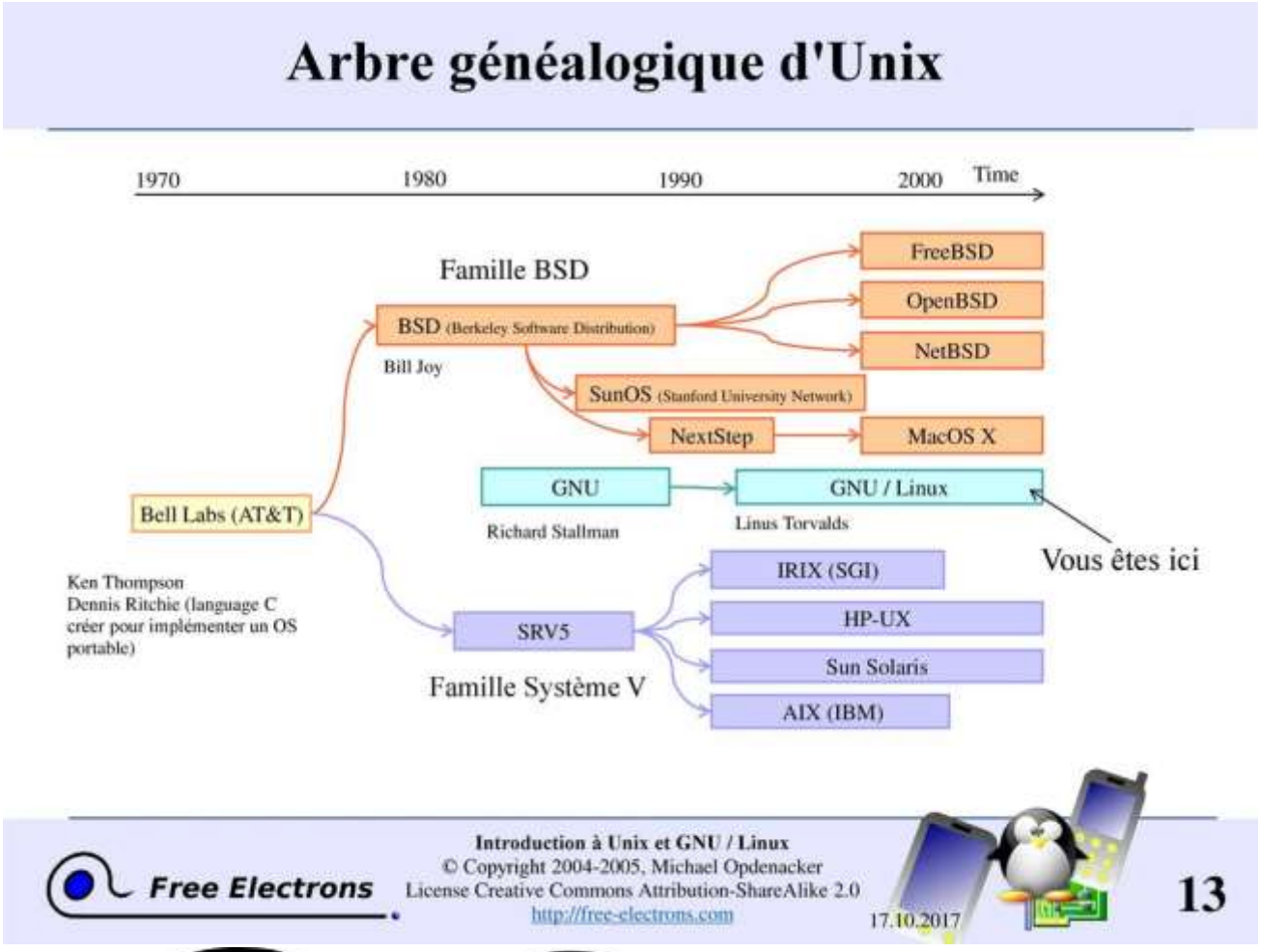
Type de document	Cours et correction	Classe	1re	Durée	2h	Date	02/04/2024
Thème et contenu(s)	Systèmes d'exploitation						
Capacités attendues	Connaître les principales commandes d'un OS libre, et ses principes de fonctionnement						
Prérequis	Avoir fait les 2 TP						
Description	Quelques points de cours et éléments de correction des TP						

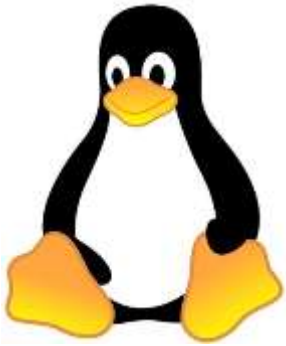
I) Éléments théoriques

Quelques éléments pour répondre aux questions des TP. Les images parlent plus que le texte !



GNU

II) Le shell Linux



II - a) Le prompt et les différents shells

Le premier contact que l'on a avec le shell lorsqu'on ouvre une console est le prompt : il se finit traditionnellement par le caractère `$` (même si d'autres caractères peuvent exister). Dans la plupart des distributions de Linux, c'est `bash` qui est le shell par défaut. Il contient : le nom de l'utilisateur, le nom de la machine, et le répertoire courant. Dans la capture d'écran suivante, `stephane` est l'utilisateur, `ubuntu` la machine, et `home` le répertoire courant.

```
stephane@ubuntu:/home$
```

Il existe plusieurs shells sur le même système, situés dans le fichier `/etc/shells` et facilement affichables grâce à la commande `cat` :

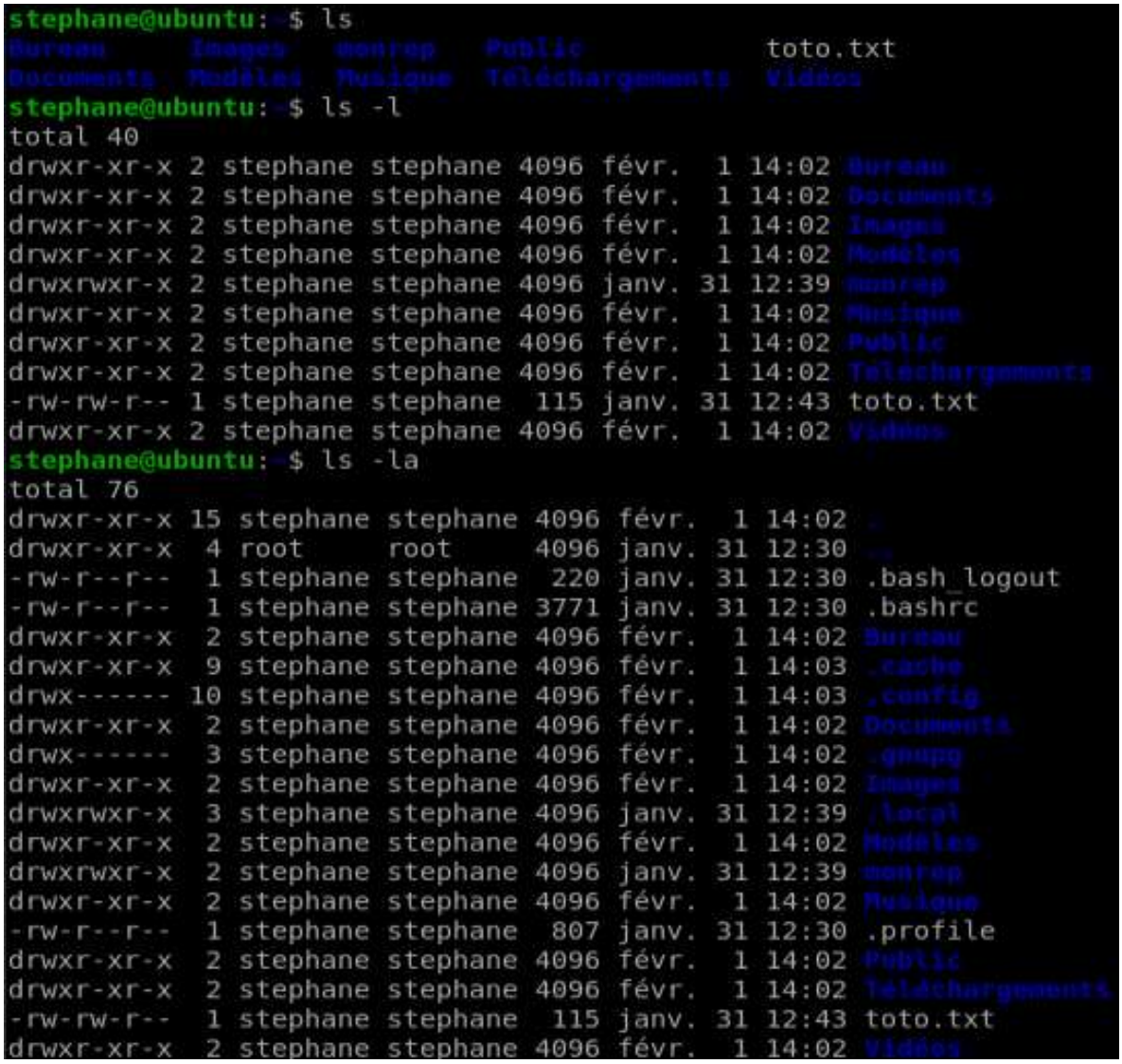
```
stephane@ubuntu:/home$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/bin/dash
/usr/bin/dash
/usr/bin/tmux
/usr/bin/screen
```

II - b) Lister les fichiers du répertoire courant

La commande `ls` est utilisée pour lister les fichiers du répertoire courant, de manière basique. Il est possible de lui ajouter plusieurs options :

- `-l` donne un format d'affichage long
- `-a` affiche les références relatives du répertoire courant : `.` et du répertoire parent : `..` ainsi que les fichiers cachés (souvent, des fichiers de configuration ou de préférences utilisateur) qui commencent aussi par `.`
- On peut aussi lier les commandes comme : `ls -la` ou `ls -al`
- Il existe plein d'autres options pour cette commande…

La commande `ls -al` montre notamment les droits d'accès au fichier (utilisateur/groupe/autres) appelés permissions, le nom du propriétaire, son groupe, la taille du fichier, la date de dernière modification, et enfin le nom du fichier/du répertoire.



II - c) Naviguer dans les répertoires : trouver son chemin

Un système d'exploitation, Linux y compris, considère l'ensemble des **répertoires** (appelés aussi **dossiers**) et des fichiers comme une **arborescence** (avec la racine en haut) qui modélise très bien les relations **hiérarchiques** (contenant/contenu).

L'exemple suivant reproduit partiellement la réalité : On peut voir la racine désignée par `/`, qui contient 3 dossiers (répertoires). Le répertoire `usr` contient `monfic.txt`. Le répertoire `var` contient deux répertoires : `lib` et `cache`. `Cache` contient lui-même le répertoire `apt`, qui contient deux fichiers : `pkgcache.bin` et `srcpkgcache.bin`.

L'utilisateur se situe dans le répertoire `cache`. Dès lors, il possède deux moyens d'accéder à un fichier ou à un dossier. Par exemple, s'il veut aller dans le dossier `apt` :

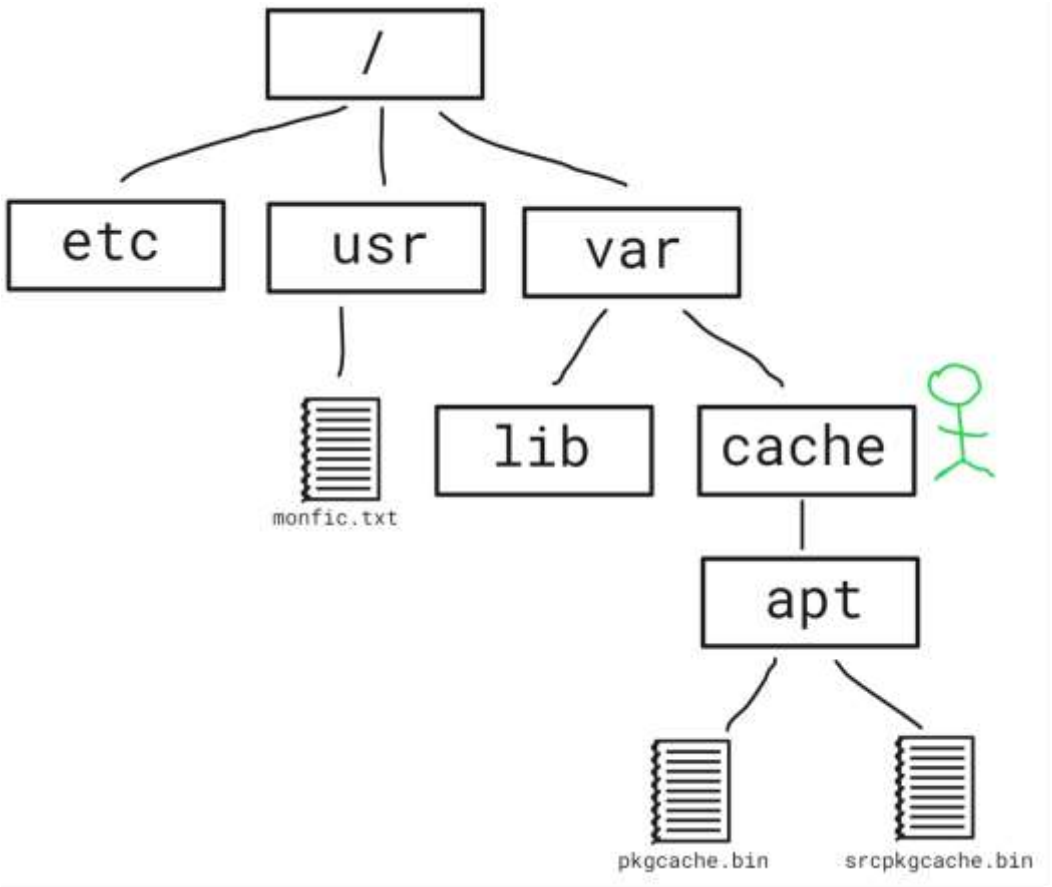
- `cd apt`
- `cd /var/cache/apt`

Dans le premier cas, la commande fonctionne car le shell « voit » les « enfants directs » du répertoire courant. Donc, en étant dans `cache`, il voit `apt`. Comme on donne un chemin depuis le répertoire courant, cela s'appelle un **chemin relatif**. Dans le second cas, on donne le chemin depuis la racine : cela s'appelle un **chemin absolu**.

Autre exemple, l'utilisateur est toujours dans le répertoire `cache` et désire aller dans le répertoire `usr`. Il peut :

- Indiquer le chemin absolu en partant de la racine, en faisant : `cd /usr` (c'est la solution la plus simple)
- Indiquer le chemin relatif en partant du répertoire courant, en remontant d'un parent, puis d'un autre (ce qui le ferait arriver à la racine) puis redescendre sur `usr` (notre destination) : `cd ../../usr`

La seconde possibilité, moins pratique et un peu déroutante lorsqu'on débute, est prisée par les hackers : elle permet de mener des **attaques par traversée de répertoires** (directory traversal).



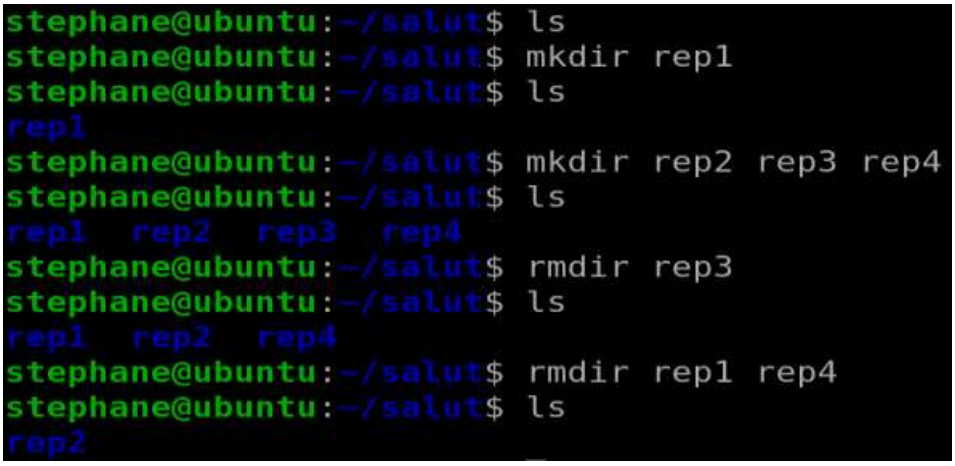
Des raccourcis existent et sont à connaître :

- `cd /` : va à la racine
- `cd ~` : va dans le répertoire courant de l'utilisateur
- `cd -` : revient au répertoire dans lequel on<, se trouvait précédemment
- `cd ..` : va dans le répertoire parent

La commande `pwd` nous indique dans quel répertoire on se trouve (même si la plupart des bash nous le disent dans le prompt) et la commande `cd`, pour change directory, suivie du nom du répertoire/de son chemin, permettent de changer de répertoire.

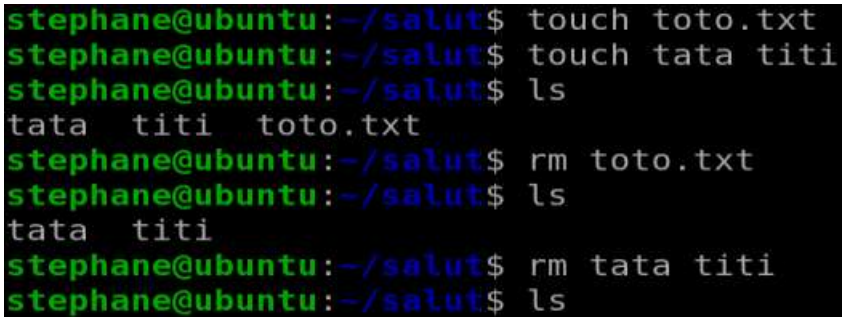
II - d) Créer des répertoires et les supprimer

Cela se fait très simplement avec la commande `mkdir` : elle a besoin qu'on lui fournisse un répertoire (au minimum) mais si on lui en donne plusieurs, elle les crée tous d'un coup, comme le montre la capture suivante. Idem pour `rmdir`, qui permet de supprimer un répertoire, s'il est vide. Sinon, il faut utiliser la commande `rm -rf` suivie du nom du répertoire (commende très dangereuse, pouvant théoriquement supprimer toute votre installation de linux car elle démarre de la racine, si vous tapez : `rm -rf /`)



II - e) Créer un fichier, ou y accéder s'il existe, et le supprimer

On crée un fichier avec la commande `touch` suivie du nom du fichier (ou de plusieurs noms, si on veut les créer d'un coup). Si le fichier existe déjà, `touch` modifie la date de dernière ouverture (la commande se comporte comme si vous aviez accédé au fichier). La commande `rm` fonctionne de la même manière.

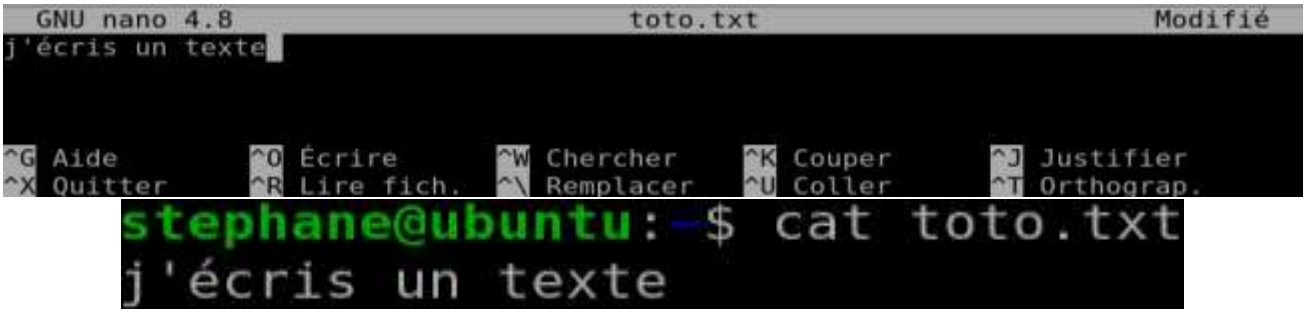


II - f) Déplacer/renommer un fichier/dossier

La commande `mv <source> <destination>` permet de déplacer un fichier (le nom vient de move). Si la destination n'existe pas, le système comprend que vous voulez modifier son nom et renomme alors le fichier/dossier.

II - g) Ecrire dans un fichier et l'afficher

On utilise pour cela un éditeur de texte, par exemple l'éditeur old-school `nano` . Voici un exemple d'utilisation. CTRL+O sert à sauvegarder. CTRL + X sert à quitter. On peut lire le contenu du fichier grâce à la commande `cat` suivie du nom du fichier.




```
stephane@ubuntu:~$ cat toto.txt
j'écris un texte
```

II - h) Créer un utilisateur

Le but d'un OS multi-utilisateurs est, comme son nom l'indique, de gérer plusieurs utilisateurs (connectés ou non simultanément sur la même machine). Pour ce faire, on va utiliser la commande `adduser` suivie du nom de l'utilisateur que l'on désire créer. Le système va nous demander de taper un mot de passe, puis de le confirmer.

Le mot de passe n'est, par sécurité, jamais affiché à l'écran. Rien n'apparaît lorsque vous tapez (des * permettraient à quelqu'un qui lit par-dessus votre épaule de connaître le nombre de caractères). Pour changer d'utilisateur courant, on va utiliser la commande `su`. Pour créer un utilisateur, comme pour faire tout un tas de choses, on a parfois besoin des privilèges du super-administrateur, le **root**, l'équivalent d'un dieu (en informatique). Dans le cas contraire, le système nous l'interdit. On va pour cela utiliser la commande `sudo` avant la commande que l'on veut exécuter. L'exemple qui suit vous montre comment faire cela. Notez au passage les deux shells différents que j'utilise : un minimaliste en une seule ligne, l'autre sur deux lignes (il s'agit du shell `zsh`).

```
kali@kali:~$ adduser toto
adduser: Only root may add a user or group to the system.

kali@kali:~$ sudo adduser toto
[sudo] password for kali:
Adding user `toto' ...
Adding new group `toto' (1003) ...
Adding new user `toto' (1003) with group `toto (1003)' ...
adduser: The home directory `/home/toto' already exists.  Not touching this directory.
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for toto
Enter the new value, or press ENTER for the default
        Full Name []: Toto la Blague
        Room Number []: 15
        Work Phone []: 0620202020
        Home Phone []: 0495202020
        Other []: Rien !
Is the information correct? [Y/n] Y
Adding new user `toto' to supplemental / extra groups `users' ...
Adding user `toto' to group `users' ...

kali@kali:~$ su toto
Password:
┌──(toto㉿kali)-[/home/kali]
└─$
```

II - i) Afficher les processus

Un OS multitâche permet de donner l'illusion que l'on peut faire « plusieurs choses en même temps ». Ces « choses » sont matérialisées par des **processus** : un **processus** est une tâche effectuée par la machine, soit durant un temps donné, soit indéfiniment, ça peut être par exemple un programme lancé par l'utilisateur, des processus venant de l'OS…etc. Chaque processus est identifié de manière unique au moyen d'un nombre entier, son Process ID, qu'on appelle `pid`. Pour afficher la liste des processus en cours, on utilise la commande `ps` comme suit :

```
stephane@ubuntu:~$ ps
    PID TTY          TIME CMD
   2310 pts/0    00:00:00 bash
   5907 pts/0    00:00:00 bash
   6400 pts/0    00:00:00 bash
   6464 pts/0    00:00:00 ps
```

II - j) Aller plus loin

Toutes les commandes que nous avons vues comportent une foule d'options. Pour en savoir plus sur une commande, plusieurs commandes vont vous aider :

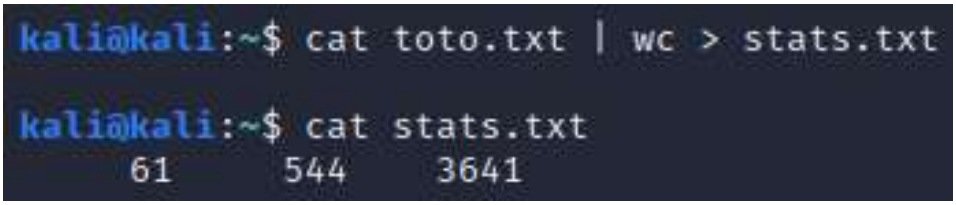
- La commande `help` est spécifique au bash, c'est une commande interne à ce dernier qui nous renseigne sur ses mots-clefs et ses commandes
- La commande `info` est un genre de précurseur du Web : elle existe depuis le projet GNU, vous pouvez la voir comme un livre numérique avec une table des matières cliquable
- La commande `man` est la plus populaire : elle correspond à un manuel utilisateur et concerne, outre les commandes basiques du système, tous les programmes de la machine

```
stephane@ubuntu: $ help cd
cd: cd [-L|[-P [-e]] [-@]] [rép]
    Change le répertoire de travail du shell.

    Change le répertoire actuel vers DIR.  Le répertoire DIR par défaut
    est donné par la variable « HOME » du shell.

    La variable CDPATH définit le chemin de recherche du répertoire contenant
    DIR. Les noms de répertoires alternatifs dans CDPATH sont séparés par un deux-point « : ».
    Un nom de répertoire vide est identique au répertoire actuel.  Si DIR commence
    avec une barre oblique « / », alors CDPATH n'est pas utilisé.

    Si le répertoire n'est pas trouvé et que l'option « cdable_vars » du shell est définie,
    alors le mot est supposé être un nom de variable. Si la variable possède une valeur,
    alors cette valeur est utilisée pour DIR.

    Options :
      -L        force le suivi des liens symboliques : résout les liens symboliques dans
                DIR après le traitement des instances de « .. »
      -P        utilise la structure physique des répertoires sans suivre les liens
                symboliques : résout les liens symboliques dans DIR avant le traitement des
                instances de « .. »
      -e        si l'option -P est fournie et que le répertoire de travail actuel ne peut pas
                être déterminé avec succès, alors sort avec un code de retour non nul
      -@  sur les systèmes qui le supporte, présente un fichier avec des attributs
                étendus comme un répertoire contenant les attributs du fichier

    Le comportement par défaut est de suivre les liens symboliques, comme si « -L » était précisé.
    « .. » est traité en retirant le composant immédiatement avant dans le chemin jusqu'à
    la barre oblique ou le début de DIR.

    Code de sortie :
    Renvoie 0 si le répertoire est changé et si $PWD est correctement défini
    quand -P est utilisé ; sinon autre chose que 0.
```

```
Next: dir invocation,  Up: Directory listing

10.1 'ls': List directory contents
==================================

The 'ls' program lists information about files (of any type, including
directories).  Options and file arguments can be intermixed arbitrarily,
as usual.

   For non-option command-line arguments that are directories, by
default 'ls' lists the contents of directories, not recursively, and
omitting files with names beginning with '.'.  For other non-option
arguments, by default 'ls' lists just the file name.  If no non-option
argument is specified, 'ls' operates on the current directory, acting as
if it had been invoked with a single argument of '.'.

   By default, the output is sorted alphabetically, according to the
locale settings in effect.(1)  If standard output is a terminal, the
output is in columns (sorted vertically) and control characters are
output as question marks; otherwise, the output is listed one per line
and control characters are output as-is.

   Because 'ls' is such a fundamental program, it has accumulated many
options over the years.  They are described in the subsections below;
within each section, options are listed alphabetically (ignoring case).
The division of options into the subsections is not absolute, since some
options affect more than one aspect of 'ls''s operation.

   Exit status:

     0 success
     1 minor problems  (e.g., failure to access a file or directory not
       specified as a command line argument.  This happens when listing a
       directory in which entries are actively being removed or renamed.)
     2 serious trouble (e.g., memory exhausted, invalid option, failure
       to access a file or directory specified as a command line argument
       or a directory loop)

   Also see *note Common options::.
```

```
LS(1)                                                        User Commands
                                                                   LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List  information  about  the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is
       specified.

       Mandatory arguments to long options are mandatory for short options too.

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..

       --author
              with -l, print the author of each file

       -b, --escape
              print C-style escapes for nongraphic characters

       --block-size=SIZE
              with -l, scale sizes by SIZE when printing them; e.g., '--block-si
ze=M'; see SIZE format below
```

La puissance du shell réside aussi dans le fait que c'est un langage de programmation à part entière : on peut donc créer des scripts, chaîner des commandes, c'est sans fin…et très rapide, une fois que l'on a apprivoisé la bête !
Un exemple pour le fun (bien entendu pas à connaître) : ici je pars d'un fichier texte que je possède (`toto.txt`), j'affiche son contenu mais je redirige la sortie du `cat` vers l'entrée de la commande `wc` (Word Count, pour compter les mots) dont j'envoie la sortie dans le fichier `stats.txt` (qui est créé pour l'occasion). Il contient donc mes stats :



Imaginez un seul instant le temps que vous mettriez à faire cela avec une interface graphique !

III) Conclusion

Les éléments donnés ici ne constituent pas un cours complet : le but des deux TP était double : vous permettre de mieux comprendre le fonctionnement d'un OS et d'interagir avec lui, mais aussi de vous apprendre à rechercher une information fiable sur un moteur de recherche. Il faut donc continuer à vous entraîner en utilisant par exemple un shell en ligne comme celui de Fabrice Bellard ou mieux : installer une machine virtuelle linux sur votre ordinateur personnel.
Cela vous permettra de comprendre le fonctionnement d'une **machine virtuelle** et cela reste sans danger pour votre ordinateur. De plus, vous pourrez ainsi tester plusieurs distributions de Linux.
Voici quelques liens pour vous apprendre à faire cette manipulation très intéressante et très utile :

https://lecafedugeek.fr/creez-votre-premiere-machine-virtuelle/
https://www.malekal.com/comment-debuter-et-utiliser-virtualbox/
https://www.youtube.com/watch?v=miYBAV7ee1w
https://www.numetopia.fr/tutoriels/tuto-applications/tuto-virtualbox/