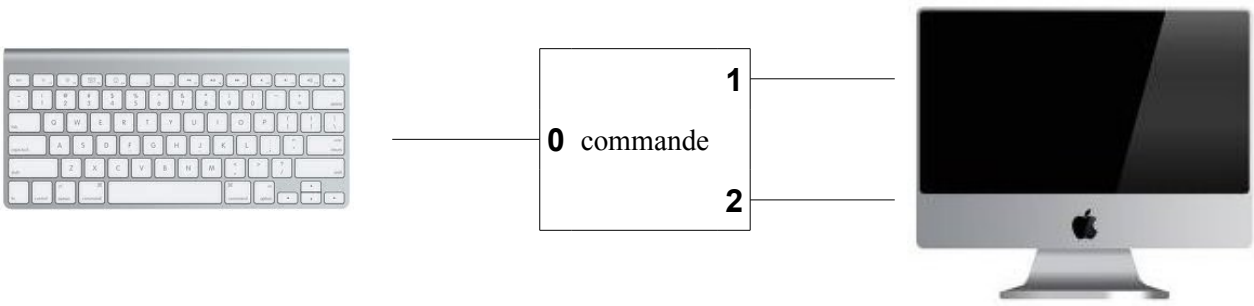
Partie II – Programmation shell sous Linux

1. Redirections des entrées/sorties standard

Les entrées/sorties standard sont représentées par trois fichiers standards qui sont associés à chaque commande du shell Linux :

flux	nom	descripteur	matériel
Entrée standard	`stdin`	0	Associée au clavier
Sortie standard	`stdout`	1	Associée à l'écran pour afficher les résultats d'une commande
Sortie erreur standard	`stderr`	2	Associée à l'écran pour afficher les messages d'erreur d'une commande



Les redirections consistent à redéfinir ces trois fichiers. Elle permettent de détourner les entrées et les sorties d'une commande soit vers ou à partir d'autres fichiers que les trois précédents, soit vers d'autres commandes.

1.1 Redirection de la sortie standard

Simple : `commande > fichier`

Les résultats de la commande `commande` sont affichés dans le fichier `fichier` au lieu d'être affichés à l'écran (dans le terminal). Si `fichier` n'existe pas, il est créé. Sinon, son contenu est écrasé, c'est-à-dire qu'il est effacé et remplacé par le résultat de la commande `commande`.

La commande `> fichier` crée un fichier vide. Le symbole > est équivalent à `1>`.

Double : `commande >> fichier` :

Les résultats de la commande `commande` sont affichés à la fin du fichier `fichier` au lieu d'être affichés à l'écran. Si `fichier` n'existe pas, il est créé. Sinon, son contenu est complété par le résultat de la commande `commande`. Le symbole >> est équivalent à `1>>`.

Exemple :

```
$ echo 'voici le contenu de mon répertoire personnel : '
voici le contenu de mon répertoire personnel
$ ls ~
/Bureau  /Documents  /Images  /Musique  /Téléchargements  /Vidéos  /tmp  /tmp1
$ echo 'voici le contenu de mon répertoire personnel : ' > toto.txt
$ ls ~ ~/Desktop >> toto.txt
```

La première commande crée le fichier `toto.txt` s'il n'existe pas (>) et affiche dedans le texte « voici le contenu de ce répertoire ». La deuxième commande ajoute à la suite de ce texte (>>) le résultat de la commande `ls -l`.

1.2 Redirection de la sortie erreur standard

Simple : `commande 2> fichier`

Les messages d'erreur produits par la commande `commande` sont affichés dans le fichier `fichier` au lieu d'être affichés à l'écran (dans le terminal). Si `fichier` n'existe pas, il est créé. Sinon, son contenu est écrasé, c'est-à-dire qu'il est effacé et remplacé par le résultat de la commande `commande`.

Double : `commande 2>> fichier`

Les messages d'erreur produits par la commande `commande` sont affichés à la fin du fichier `fichier` au lieu d'être affichés à l'écran (dans le terminal). Si `fichier` n'existe pas, il est créé. Sinon, son contenu est complété par le résultat de la commande `commande`.

Exemple :

```
$ rm tmp tmp1
rm: impossible de supprimer «tmp/»: est un dossier
rm: impossible de supprimer «tmp1/»: est un dossier
$ rm tmp 2> err.txt
$ rm tmp1 2>> err.txt
```

Avec la première commande, le message d'erreur `rm: impossible de supprimer «tmp/»: est un dossier` est enregistré dans le fichier `err.txt`. Avec la deuxième commande, le message d'erreur `rm: impossible de supprimer «tmp1/»: est un dossier` est ajouté à la fin fichier `err.txt`.

1.3 Redirection combinée

Simple : `commande &> fichier` et `commande &>> fichier`

À la fois la sortie standard et la sortie erreur standard sont redirigées vers le fichier `fichier`.

1.4 Redirection de l'entrée standard

Simple : `ommande < fichier`

Les arguments de la commande `ommande` sont récupérées dans le fichier `fichier` au lieu d'être entrées au clavier par l'utilisateur. Le symbole < est équivalent à `0<`.

Exemple :

Le fichier `toto.txt` contient le texte «`bonjour le monde !`».

```
$ cat < toto.txt
bonjour le monde !
$ cat 0< toto.txt
bonjour le monde !
```

Double : `commande << MOT`

Les arguments sont lus sur l'entrées standard jusqu'à ce que le mot `MOT` soit rencontré, puis le résultat de la commande `ommande` est affiché.

Exemple :

```
$ cat << STOP
> Bonjour le monde !
> STOP
Bonjour le monde !
$ cat > toto.txt << STOP
> Bonjour le monde !
> STOP
```

Lorsque l'utilisateur a saisi la commande, l'interpréteur de commande a affiché le caractère < pour lui indiquer qu'il attend une saisie texte. Dès que l'utilisateur a saisit le mot `STOP`, l'interpréteur de commandes a repris la main. Le texte `Bonjour le monde` a alors été affiché à l'écran dans le premier exemple et dans le fichier `toto.txt` dans le deuxième exemple.

1.5 Les tubes de communication

Les tubes : `commande1 | commande2 | ...`

La sortie d'une commande `commande1` est utilisée comme entrée d'une commande `commande2`., puis la sortie de `commande2` est utilisée comme entrée de `commande3`, etc. Les sorties intermédiaires ne sont pas affichées. Seule la sortie de la dernière commande est affichée (ou enregistrée dans un fichier si on a utilisé une redirection pour la sauvegarder).

Exemple :

```
$ sort toto.txt | uniq > tutu.txt
```

Les lignes du fichier `toto.txt` sont triées (par ordre alphabétique et/ou numérique). Le résultat est fourni en entrée de la commande `uniq` qui supprime les doublons (les lignes consécutives dont le contenu est le même). La sortie de cette dernière commande est sauvegardée dans le fichier `tutu.txt`. Ce tube de commandes est l'équivalent de la suite de commandes suivantes :

```
$ sort toto.txt > titi.txt
$ uniq titi.txt > tutu.txt
```

Exercice 1

1. Cherchez tous les fichiers de nom `passwd` depuis le répertoire racine `/`. Sauvegardez la sortie standard de votre commande dans un fichier `lst_passwds.log` et sa sortie erreur standard dans un fichier `lst_passwds.err`.
2. Affichez le contenu de votre répertoire personnel et de ses sous-répertoires et sauvegardez le résultat dans un fichier `sauvegarde.txt`.
3. Utilisez la commande `wc` pour afficher le nombre de fichiers et répertoires dans votre compte utilisateur.
4. Triez le contenu du fichier `sauvegarde.txt` par ordre alphabétique et supprimez les doublons de lignes en utilisant les commandes `sort` et `uniq` (en une seule commande). Affichez à nouveau le nombre de fichiers et répertoires dans votre compte utilisateur. Votre compte personnel contient-il des fichiers et répertoires dupliqués ?
5. Utilisez les commandes `ls` et `wc` dans une seule commande pour afficher le nombre d'entrées dans votre répertoire courant.
6. Utilisez les commandes `ls` et `egrep` et `wc` pour afficher le nombre de fichiers de nom `passwd` depuis la racine `/`.
7. Utilisez les commandes `find` et `cat` pour concaténer tous les fichiers de nom `passwd` dans un fichier `passwd_all`.
8. À quoi sert la commande `tee` ?
9. Que fait la commande suivante ? Proposez une commande différente qui produit le même résultat.

```
$ > fic
```

10. Après exécution de la commande suivante, que contient le fichier `~/mon_rep` ?

```
$ ls -l | tee ~/mon_rep | wc -l
```

11. Que fait la commande suivante ?

```
$ ps ax > ps.txt 2>&1
```

2. Les caractères spéciaux du shell Linux

2.1 Les caractères génériques

Nombreuses sont les commandes Linux qui prennent en argument un nom de fichier (`ls`, `cd`,...). Ces noms de fichiers peuvent être fournis soit littéralement, soit à l'aide d'expressions régulirères. Le shell Linux propose en effet des caractères spéciaux dédiés à l'écriture de noms de fichiers sous la forme de motifs. Ces motifs sont ensuite utilisés pour effectuer des recherches sur les entrées d'un répertoire. Ces caractères spéciaux sont appelés **caractères génériques**.

Un motif peut contenir plusieurs caractères génériques (plusieurs fois le même et/ou plusieurs différents).

Si le shell ne trouve aucune correspondance avec le motif recherché, ce dernier est transmis tel quel à la commande :

```
$ ls sz*
ls: ne peut accéder sz*: Aucun fichier ou répertoire de ce type
```

Expressions simples

***** : (astérisque)

- Substitué par une suite de caractères quelconque qui peut être de longueur nulle.
- Remplace n'importe quel caractère dans le motif sauf le `'.'` (point) en première position dans un nom de fichier afin d'éviter notamment la suppression de fichiers cachés avec `rm *`.

```
$ ls a*
(fichiers dont le nom commence par 'a')
$ ls r*e
(fichiers dont le nom commence par un 'r' et se termine par 'e')
$ ls *s
(fichiers dont le nom se termine par un 's', comme par exemple 'textes', 'cours', 's',
'sss', mais pas '.os')
$ ls *ex*
(fichiers dont le nom contient la chaîne « ex » précédée et suivie de n'importe quelle
chaîne de caractères, comme par exemple 'exemples', 'textes', 'ex')
```

? : (point d'interrogation)

- Substitué par un unique caractère, et obligatoirement un seul
- Remplace n'importe quel caractère dans le motif sauf le `'.'` (point) en première position dans un nom de fichier afin d'éviter notamment la suppression de fichiers cachés avec `rm *`.

```
$ ls ro?e
(fichiers dont le nom contient exactement 4 caractères, commence par « ro » suivi d'un
caractère quelconque et se termine par 'e', comme par exemple 'rose' ou 'robe')
$ ls ???
(fichiers dont le nom contient exactement 3 caractères, comme par exemple 'aac', 'haa')
$ ls ?a*
(fichiers dont le nom contient un 'a' en 2e position, précédé d'un caractère quelconque
et suivi de n'importe quelle chaîne de caractères, par exemple 'mathématiques', 'ka')
$ ls ??a*b?
(fichiers dont le nom contient au moins 5 caractères dont un 'a' en 3e position et un
'b' en avant-dernière position)
$ ls *.?
(fichiers dont le nom a une extension d'un seul caractère)
```

[liste_de_caractères] : (crochets)

- Représentent un et un seul caractère parmi ceux de la liste.
- Pas d'espace entre les caractères, sauf si l'espace lui-même fait partie de la liste des caractères recherchés
- On peut simplifier une liste de caractères s'ils sont consécutifs dans l'ordre de la table ASCII en séparant le premier et le dernier par un signe `'-'` (moins).
- Pour spécifier le caractère `'-'` parmi les caractères recherchés, on le place en début ou en fin de liste.

```
$ ls [abc]*
(fichiers dont le nom commence par un 'a', un 'b' ou un 'c')
$ ls *[aeiouy]*
(fichiers dont le nom contient une voyelle, précédée et suivie d'une chaîne quelconque)
$ ls ?[aeiouy-]*
(fichiers dont le nom contient une voyelle ou un '-' en 2e position)
$ ls [a-zA-Z]*
(fichiers dont le nom commence par un caractère alphabétique, minuscule ou majuscule)
```

- **!** : On peut aussi représenter la négation d'une liste de caractères entre crochets en plaçant le caractère `!` (point d'exclamation) en première position à l'intérieur de ceux-ci :

```
$ ls [!a-z]*
(fichiers dont le nom ne commence pas par une minuscule)
$ ls [!0-9]
(fichiers dont le nom ne contient pas de caractères numériques)
```

Une commande peut prendre en argument plusieurs expressions séparées par un espace.

```
$ rm -Rf *.c *.o
(supprimer tous les fichiers dont le nom se termine par « .c » ou « .o »)
```

Expressions complexes

Pour utiliser des expressions complexes dans le shell bash, il faut préalablement activer l'option `extglob` avec la commande suivante :

```
$ shopt -s extglob
```

?(expression) : l'expression est présente **0 ou 1** fois

***(expression)** : l'expression est présente **0 à n** fois

@(expression) : l'expression est présente **exactement 1** fois

+(expression) : l'expression est présente **1 à n** fois

!(expression) : l'expression n'est pas présente

```
$ ls foo?(415).tmp
(fichiers dont le nom commence par « fich » suivi de 0 ou 1 occurrence de « 866 » suivi
de « .tmp »)
$ ls foo*(415).tmp
(idem avec 0 à 1 occurrence(s) de « 415 »)
```

[?*@+!](expression|expression|...) : le symbole **|** prend le sens de "OU" dans une expression complexe

```
$ ls foo?(415|416).tmp
(fichiers dont le nom commence par « fich » suivi de 0 ou 1 occurrence de « 866 » ou
« 867 » suivi de « .tmp »)
```

2.2 Les caractères de citation

Les caractères de citation (ou caractères de protection) servent à déspécialiser les caractères spéciaux du shell.

' ' : (guillemets simples, apostrophes sur le clavier)

- Tous les caractères spéciaux entre apostrophes sont ignorés par le shell
- Les guillemets simples fonctionnent pas paire sur la ligne de commande
- Les guillemets simples ne se protègent pas eux-mêmes

« » : (guillemets doubles)

- Tous les caractères spéciaux entre guillemets sont ignorés par le shell, sauf $, `` et \.

**** : (antislash)

- Le caractère suivant l'antislash perd sa signification de caractère spécial.

```
$ ls '*'
(fichiers dont le nom est '*')
$ ls *'*''
$ echo '$PATH'
$PATH
$ echo >
bash: syntax error near unexpected token '>'
('>' est ici le caractère de redirection de la sortie standard)
$ echo '>'
>
$ echo 'le monde n'est pas une marchandise'
>
(le shell attend la suite)
$ echo 'le monde n'est pas ce que l'on croit'
le monde nest pas ce que lon croit
$ ls *\**
  (fichiers dont le nom contient au moins une fois '*', comme par exemple 'd*g.txt',
'textes**anglais.txt')
```

2.3 Les caractères de substitution de commandes

Le shell Linux propose des caractères dits de substitution qui permettent d'insérer dans une ligne de commande le résultat d'une autre commande. Deux syntaxe équivalentes peuvent être utilisées.

`commande` : (apostrophes inversées)

$(commande)

```
$ echo vous êtes logname
(la commande logname permet d'afficher le login de l'utilisateur connecté)
$ echo vous êtes `logname`
vous êtes jdupont
$ echo vous êtes $(logname)
```

```
vous êtes jdupont
$ mon_login=`logname`
$ mon_login=$(logname)
$ echo $mon_login
jdupont
(le résultat de la commande logname est placé dans la variable 'mon_login')
```

Exercice 2

1. Quelle commande permet d'afficher la liste des entrées du répertoire courant.
2. À partir d'un répertoire différent de `/usr/include`, affichez la liste des entrées du répertoire `/usr/include`.
3. À partir d'un répertoire différent de `/usr/include`, affichez la liste des entrées du répertoire `/usr/include` dont le nom commence par `'c'`.
4. À partir d'un répertoire différent de `/usr/include`, affichez la liste des entrées du répertoire `/usr/include` cont le nom contient un `'i'`.
5. À partir d'un répertoire différent de `/usr/include`, affichez la liste des entrées du répertoire `/usr/include` dont le nom peut contenir un `'i'` mais pas en première position et qui se terminent par `'.h'`.
6. Affichez la liste des entrées du répertoire `/usr/include` dont le nom contient la chaîne «`std`».
7. Affichez la liste des entrées du répertoire `/usr/include` dont le nom contient exactement 3 caractères.
8. Que font les commandes suivantes ?

   a. `ls foo+(415|416).tmp`

   b. `ls foo@(+(415)|+(416)).tmp`

   Pour les tester, lancez la commande suivante :

```
touch foo415415.tmp foo415416415.tmp foo415.tmp foo416.tmp foo416416.tmp
```

9. Utilisez la commande `echo` pour afficher des lignes de texte contenant les caractères suivants : `& ~ # () | \ ^ @ $ * !`.
10. Quelle est la différence entre les 4 commandes suivantes ?

    a. `echo $HOME` **b.** `echo '$HOME'` **c.** `echo ''$HOME''` **d.** `echo \$HOME`

11. Afficher le texte suivant en utilisant les commandes `whoami` et `date` :

    `Bonjour, je suis <votre login>, nous sommes le <la date du jour>`

    L'heure devra par exemple être afficheé ainsi : `8 heures, 31 minutes et 23 secondes`

12. Expliquez le fonctionnement de la commande suivante :

```
$ diff <(ps axo comm) <(ssh user@host ps axo comm)
```

13. Dans la commande suivante, quelle est l'entrée de la commande `grep` ? Que signifie l'opérateur `2>&1` ?

```
$ ps ax 2>&1 | grep init
```

3. Commandes filtres

Exercice 3

1. Parcourez le manuel des commandes suivantes puis donnez 5 exemples d'utilisation pour chacune en précisant le résultat correspondant à chacun : `grep`, `head`, `tail`, `tee`, `nl`, `cut`, `paste`, `tr`.

2. Affichez avec `ps` et `grep` dans un tube de commande tous les processus en cours (avec la commande `ps`) dont l'entrée contient la chaîne `'pts'`.

3. Affichez avec `ls` et `grep` dans un tube de commande la liste des entrées de votre répertoire personnel qui sont des répertoires ou des fichiers ordinaires.

4. Affichez avec `ls` et `grep` dans un tube de commande la liste des entrées de votre répertoire dont le nom commence par `'.'`.

5. Affichez avec `ps` et `wc` dans un tube de commande le nombre de processus en cours.

6. Affichez avec `ps` et `grep` dans un tube de commande la liste des processus qui sont lancés depuis une interface en ligne de commande (`tty`) ou depuis une fenêtre terminal (`pts`).

   Affichez le résultat de votre commande à la fois sur la sortie standard et dans un fichier.

7. Le fichier `/etc/passwd` contient la liste des utilisateurs qui ont un compte sur la système. Chaque ligne contient les informations concernant un utilisateur. Chaque ligne est composée de 7 champs séparés par le caractère `':'` :

```
login:mdp:UID:GID:description:shell
```

`login`	: le login de l'utilisateur
`mdp`	: son mot de passe, représenté par un `'x'` si le mot de passe est dans `/etc/shadow` (le fichier qui contient les mots de passe cryptés des utilisateurs) ou un `'!'` si le compte est verrouillé
`UID`	: l'identifiant unique de l'utilisateur
`GID`	: l'identifiant unique de son groupe principal
`description`	: un commentaire
`shell`	: le shell par défaut de l'utilisateur (par exemple `/bin/bash`) ou toute autre commande (par exemple `/usr/sbin/nologin`, pour interdire la connexion)

   a. Affichez avec `head` et `tail` dans un tube de commande les lignes 3 à 10 du fichier `/etc/passwd`.

   b. Affichez avec `head`, `tail` et `nl` dans un tube de commande les lignes 3 à 10 du fichier `/etc/passwd` en numérotant les lignes du résultat.

   c. Quelle commande permet de créer une variable `LOGIN` et de lui donner pour valeur le login d'un utilisateur dont l'UID est `UID`.

   d. Triez les lignes du fichier `/etc/passwd` sur le champ `UID` (tri numérique du 3e champ de chaque ligne)

8. À quoi sert la commande `split` ?

9. **a.** Utilisez la commande `wc` pour afficher le nombre de lignes dans les fichiers `/etc/passwd` et

`/etc/shadow`.

b. Placez-vous dans votre répertoire personnel. Créez un répertoire `~/temp`.

c. Copiez le fichier `/etc/passwd` dans le répertoire `~/temp`.

Copiez ensuite le fichier `/etc/shadow` dans le répertoire `~/temp`.

d. Utilisez la commande `cut` avec les options `-d` et `-f` pour extraire le premier champ de chaque ligne du fichier `~/temp/passwd`. Enregistrez le résultat dans un fichier `~/temp/passwd.users`. Faites de même avec le fichier `~/temp/shadow` et enregistrez le résultat dans un fichier `~/temp/shadow.users`.

e. Vérifiez que ces deux derniers fichiers créés (`~/temp/passwd.users` et `~/temp/shadow.users`) contiennent bien le même nombre de lignes (en une seule commande).

f. Triez les fichiers `/etc/passwd.users` et `/etc/shadow.users` par ordre alphabétique et supprimez-en les éventuels doublons. Enregistrez le résultat dans des fichiers `/etc/passwd.users.s` et `/etc/shadow.users.s`, respectivement.

g. Utilisez la commande `diff` pour vérifier que les fichiers `/etc/passwd.users.s` et `/etc/shadow.users.s` contiennent la même liste de noms d'utilisateurs.

h. Utilisez la commande `sort` pour trier le contenu du fichier `~/temp/passwd` par ordre numérique croissant des UID (c'est-à-dire le 3e champ de chaque ligne). Enregistrez le résultat dans un fichier `~/temp/passwd.sUID`. Attention : le séparateur de champs par défaut de `sort` est la tabulation mais il peut être changé à l'aide d'une option.

i. Utilisez les commandes `head` , `tail` et `sort` pour extraire les lignes 10 à 19 du fichier `/etc/passwd`, les trier par ordre alphabétique décroissant et afficher le résultat dans un fichier `sub.txt`.

j. Utilisez la commande `tr` pour crypter le contenu du fichier `~/temp/passwd` de la manière suivante :

- les voyelles `aeiouy` sont remplacées par les chiffres `12345`, respectivement (a=1, e=2, i=3,...)
- le caractère de fin de ligne `\n` est remplacé par `!` (point d'exclamation)
- chaque caractère espace est remplacé par un `_` (underscore).
- tous les 0 (zéro) sont remplacés par un **.** (point)

4. Processus et signaux

4.1 Processus

De même que les fichiers sont identifiés par un inode unique, les processus sont identifiés par un numéro unique : son **PID** (*Processus Identifier*). Les processus sont gérés à l'aide d'une table des processus.

Tout processus a obligatoirement un père, à l'exception du processus **`init`**, le premier processus du système et l'ancêtre de tous les processus. Le PID de `init` est 1.

`ps` : (*process status*) affiche la liste des processus en cours sur le système. Cette commande utilise **`/proc`**, qui contient toutes les informations l'état actuel du système. Ses options les plus utilisées sont :

`-a` : (*all*) affiche les processus de tous les utilisateurs

`-x` : affiche aussi les processus qui ne sont pas rattachés au terminal en cours

`-u` : (*user*) affiche le nom de l'utilisateur qui a lancé le processus et sa date de lancement. Cette option peut prendre en argument une liste d'utilisateurs séparés par une virgule

`-g` : (*group*) même chose pour les groupes

`-t` : même chose pour les terminaux

`-p` : même chose pour les PID

`-f` : (*full-format*) affiche des informations techniques sur chaque processus

`-l` : (*long format*) affiche encore plus d'informatins détaillées sur chaque processus

`-H` : (*hierarchy*) affiche les processus sous la forme d'une hiérarchie (parfois représentée par des tabulations)

```
$ ps aux
$ ps -u root
(affiche les processus lancés par root)
$ ps -H
  PID TTY          TIME CMD
19163 pts/1    00:00:01 bash
22288 pts/1    00:00:00   ps
(le processus 'ps' est fils du processus 'bash')
```

`pstree` : affiche l'arbre des processus en cours sur le système (plus intéressante que l'option `-H` de `ps`)

`top` : affiche l'activité du processeur en temps réel en l'actualisant régulièrement de manière automatique. Pour terminer cette commande, on clique sur **`q`**. son affichage par défaut est défini dans le fichier **`/toprc`** (global à tous les utilisateurs) et **`~/.toprc`** (pour l'utilisateur connecté).

4.2 Gestion des processus

Les commandes du shell peuvent être lancées de deux manières :

- en avant-plan : Le shell lance l'exécution de la commande et s'endort. Il reprend la main lorsque son exécution est terminée. On n'a donc pas la main pendant l'exécution de la commande, ce qui signifie qu'on doit attendre que l'exécution soit terminée pour lancer une autre commande.
- En arrière-plan : Le shell lance l'exécution de la commande et réaffiche son invite de commande. On peut donc lancer une autre commande pendant que la précédente s'exécute. Pour lancer une commande en arrière-plan dans le shell, on ajoute le caractère **`&`** à la fin de la ligne de commande.

```
$ find / -name passwd 1> tmp.txt 2> err.txt &
[3] 22039
$
(le shell lance la commande find, affiche le numéro de son processus puis redonne la
main à l'utilisateur)
```

`wait` : (*attendre*) cette commande permet d'attendre la fin de l'exécution d'un processus lancé en arrière-plan. Elle prend en argument le PID du processus en question.

```
$ wait
(pour attendre la fin de tous les processus lancés en arrière-plan)
$ wait 22039
(pour attendre la fin du processus numéro 22039 lancé en arrière-plan dans le shell
courant)
```

```
$ find / -name passwd 1> tmp.txt 2> err.txt &
(le shell lance la commande find
[3] 22039
et affiche le numéro de son processus
$ wait 22039
il redonne ensuite la main à l'utilisateur qui entre cette dernière commande 'wait'
find: [3]+  Termine 1               find / -name passwd > tmp.txt 2> err.txt
$
quant le processus est terminé, le shell se réveille et affiche l'invite de commandes)
```

4.3 Signaux

Arrêt de processus

`kill -<signal> <PID>` : (*tuer*) permet d'envoyer des signaux aux processus. L'option **`-l`** de cette commande permet d'afficher la liste des signaux disponibles. Le signal envoyé par défaut à cette commande est `SIGTERM (15)`, qui termine le processus.

```
$ kill -15 14818
$ kill 14818
(ces deux lignes sont équivalentes)
```

La section 7 du manuel (**`man 7 signal`**) et le fichier **`/usr/include/asm/signal.h`** contiennent des informations détaillées sur l'utilisation de la commande `kill` et sur la signification des différents signaux.

Un processus peut être interrompt de trois manières dans le shell :

`Ctrl+c` : envoie le signal 2 (`SIGINT`) au processus qui est en cours d'exécution

`Ctrl+\` : envoie le signal 3 (`SIGQUIT`) au processus qui est en cours d'exécution

`kill` : utiliser cette commande avec le PID du processus en cours d'exécution

Suspension de processus

Après avoir lancé un processus en arrière-plan, on peut suspendre son exécution pour le relancer en arrière-plan. On utilise pour cela les commandes suivantes.

`jobs` : affiche la liste des processus en cours d'exécution dans le shell courant avec : le numéro de 'job' entre crochets

```
$ sleep 7777 &
[1] 22471
```

```
$ sleep 9999 &
[1] 22472
$ jobs
[1]   En cours d'exécution   sleep 7777 &
[2]-  En cours d'exécution   sleep 9999 &
[3]+  En cours d'exécution   sleep 11111 &
('[3]+' signifie que le job numéro 3 est le dernier lancé
'[2]-' signifie que le job numéro 2 est l'avant-dernier
```

`Ctrl+z` : suspend le processus en cours d'exécution

`bg` : (*background*) relance l'exécution d'un processus suspendu en arrière-plan

`fg` : (*foreground*) relance l'exécution d'un processus suspendu en avant-plan

```
(suite du listing précédent)
$ sleep 4321 &
Ctrl+z
[4]+  Stoppé                 sleep 4321
(le job numéro 4 est suspendu)
$ jobs
[1]   En cours d'exécution   sleep 7777 &
[2]   En cours d'exécution   sleep 9999 &
[3]-  En cours d'exécution   sleep 11111 &
[4]+  Stoppé                 sleep 4321
$ bg %4
[4]+ sleep 4321 &
(le job numéro 4 est relancé en arrière-plan)
$ jobs
[1]   En cours d'exécution   sleep 7777 &
[2]   En cours d'exécution   sleep 9999 &
[3]-  En cours d'exécution   sleep 11111 &
[4]+  En cours d'exécution   sleep 4321 &
```

Exercice 4

1. Dans un système Linux, quel est le processus qui n'a pas de père. Quel est son PID ?
2. À quoi servent les options `-f` et `-L` de la commande `ps` ?
3. Lancez la commande `ps ax -f -L` et décrivez les différentes colonnes affichéees.
4. Utilisez la commande `pstree` pour afficher l'arbre des processus en cours avec leur PID.
5. Quelle option de `pstree` permet d'afficher les ancêtres d'un certain processus dont le PID est donné en argument à la commande ?
6. Dans le résultat suivant, que signifie le `'2*'` ?

```
$ ├─udevd───2*[udevd]
```

7. Quelle option de `pstree` permet d'afficher l'arbre complet des processus en cours (i.e., permet d'afficher toutes les occurrences de chaque processus) ?
8. Quel signal est lancé par défaut à la commande `kill` ?
9. Que font les signaux suivants : `SIGQUIT`, `SIGKILL`, `SIGCHLD` ?
10. Quel caractère permet de lancer une commande en arrière-plan ? Quelle commande permet de lancer une commande en arrière-plan ?
11. Affichez la liste des processus actifs dans votre terminal courant en utilisant `ps`.
12. Quelles informations sont affichées par défaut lorsque vous lancez la commande `top`?
13. Utilisez la commande `top` pour afficher la liste des processus en temps réel en ajoutant les colonnes `PID` et `UID` dans l'affichage.
14. Lancez 3 fois la commande `sleep` en arrière-plan avec des arguments différents à chaque fois.

    Affichez la liste des travaux en cours. Que signifient les caractères + et − dans le résultat obtenu ?

    Lancez ensuite la commande `sleep` en avant-plan.

    Reprenez la main dans votre terminal et affichez à nouveau la liste des travaux en cours.

    Relancez en arrière-plan la dernière commande `sleep`, que vous avez suspendue auparavant.

    Arrêtez l'exécution de la première commande `sleep` en utilisant son PID.

    Arrêtez l'exécution de la deuxième commande `sleep` en utilisant son numéro de travail.

    Arrêtez l'exécution des deux dernières commandes `sleep` en utilisant la combinaison de touches `Ctrl+c`.
15. Lancez la commande `gedit` depuis un terminal en avant-plan. Attendez que la fenêtre de l'application s'ouvre puis fermez le terminal d'où vous l'avez lancée. Que se passe-t-il ?
16. Lancez la commande `gedit` depuis un terminal en arrière-plan. Attendez que la fenêtre de l'application s'ouvre puis fermez le terminal d'où vous l'avez lancée. Que se passe-t-il ?
17. À quoi sert la commande `nohup` ? Comment faire pour à la fois détacher un processus du shell à partir duquel il est lancé et le lancer en arrière-plan ? Donnez un exemple.