

Manipulation de processus

Lorsque vous utilisez un interpréteur de commandes (shell) UNIX, chaque commande est exécutée dans un nouveau processus. Par défaut les processus démarrés par le langage de commande le sont en **avant-plan** : vous ne pouvez exécuter de nouvelles commandes (on dit que vous *n'avez pas la main sur le terminal*) tant que le processus démarré n'est pas terminé. Il est possible de démarrer une commande via un processus en **arrière-plan** (ou *tâche de fond*). Pour cela il suffit de faire suivre la commande souhaitée par le caractère «&».

De ce fait les langages de commande permettent de démarrer plusieurs commandes en **séquence** via l'utilisation du caractère de séparation «;» ou en **parallèle** via l'utilisation du caractère «&».

De plus il est possible de grouper le démarrage des processus d'une suite de commande en les entourant de parenthèses. Par exemple, si un processus à créer en arrière plan doit comporter plusieurs commandes successives il suffit de grouper les commandes avec des parenthèses, de séparer les commandes souhaitées par des points virgule et de placer le caractère «&» après la dernière parenthèse fermante comme dans l'exemple suivant :

```
( commande1 ; commande2 ) &
```

Chaque processus est repéré par un identifiant système unique (PID). La commande `ps` permet de lister les processus en cours d'exécution sur le système. Cette commande comporte de très nombreuses options permettant, entre autre, de modifier les informations affichées ou la forme des affichages. La page du manuel vous renseignera en détail sur ces options.

Dans certains shells (dont celui que vous utilisez et qui se nomme `bash`) il existe une notion connexe aux identifiants de processus : les identifiants de *tâches* ou de *jobs*. Un job est un processus attaché à un terminal en cours d'utilisation. Dans les lignes de commandes on représentera les jobs par le caractère «%» suivi du numéro identifiant la tâche. La commande `jobs` permet de lister les tâches en cours et leur état. Pour chaque démarrage en arrière-plan le langage de commande signifie à l'utilisateur le numéro de job, suivi du numéro de processus (PID).

En `bash` si un processus est exécuté en avant plan, il est possible de le suspendre en frappant simultanément sur les touches `Ctrl` et `Z`. Un processus suspendu peut être redémarré en avant-plan via la commande `fg` (pour *foreground*), ou en arrière plan avec la commande `bg` (pour *background*).

La commande `kill` permet d'envoyer des signaux aux processus. Cette commande demande au moins deux paramètres. Le premier permet de préciser le signal à envoyer au processus. Le second correspond à l'identification du processus à signaler (via un PID ou numéro de job). Trois signaux sont particulièrement utiles : `KILL` qui demande la destruction sans condition d'un processus, `STOP` qui en demande la suspension, et `CONT` qui demande le redémarrage d'un processus préalablement stoppé.

Exercice 1 : Manipulation de base

- Avant de commencer les exercices lisez les pages du manuel de `kill` et de `ps`.
- De façon à ce que le shell vous signifie la fin d'un processus sans aucune attente exécutez la commande «`set -b`» dans votre terminal.

1. Démarrez la commande `xeyes`.
2. Essayez maintenant de démarrer une nouvelle fois la commande `xeyes`. Est-ce possible ?
3. Stoppez la commande démarrée en frappant simultanément les touches `Ctrl` et `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` ainsi que les messages produits par la ligne de commande de la question précédente associez à chaque numéro de *jobs* démarré le nom du programme qu'il exécute.

8. Vérifier votre réponse avec la commande `jobs`.
9. Après avoir lu la page du manuel de la commande `ps`, déterminez graphiquement 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 (numéros généralement inférieur à 10) c'est-à-dire sans utiliser de PID.
11. Stoppez maintenant les processus exécutant les commandes `xclock` et `xcalc` en utilisant leur numéro de processus (PID). Pour stopper les processus vous devez envoyer le signal `STOP` aux processus concernés via la commande `kill`.
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 (`CONT`) avec la commande `kill` à celui-ci.
16. Le processus redémarre-t-il en arrière plan ou en avant-plan ?
17. Faites redémarrer le processus exécutant `xcalc` en avant plan via l'appel de la commande `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 la commande `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.

Exercice 2 : Manipulation des processus et de leur état

Q 1. Dans cet exercice vous allez utiliser une commande particulière nommée `sleep`. À l'aide du manuel informez-vous du comportement et de l'utilisation de cette commande.

Q 2. Essayez et comparez les commandes suivantes :

```
sleep 5 ; echo A
echo A ; sleep 5
sleep 5 & echo A
echo A & sleep 5
(echo A ; sleep 5) &
(sleep 5 ; echo A) &
```

Q 3. Remplissez le tableau suivant, de façon à représenter *graphiquement* le comportement de chaque commande (par convention le caractère \$ représentera l'apparition du prompt) :

Commandes	t = 0s	t = 5s	t=10s
echo A ; sleep 5 ; echo B	A	B\$	
echo A ; sleep 5 & echo B			
(echo A ; sleep 5) & echo B			
echo A ; (sleep 5 & echo B)			
echo A ; (sleep 5 ; echo B) &			
sleep 5 & echo A ; (sleep 5 ; echo B)			
sleep 5 & echo A & (sleep 5 ; echo B)			
sleep 5 & echo A & (sleep 5 & echo B) &			
sleep 5 & echo A & (sleep 5 ; echo B) &			

Q 4. Exécutez la commande suivante, puis uniquement à l'aide des facilités de manipulations des jobs faites en sorte que le message `fini2` apparaissent avant le message `fini1` :

```
(sleep 60 ; echo fini1) & (sleep 120 ; echo fini2)
```

Q 5. Quelle suite de commandes ou actions avez-vous utilisée ?

Redirections

Les processus possèdent au moins trois descripteurs de fichiers qui peuvent être redirigés :

- **0** : l'**entrée standard**, par défaut c'est le terminal en mode lecture
- **1** : la **sortie standard**, par défaut c'est le terminal en mode écriture
- **2** : la **sortie d'erreur**, par défaut c'est le terminal en mode écriture

Pour rediriger ces fichiers il suffit de compléter les commandes par une syntaxe particulière. Pour mémoire le tableau 1 rappelle ces différentes syntaxes de redirection. Une même ligne de commande peut contenir plusieurs redirections qui seront évaluées en séquence.

Les terminaux correspondent, comme quasiment tous les éléments sous UNIX, à des fichiers. La commande `tty` permet d'obtenir le chemin absolu du fichier correspondant au terminal dans lequel elle est exécutée.

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

Table 1 – La syntaxe des redirections sous bash

Exercice 3 : Les redirections d'entrées/sorties

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 :

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

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

Q 2. Lisez la page du manuel de `xterm` de façon à trouver comment vous pouvez lors du lancement de la commande spécifier un titre au terminal que vous démarrez. Ce titre devra être présent dans la barre de titre de la fenêtre démarrée. Essayez par exemple de démarrer un terminal X ayant pour titre le mot `toto`.

Q 3. Démarrez 3 terminaux graphiques en utilisant la commande `xterm` et les possibilités de démarrage en arrière-plan. Dans la suite de l'énoncé on désignera ces 3 terminaux par les alias A, B et C. Le terminal A correspondra toujours à celui dans lequel vous entrez vos commandes. Pour vous simplifier le travail donner comme titre à vos terminaux les alias spécifiés.

Q 4.

1. Identifiez pour chacun des terminaux le fichier associé.
2. Afficher 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.

Q 5. Faites en sorte que tout ce que ce qui sera saisi dans le terminal A s’affiche dans le terminal C.

Q 6. Faites en sorte que tout ce que ce qui sera saisi dans le terminal terminal B s’affiche dans le terminal C mais après avoir été mis en majuscule. Une étude attentive de la page du manuel de la commande `tr` devrait vous aider.

Q 7. Faites en sorte que toutes les lignes qui seront saisies dans le terminal B s’affiche dans le terminal C uniquement si elles comportent le mot `toto` au moins une fois. Une étude attentive de la page du manuel de la commande `grep` devrait vous aider.

Les tubes

Contrairement aux processus lancés simultanément qui s’exécutent sans relation entre eux, des processus concurrents sont synchronisés via la production d’information par l’un et la consommation d’information par l’autre. Il est nécessaire pour cela, que le processus producteur soit apte à produire des caractères sur la sortie standard et que le processus consommateur soit apte à lire des caractères sur l’entrée standard.

C’est ce que réalise la fonctionnalité des tubes (*pipe*) dont la syntaxe est :

```
commande1 | commande2
```

De ce fait il existe de nombreuses commandes UNIX qui profitent de ce genre de communication, notamment les *filtres* :

«des programmes qui lisent des données (généralement ligne par ligne) sur leur entrée standard, y appliquent un certain nombre de manipulations et les retournent sur leur sortie standard»

Le tableau 2 résume un certain nombre de ces filtres. Vous obtiendrez plus d’informations en consultant les pages du manuel concernées.

<code>cat</code>	retourne les lignes lues sans modification.
<code>cut</code>	ne retourne que certaines parties de chaque lignes lues.
<code>grep</code>	retourne uniquement les lignes lues qui correspondent à un modèle particulier ou qui contiennent un mot précis.
<code>head</code>	retourne les premières lignes lues.
<code>more</code>	retourne les lignes lues par bloc (dont la taille dépend du nombre de lignes affichables par le terminal) en demandant une confirmation à l'utilisateur entre chaque bloc.
<code>sort</code>	trie les lignes lues.
<code>tail</code>	retourne les dernières lignes lues.
<code>tee</code>	envoie les données lues sur la sortie standard ET dans un fichier passé en paramètre.
<code>tr</code>	remplace des caractères lus par d'autres.
<code>uniq</code>	supprime les lignes consécutives identiques.
<code>wc</code>	retourne le nombre de caractères, mots et lignes lus.
<code>sed</code>	version filtre de la commande <code>ed</code> (les requêtes <code>ed</code> utilisées s'appliquent à chaque ligne lue)

Table 2 – Quelques filtres UNIX sur les fichiers textes

Exercice 4 : Petite introduction

Par exemple, la commande `ps -ef | grep bash` :

1. le premier processus, celui qui exécute la commande `ps`, envoie sur sa sortie standard la liste de tous les processus qui s’exécutent sur la machine,
2. le tube (*pipe*) sert de raccordement de cette sortie standard vers l’entrée standard du second processus,
3. le processus qui exécute la commande `grep` ne va lui-même afficher que les lignes contenant la chaîne `bash`.

Q 1. Exécutez la commande `ps -ef` puis la commande `ps -ef | grep bash`

Q 2. Créez, dans votre dossier `asr3/unix03`, un fichier `pipes` avec l’éditeur `vi` dont le contenu est le suivant :

```
1 :GGG: la commande tty (5) permet de connaitre le fichier associe au terminal
2 :HHH: la commande pipe (4) sert a la communication entre processus
3 :EEE: la commande tee (3) est utile pour capturer les informations qui circulent dans un pipe
4 :DDD: la commande tee (2) recopie son entree standard sur sa sortie standard et sur un fichier
```

```
5 :CCC: la commande tee (1) peut etre utilisee pour sauvegarder dans un fichier les traces des
6 :BBB: informations qui circulent sur sa sortie standard.
7 :AAA: qu'est-ce qu'un pipe (0) et que fait la commande TEE (c'est pourtant simple)
```

Q 3. Pour comprendre le fonctionnement des filtres standards `cat`, `tr`, `cut` et `grep` écrivez les commandes permettant d'effectuer les actions suivantes **sans tubes (lisez bien leur page dans le manuel)** :

1. afficher le contenu du fichier `pipes`
2. afficher le contenu du fichier `pipes` avec toutes les caractères « `i` » remplacé par des caractères « `I` »
3. afficher le contenu du fichier `pipes` avec toutes les lettres remplacées par leur équivalent en majuscules
4. afficher le contenu du fichier `pipes` en supprimant toutes les répétitions de lettres majuscules (on ne garde qu'une fois les lettres majuscules répétées).
5. afficher le contenu du fichier `pipes` en supprimant tous les chiffres.
6. afficher le premier caractère de chaque ligne du fichier `pipes`
7. afficher le contenu du fichier `pipes` en supprimant le premier caractère de chaque ligne
8. afficher le contenu du fichier `pipes` en supprimant le second **mot** de chaque ligne (un mot est une suite de caractères et est séparé du mot suivant par un espace)
9. afficher uniquement les lignes du fichier `pipes` contenant le *mot* `commande`
10. afficher les lignes du fichier `pipes` contenant le caractère « `-` ».
11. afficher les lignes du fichier `pipes` **ne contenant pas** le caractère « `-` ».
12. afficher les lignes du fichier `pipes` contenant le mot `tee` quelque soit la hauteur des lettres (majuscule ou minuscule) qui le composent.

Q 4. En combinant différents filtres via des tubes écrivez maintenant les commandes permettant d'effectuer les actions suivantes :

1. afficher dans l'ordre lexicographique le second caractère de chaque ligne du fichier `pipes`
2. afficher uniquement la 5ème ligne du fichier `pipes` en remplaçant toutes les lettres par leur équivalent en minuscules
3. afficher le nombre de ligne du fichier `pipes` contenant **exactement** le mot *pour*
4. afficher le nombre de ligne du fichier `pipes` contenant le mot *fichier* dans le terminal courant et les lignes contenant le mot *fichier* dans un second terminal (la commande `tty` permet de connaître le fichier attaché à un terminal).
5. afficher le nombre de ligne du fichier `pipes` contenant le mot *fichier* dans le terminal courant et les lignes contenant le mot *fichier* dans un second terminal. Les lignes apparaissant sur le second terminal devront être triées sur le premier mot de chaque ligne.
6. afficher le nombre de ligne du fichier `pipes` contenant le mot *commande* dans le terminal courant et les lignes contenant le mot *commande* dans un second terminal. Les lignes apparaissant sur le second terminal devront être triées sur le cinquième mot de chaque ligne.
7. afficher le nombre de ligne du fichier `pipes` contenant le mot *pipe* dans le terminal courant et les lignes contenant le mot *pipe* dans un second terminal. Le **mot** *pipe* devra de plus être remplacé par le mot *tube*.