Scripts

Sous Unix il est possible d'écrire des programmes en shell. Pour cela il suffit de regrouper dans un fichier texte (*méthode* Unix) des lignes de commandes en s'assurant que :

1. la **première ligne** du fichier définie le *langage* à utiliser : `#!/bin/bash`,
2. le fichier doit être exécutable,
3. le shell puisse trouver le fichier.

Pour rendre le fichier `toto` exécutable on doit appliquer la commande :

```
chmod u+x toto
```

Lors de l'exécution du programme certaines chaînes de caractères sont remplacées par les éventuels paramètres de la ligne ayant servie à démarrer la commande :

- `$0`, `$1`, `$2`, etc. par le premier, second, troisième, etc. mots présents sur la ligne de commandes ;
- `$#` par le **nombre** d'arguments présents sur la ligne de commandes ;
- `$*` ou `"$@"` par l'**ensemble des arguments** présents sur la ligne commandes.

Dans ce contexte la commande `shift` permet d'oublier certains des arguments. Quand elle est appelée sans paramètre elle permet d'oublier le premier paramètre en réadaptant ces chaînes de caractères (le contenu de `$1` devient celui de `$2`, le contenu de la `$2` devient celui de `$3`, etc.) Dans ce cas le contenu de `$*`, `$#` et `"$@"` sont également mis à jour.

Exercice 1 : Premiers programmes : identité

Dans cet exercice vous allez travaillez sur la base des utilisateurs du système du département.

La commande `getent passwd toto` affiche une ligne d'information sur l'utilisateur `toto`. Cette ligne a la même structure que ce qui est décrit dans `passwd(5)`. Au département le nom complet de l'utilisateur (*user name or comment field*) est donné sous la forme `Prénom NOM`.

La commande `getent passwd` affiche sous la même forme la liste de tous les utilisateurs.

Q 1. Exécutez la commande suivante dans votre terminal :

```
source /home/public/m1101/tp/scripts.env
```

Q 2. Déterminez une ligne de commande permettant d'afficher le prénom et le nom de l'utilisateur `beaufils`.

Q 3. Écrivez un script `~/asr/bin/identite` qui prend en paramètre un *login* et qui affiche le prénom et le nom de l'utilisateur concerné.

Q 4. Écrivez un script `~/asr/bin/prenom` qui prend en paramètre un *login* et qui affiche le prénom de l'utilisateur concerné.

Q 5. Écrivez un script `~/asr/bin/nom` qui prend en paramètre un *login* et qui affiche le nom de l'utilisateur concerné.

Q 6. Écrivez un script `~/asr/bin/login` qui prend en paramètre un *prénom* et un *nom* et qui affiche le login de l'utilisateur concerné.

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 crées hierarchiquement par le système. La commande `pstree(1)` permet d'observer l'arbre de filiation des processus en cours d'exécution.

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 préalablemen stoppé.

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.

Exercice 2 : Prise en main

Q 1. Exécutez la commande suivante dans votre terminal :

```
source /home/public/m1101/tp/processus.env
```

Q 2. 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.
5. Démarrez un nouveau processus en arrière plan exécutant la commande `xeyes`.
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 ?
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 stoppé ? 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 redémarrez via `bg`. Comment avez-vous du 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.

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 connectés 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.

Il est possible de fixer ces connexions vers d'autres fichiers. Pour cela il suffit de placer des chevrons sur la ligne de commande qui démarre le processus. 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 détaillé dans ce tableau :

⟨*n*⟩**<**⟨*fichier*⟩	redirige le descripteur numéro ⟨*n*⟩ en lecture vers ⟨*fichier*⟩.
⟨*n*⟩**>**⟨*fichier*⟩	redirige le descripteur numéro ⟨*n*⟩ en écriture vers ⟨*fichier*⟩.
⟨*n*⟩**<<**⟨*marque*⟩	redirige le descripteur numéro ⟨*n*⟩ en lecture vers les lignes suivantes jusqu'à ce que la ⟨*marque*⟩ soit lue.
⟨*n*⟩**>>**⟨*fichier*⟩	redirige le descripteur numéro ⟨*n*⟩ à la fin de ⟨*fichier*⟩ sans détruire les données préalablement contenues dans ce fichier.
⟨*n*⟩**<&**⟨*m*⟩	duplique le descripteur numéro ⟨*n*⟩ sur le descripteur numéro ⟨*m*⟩ en lecture, ainsi ⟨*n*⟩ et ⟨*m*⟩ seront dirigés vers le même fichier.
⟨*n*⟩**>&**⟨*m*⟩	duplique le descripteur numéro ⟨*n*⟩ sur le descripteur numéro ⟨*m*⟩ en écriture.

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.

Exercice 3 : Manipulations basiques

Q 1. Sans utiliser la commande `vi`, ni un autre éditeur de texte, mais en utilisant la commande `cat` et l'opérateur **>** créez les fichiers `fich1` et `fich2` suivant :

`fich1`

```
Ceci est un fichier utilise en TP
Lecture de l'entree standard
Redirection dans le fichier fich1
Ce fichier se nomme donc fich1
```

`fich2`

```
Ce fichier se nomme fich2
Redirection dans le fichier fich2
Lecture de l'entree standard
Ceci est un fichier utilise en TP
```

Q 2. 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 donner comme titre à vos terminaux les alias spécifiés grâce à l'option **-t** de la commande `mate-terminal`.

Q 3. Pour cette question vous devez saisir vos commandes **uniquement** dans le terminal `A`.

1. Identifiez pour chacun des terminaux le fichier associé grâce à la commande **tty**.
2. Affichez le contenu du fichier `fich1` dans le terminal `B`
3. 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`.
4. Lancez la commande **cat fich1 fich150**.
5. Lancez la commande précédente en redirigeant la sortie standard dans le terminal `B` et la sortie d'erreur dans `C`.
6. Comment peut-on rediriger la sortie standard sur la sortie d'erreur ?
7. En utilisant la réponse à la question précédente refaites la question 5 en redirigeant le tout vers le terminal `B`.

Exercice 4 : Créateur de commandes

Q 1. Écrivez un script **~/asr/bin/creer-id** qui construit un script, dans un dossier dont le nom lui est passé en premier paramètre, permettant d'afficher l'identité d'un utilisateur, dont le login lui est donné en second paramètre. Le script que **creer-id** fabrique doit être nommé **id-login** avec **login** remplacé par le login de l'utilisateur recherché. Pour vos essais utilisez **~/asr/bin** comme premier paramètre et votre propre login ainsi que celui de votre enseignant comme second paramètre.