche le fichier `/etc/passwd` trié dans l'ordre alphabétique.
- ✓ affiche le fichier `/etc/passwd` en numérotant ses lignes.
- ✓ affiche tous les identifiants des utilisateurs du système.
- ✓ le nombre de lignes du fichier `/etc/passwd`
- ✓ le nombre de lien attaché à un fichier
- ✓ seulement l'heure de création des fichiers ou répertoire

4. L'enchaînement de plusieurs filtres à l'aide des *redirections* ou des *tubes* (« pipes ») vus, permet de construire de nouvelles opérations de filtrage et d'effectuer simplement des traitements évolués sur des fichiers contenant du texte.
Donner la commande qui :

- ✓ afficher écran par écran le contenu du répertoire `/usr/bin`
- ✓ compte le nombre de commandes stockées dans le répertoire `/bin`
- ✓ crée un fichier contenant les 5 premières lignes numérotées du fichier `/etc/passwd`
- ✓ crée un fichier contenant les lignes 5 à 15 numérotées du fichier `/etc/passwd`
- ✓ crée un fichier obtenu en remplaçant les minuscules par des majuscules du fichier `/etc/passwd` (utiliser `tr` derrière un tube)

C'est tout! ☺