ATTENTION : L'ensemble du TP doit être réalisé en utilisant les terminaux.

## Exercice 1 : Preparation de l'environnement du TP

Les commandes de préparation de l'environnement de TP doivent être effectuée depuis un terminal.

**Q1.** Créez le dossier `TP03` dans votre dossier `R1.04`.

**Q2.** Utilisez la commande `tree` sur votre dossier `R1.04`. Vérifiez que vous avez bien les dossiers `TP01`, `TP02` et `TP03` dans le dossier `R1.04`.

**Q3.** Rendez-vous dans votre dossier `TP03` et travaillez depuis ce dernier.

## Exercice 2 : Manipulation de processus

Par défaut un processus est démarré en **avant-plan** : tant qu'il n'est pas terminé, le shell ne permet pas d'en démarrer un autre. On peut également le démarrer en **arrière-plan** : le shell démarre le processus et s'en détache immédiatement (n'attend pas sa terminaison) en permettant d'en démarrer un autre immédiatement. Pour cela la commande démarrant le processus doit être suivie du caractère `&`.

Les processus sont créés hiérarchiquement par le système. La commande `pstree(1)` permet d'observer l'arbre de filiation des processus en cours d'exécution. (La syntaxe `pstree(1)` signifie que la documentation de `pstree` peut être trouvé dans la section `1` du manuel `man`.)

On peut envoyer des signaux aux processus grâce à la commande `kill(1)`. Parmi les signaux disponibles, on remarque notamment :

— `KILL` qui détruit le processus qui le reçoit ;
— `TERM` qui demande au processus qui le reçoit de s'arrêter ;
— `STOP` qui stoppe provisoirement le processus qui le reçoit ;
— `CONT` qui redémarre un processus préalablement stoppé.

ATTENTION : Les signaux ne sont pas des commandes mais bien des options à donner à la commande `kill(1)`.

Les processus peuvent être désignés par leur identifiant attribué par le système (PID) ou par leur identifiant attribué par le terminal auquel ils sont attachés (*job*).

Dans le cas des PID on peut utiliser leur numéro directement depuis n'importe quel terminal. On peut identifier leur numéro grâce à `ps(1)` qui liste les processus en cours d'exécution.

Dans le cas des jobs on peut utiliser leur numéro uniquement depuis le terminal auquel ils sont attachés et en préfixant celui-ci par le caractère `%`. La commande interne `jobs` liste les processus attachés au terminal dans lequel elle est exécutée.

On rappelle que la saisie de certaines combinaisons de touches dans un terminal permet d'envoyer des signaux au processus qui s'y exécute :

— Ctrl + c envoie le signal `KILL` ;
— Ctrl + z envoie le signal `STOP`.

Enfin les commandes `fg` et `bg` acceptent en paramètre un numéro de job et permettent respectivement de redémarrer un processus en avant-plan ou en arrière plan.

**Q1.** Essayez de comprendre le fonctionnement des processus en faisant les manipulations suivantes et en notant comment vous y êtes parvenu.

1. Démarrez la commande `xeyes`.
2. Essayez maintenant de démarrer une nouvelle fois la commande `xeyes`. Est-ce possible ?
3. **Stoppez** la commande démarrée la combinaison de touches Ctrl + z .
4. Redémarrez la commande stoppée en arrière-plan. (En cas de difficulté, relisez l'énoncé.)
5. Démarrez un nouveau processus en arrière plan exécutant la commande `xeyes`. (En cas de difficulté, relisez l'énoncé.)
6. Exécutez la ligne de commande suivante :

   ```
   xeyes & xcalc & xlogo & xclock & xload & xterm &
   ```

7. En utilisant la commande `ps` et les messages produits par les lignes de commande précédentes, associez à chaque numéro de *jobs* démarré le nom du programme qu'il exécute.
8. Vérifiez votre réponse avec la commande `jobs`.

9. Faites afficher la hiérarchie des processus en cours. Quelle commande avez-vous utilisée ? (En cas de difficulté, relisez l'énoncé.)
10. **Arrêtez** maintenant complètement les processus exécutant les commandes `xlogo` et `xload` en utilisant uniquement la commande `kill` que vous appliquez à des numéros de jobs.
11. Stoppez maintenant les processus exécutant les commandes `xclock` et `xcalc` en utilisant leur numéro de PID.
12. Essayez de vous servir de la calculatrice. Que se passe-t-il ?
13. Utilisez la commande `jobs` pour faire le bilan des processus actuellement attachés à votre terminal (les *jobs*).
14. Combien y a-t-il de processus stoppés ? en cours d'exécution ? Lesquels ?
15. Faites redémarrer le processus exécutant `xclock` via l'envoi d'un signal.
16. Le processus redémarre-t-il en arrière plan ou en avant-plan ?
17. Faites redémarrer le processus exécutant `xcalc` en avant plan via `fg`. Comment avez-vous du spécifier le processus : via son `PID` ou son numéro de job ?
18. Arrêtez le processus exécutant la commande `xcalc` sans utiliser la souris. Comment avez-vous fait ?
19. Faites passer le processus exécutant la commande `xclock` en avant plan. Comment avez-vous fait ?
20. Stoppez-le, puis faites-le redémarrer via `bg`. Comment avez-vous dû spécifier le processus : via son `PID` ou son numéro de job ?
21. Arrêtez tous les processus démarrés lors de cet exercice en une seule ligne de commande.

## Exercice 3 : Les redirections d'entrées/sorties

Par défaut quand un processus démarre, son entrée standard (fichier numéro 0), sa sortie standard (fichier numéro 1) et sa sortie d'erreur (fichier numéro 2) sont connectées au canal d'entrée et au canal de sortie du terminal auquel il est attaché.

Généralement il s'agit du clavier pour les entrées et de la partie de l'écran où se trouve la fenêtre pour les sorties. La commande `tty(1)` permet d'identifier le chemin absolu du fichier que le terminal utilise pour ses canaux.

Les descripteurs de fichiers peuvent être vu comme des variables. Par defaut les descripteurs 0, 1 et 2 contiennent le chemin vers le terminal actuellement ouvert. Dans ce cas, le processus considérera que le terminal est un fichier qu'il peut lire et dans lequel il peut écrire. Il est possible de changer les fichiers qui seront lu ou dans lequel le processus va écrire. Pour cela on utilise les chevrons, `<>`. Les chevrons inférieurs, `<`, sont utilisés pour les fichiers en entrées (ceux dans lesquels le processus peut lire) et les chevrons supérieurs, `>`, pour les fichiers en sortie (ceux dans lesquels le processus peut écrire).

La syntaxe complète utilisable est détaillée dans ce tableau :

| | |
|---|---|
| ⟨n⟩>⟨fichier⟩ | **Le descripteur numéro ⟨n⟩ correspond désormais au ⟨fichier⟩ et est utilisable en écriture. On dit que l'entrée ⟨n⟩ est redirigée vers ⟨fichier⟩ en écriture.** |
| ⟨n⟩>>⟨fichier⟩ | Le descripteur numéro ⟨n⟩ correspond désormais au ⟨fichier⟩ et est utilisable en écriture. De plus, il écrira à la suite du fichier sans détruire les données préalablement contenues dans ce fichier. |
| ⟨n⟩>&⟨m⟩ | Le descripteur numéro ⟨n⟩ correspond désormais au même fichier que le descripteur numéro ⟨m⟩ et est utilisable en écriture. |
| ⟨n⟩<⟨fichier⟩ | **Le descripteur numéro ⟨n⟩ correspond désormais au ⟨fichier⟩ et est utilisable en lecture. On dit que la sortie ⟨n⟩ est redirigée vers ⟨fichier⟩ en lecture.** |
| ⟨n⟩<<⟨marque⟩ | Le descripteur numéro ⟨n⟩ correspond désormais au ⟨fichier⟩ et est utilisable en lecture. De plus le fichier sera parcouru jusqu'à ce que la ⟨marque⟩ soit lue, puis la lecture s'arrêtera. |
| ⟨n⟩<&⟨m⟩ | Le descripteur numéro ⟨n⟩ correspond désormais au même fichier que le descripteur numéro ⟨m⟩ et est utilisable en lecture. |

Quelques exemples :
— `echo truc 1>toto` va créer un fichier `toto` (s'il n'existe pas) et écrire `truc` dans le fichier `toto`.
— `echo bidule 1>>toto` va ajouter le texte `bidule` à la suite du fichier `toto`.
— `mkdir / 2>toto` va écrire l'erreur provoquée par la commande `mkdir /` dans le fichier `toto`.
— `cat tata 1>toto 2>&1` va rediriger la sortie standard (descripteur numéro 1) de `cat tata` dans le fichier `toto` et la sortie erreur (descipteur numéro 2) vers le même fichier que la sortie standard (donc ici le fichier `toto`).

On rappelle par ailleurs que la saisie simultanée de la touche Ctrl + d dans un terminal ferme l'entrée standard pour la commande qui s'y exécute.

La commande `cat`, sans argument, a le comportement suivant : Elle lit ce qui lui est donnée en entrée standard et écrit ce qu'elle a lu dans sa sortie standard.

**Q1.** Sans utiliser d'éditeur de texte, mais en utilisant la commande `cat` et les redirection de fichiers (en particulier l'opérateur `1>`), créez les fichiers `fich1` et `fich2` suivant :

fich1

```
Ceci est un fichier utilisé en TP
```

fich2

```
Ce fichier se nomme fich2
Il est créé par une redirection dans le fichier fich2
En utilisant cat qui lit l'entrée standard

```

**Q2.** Démarrez en arrière-plan 3 terminaux graphiques différents. Dans la suite de l'énoncé, on désignera ces 3 terminaux par les alias `A`, `B` et `C`. Pour vous simplifier le travail, donnez comme titre à vos terminaux les alias spécifiés grâce à l'option `-t` de la commande `mate-terminal`.

**Q3.** Identifiez pour chacun des terminaux le fichier associé grâce à la commande `tty`.

**Q4.** Pour cette question vous devez saisir vos commandes **uniquement** dans le terminal `A`.

1. Affichez le contenu du fichier `fich1` dans le terminal `B`
2. Toujours en utilisant la commande `cat`, mais cette fois après avoir lu le manuel, créez le fichier `fich3` constitué de la concaténation du contenu des fichiers `fich1` et `fich2`.
3. Lancez la commande `cat fich1 fich150`.
4. Lancez la commande précédente en redirigeant la sortie standard dans le terminal `B` et la sortie d'erreur dans `C`.
5. Comment peut-on rediriger la sortie standard sur la sortie d'erreur ?
6. En utilisant la réponse à la question précédente, refaites la question 4 en redirigeant le tout vers le terminal `B`.