

# Initiation aux systèmes d'exploitation Math-Info

## TP n° 5 : les processus UNIX et leurs entrées / sorties

Rappel : sous Ubuntu configuré en français, bash interprète [M-P] comme [MnNoOpP] et non [MNOP]. Pour remédier à ce problème, vous devez ajouter la commande suivante dans votre fichier `.bashrc` :

```
export LC_COLLATE=C
```

(sans espace autour du signe =). Faites-le impérativement sur votre compte du script **avant** le contrôle sur machine.





### Les processus

#### Obtenir des informations sur les processus

« `ps` » appelée sans option, affiche la liste des processus de l'utilisateur courant rattachés au terminal courant.

#### Exercice 1 – lister les processus : la commande « `ps` »

Lancer quelques programmes (navigateur web, emacs) depuis votre interface graphique (sans utiliser le terminal). Ensuite, dans un terminal, lancer les utilitaires « `xcalc` » et « `xclock` » via les commandes « `xcalc &` » et « `xclock &` ». Notez que le caractère `&` permet de reprendre la main immédiatement dans le terminal.

1.  Sans fermer les programmes que vous venez de lancer, tester la commande « `ps` » (sans options) dans ce terminal, puis dans un autre. Quels processus voyez-vous ? Quelle option de « `ps` » permet d'afficher tous vos processus y compris ceux qui ne sont pas rattachés à un terminal ? Quelle option de « `ps` » permet d'afficher tous vos processus rattachés à un terminal (éventuellement sans afficher les processus bash de chaque terminal) ? Pour chaque processus, quelle est la signification des informations fournies ?
2. Certaines options de « `ps` », comme « `-l` » ou « `u` », permettent d'afficher plus de détails sur les processus.  Tester ces deux options et repérer les caractéristiques des processus correspondant à « `xcalc` » et « `xclock` ».  Déterminer (à l'aide du manuel) la signification des colonnes UID, PPID, PRI, TT(Y), SZ, S(TAT).
3.  Comment afficher avec « `ps` » la liste de tous les processus du système, y compris ceux dont vous n'êtes pas le propriétaire et ceux qui ne dépendent pas d'un terminal ?

4. 🚧 Comment obtenir la liste des processus dont le propriétaire est *machin*? En déduire comment afficher les processus appartenant au *super utilisateur (root)*.
5. 🚧 Quelle option utiliser pour afficher le processus dont le PID est *n*? (remplacer *n* par un PID existant).
6. 🚧 Pour chacun des processus que vous avez lancés dans le terminal (*xclock*, etc.), quel est le PID de son parent? Quel est ce processus parent? Remonter l'arbre généalogique du processus exécutant « *xclock* » et identifier ses ancêtres sur 3 générations.
7. À l'aide de la commande « *ps tree* », afficher l'arbre généalogique des processus rattachés à votre terminal.

## Communiquer avec les processus et les contrôler via des signaux

« *kill* » permet d'envoyer différents types de signaux à un processus dont on connaît l'identifiant (PID). Malgré son nom, elle ne sert pas *seulement* à «tuer» un processus.

Syntaxe d'envoi d'un signal : `kill -signal PID` où *signal* est un numéro ou un nom de signal (le signal par défaut est SIGTERM), et *PID* l'identifiant du processus concerné.

Il existe de nombreux signaux différents dont vous trouverez la signification dans le man. Les signaux les plus courants sont SIGTERM, SIGKILL pour terminer (tuer) un processus, ainsi que SIGSTOP et SIGCONT qui servent respectivement à arrêter provisoirement / reprendre l'exécution d'un processus là où elle en était restée.

### Exercice 2 – envoyer des signaux : la commande « *kill* »

1. 🚧 Comment obtenir la liste de tous les signaux que l'on peut envoyer aux processus? Quels sont les numéros correspondant à SIGTERM, SIGKILL, SIGINT, SIGSTOP et SIGCONT?
2. Tester les cinq signaux ci-dessus sur des processus « *xcalc &* ». 🚧 Que se passe-t-il?
3. 🚧 Essayer maintenant d'envoyer ces mêmes signaux au processus *bash* d'où « *xcalc &* » a été lancé. Noter et comparer les effets obtenus, notamment en termes du sort destiné au fils « *xcalc* » et de la survie de la fenêtre. (Vous ouvrirez une nouvelle fenêtre, d'où vous lancerez un nouvel « *xcalc &* » à chaque fois que nécessaire).
4. Créer une fenêtre terminal par la commande « *xterm &* », dans cette fenêtre lancez un processus « *xcalc &* ». 🚧 Remonter l'arbre généalogique du processus exécutant « *xcalc* » et identifier ses ancêtres sur 3 générations.

5. 🚧 Tester les signaux SIGTERM, SIGKILL et SIGINT sur le processus « xterm & » correspondant au terminal où ce « xcalc & » est exécuté . Noter et comparer les effets obtenus, notamment en termes du sort destiné aux descendants du processus tué (vous vérifierez leur statut par « ps » après avoir envoyé le signal) et de la survie de la fenêtre terminal. (Vous ouvrirez une nouvelle fenêtre avec « xterm & », d’où vous lancerez un nouvel « xcalc & » à chaque fois que nécessaire).
6. Repérer parmi les processus actifs un processus dont vous n’êtes pas propriétaire et tenter de le stopper par l’envoi du signal SIGSTOP. 🚧 Que se passe-t-il ? Qu’en déduisez-vous ?
7. La commande « killall » permet d’envoyer des signaux à tous les processus repérés par leur nom et non par leur identifiant. 🚧 Testez-la sur les processus correspondant à la commande xclock après avoir lancé plusieurs processus avec cette commande.

### Exercice 3 – le signal SIGHUP

1. 🚧 Lancer un terminal et y exécuter « xclock & ». Relever le PID du processus, puis fermer le terminal. Dans un autre terminal, afficher les informations concernant le processus « xclock ». Que constatez-vous ?

« nohup » permet de protéger un processus contre le signal SIGHUP : lorsqu’un terminal est fermé, les processus qui en dépendent reçoivent le signal SIGHUP, ce qui par défaut provoque leur terminaison.

2. 🚧 Relancer un processus « xclock » immunisé contre le signal SIGHUP, et vérifier qu’il survit à la fermeture du terminal depuis lequel il a été lancé.
3. 🚧 Quel est le PPID de « xclock » avant et après la fermeture du terminal ? On dit que le processus *orphelin* a été adopté.

### Gérer les tâches (ou jobs) liées au shell courant

Les shells modernes, en particulier bash, fournissent un ensemble de mécanismes pour gérer l’exécution des processus lancés depuis un même terminal. Dans ce contexte, on parle de *tâches (job)* de la *session de travail* (définie par le shell courant).

Chaque tâche, correspondant à un *groupe de processus*, est identifiée de manière interne au shell, et peut se trouver dans trois états distincts :

- en avant-plan (*foreground*) : elle peut lire et écrire dans le terminal ; **au plus une** tâche peut avoir cet état (par défaut, le shell, qui réagit aux commandes saisies) ;
- en arrière-plan (*background*) : elle s’exécute sans lire ni écrire dans le terminal ;

- *stoppé* ou *suspendu* : la tâche est en sommeil, son exécution est interrompue.

#### Exercice 4 – états et changements d'états

1. Depuis un terminal, lancer un processus « `xcalc` » **avec un & à la fin**. ➤ Dans quel état se trouve ce processus ?
2. Maintenant, dans le terminal, lancer un processus (par exemple « `emacs` ») **sans le & final**. Dans quel état se trouve ce processus ? Presser dans le terminal la combinaison de touches `ctrl-Z`, aussi appelée commande de suspension (*suspend*). Pouvez-vous encore utiliser « `emacs` » ? ➤ Dans quel état se trouve ce processus ?
3. La combinaison de touches `ctrl-Z` correspond à l'envoi d'un signal. ➤ En utilisant le manuel de « `kill` », déterminer le nom du signal envoyé. Vérifier votre réponse en utilisant « `kill` ».
4. ➤ En utilisant « `kill` », envoyer un signal à un processus suspendu pour lui faire reprendre son exécution.

Nous sommes maintenant capables de lancer des processus à l'avant ou à l'arrière-plan, de suspendre des processus et de reprendre leur exécution.

Pour simplifier la manipulation des processus dépendant d'un même shell, il leur est attribué un numéro de tâche (*job number*) propre au shell qui les contrôle (différent en particulier du PID).

« `jobs` » affiche une liste de ces processus triée par numéro de job, ainsi que leur état actuel. Un signe "+" marque le job courant, un "-" le job précédent.

#### Exercice 5 – la commande « `jobs` »

1. ➤ Dans un nouveau terminal, tester cette commande et comparer sa sortie à celle de « `ps` » sans argument.
2. ➤ Exécuter « `xclock &` », « `xcalc &` » puis « `jobs` ». Que constatez-vous sur les numéros de job associés aux programmes ? Comment afficher le PID du processus correspondant à une tâche avec la commande « `jobs` » ?

Deux commandes supplémentaires permettent de ramener un processus à l'avant-plan ( « `fg` » ) ou de faire reprendre son exécution à l'arrière-plan à un processus interrompu ( « `bg` » ). Sans argument, ces commandes s'appliquent au processus courant (cf. remarques précédentes). Elles peuvent aussi être suivies d'un argument de la forme %n, où n est un numéro de job.

**Exercice 6 – les commandes « fg » et « bg »**

1. Lancer les jobs « xclock & » et « xcalc & ». ➤ Amener « xclock » à l'avant-plan et puis de nouveau en arrière-plan. Combien de processus peuvent être en avant-plan ?
2. ➤ À quelle commande est équivalente la séquence : « xcalc » suivi de *ctrl-Z* puis de « bg » ?
3. ➤ Terminer chacun de vos jobs en les ramenant à l'avant-plan et en pressant la combinaison de touches *ctrl-C*. À quel signal correspond ce raccourci ?

**Les entrées / sorties**

Chaque processus a dans son contexte d'exécution trois fichiers logiques : un nommé *entrée standard* (« *stdin* ») et les deux autres appelés respectivement *sortie standard* (« *stdout* ») et *sortie erreur standard* (« *stderr* »), comme illustré dans la Figure 1. Un processus peut utiliser l'entrée standard pour lire des données, et peut écrire deux types de réponses dans les fichiers de sortie. On pourra aussi parler de *flots* (ou *flux*) d'entrée et de sortie.



FIGURE 1 – Flot d'entrée et de sortie d'un processus UNIX.

Quelques exemples :

- « cat » écrit sur la sortie standard ce qu'elle reçoit sur l'entrée standard.
- « date » ne prend rien sur le flot d'entrée et écrit sur le flot de sortie.
- « cd » ne prend rien en entrée et ne renvoie rien en sortie.

Toutes ces commandes peuvent renvoyer des informations sur la sortie erreur standard pour signaler un problème quelconque (mauvaise utilisation de la commande, arguments inexistants, problèmes de droits, etc.).

Par commodité, il est souvent nécessaire de sauver les données en entrée ou en sortie d'un processus dans des fichiers afin de pouvoir ensuite les réutiliser, les voir en totalité ou les traiter. Par défaut, les flots standard correspondent au terminal : l'entrée standard est donc saisie au clavier et les sorties (standard et erreur) s'affichent à l'écran. Mais le mécanisme de *redirection* permet de leur substituer un fichier.

À chaque flot d'entrée/sortie est associé un entier, appelé *descripteur* :

- 0 correspond à l'entrée standard,
- 1 correspond à la sortie standard et
- 2 correspond à la sortie erreur standard.

**Avant de passer aux exercices,** télécharger depuis Moodle le script `tp5.sh`. Son exécution crée une arborescence de racine `~/Cours/IS1/TP5`. Placez-vous dans ce répertoire.

## Redirection de la sortie standard : les symboles `>` et `>>`

### Exercice 7 – les symboles `>` (ou `1>`) et `>>` (ou `1>>`)

1. Le mécanisme de redirection utilisé avec la commande « `echo` » lors des précédents TP pour créer des fichiers s'applique à toutes les commandes. Exécuter par exemple `date > fic` puis afficher le contenu du fichier `fic`. Exécuter ensuite `cal > fic` et afficher de nouveau le contenu du fichier `fic`. ➤ Qu'en concluez-vous sur le rôle de « `>` » ?
2. Refaire les manipulations précédentes en utilisant à chaque fois `>>` (ou `1>>`) au lieu de `>`. ➤ Qu'en concluez-vous sur le rôle de `>>` ?

Sous UNIX, les périphériques sont considérés comme des fichiers spéciaux, accessibles par des liens situés dans la sous-arborescence `/dev` du système de fichiers. Il existe deux types de tels fichiers spéciaux : “caractère” (terminaux etc.) ou “bloc” (disques etc.). Ces termes désignent la manière de traiter les lectures et écritures : soit caractère par caractère, soit par blocs.

### Exercice 8 – le terminal, un fichier très spécial...

1. La commande « `tty` » retourne la référence absolue du fichier spécial correspondant au terminal dans lequel elle est exécutée. Afficher les caractéristiques (droits, etc.) de ce fichier. ➤ Quel est son type : bloc ou caractère ?
2. ➤ Dans un autre terminal, exécuter la commande `date` en redirigeant sa sortie standard vers le fichier spécial correspondant au premier terminal. Que se passe-t-il ?

## Redirection de la sortie erreur : les symboles `2>` et `2>>`

En bash, la sortie erreur est redirigée grâce au symbole `2>`.

### Exercice 9 – rediriger les messages d'erreur

1. Dans `~/Cours/IS1/TP5/Shadocks`, exécuter la commande « `cat gabuzomeu` » (nom qui n'est pas censé exister). Vous devez obtenir un message d'erreur. Refaire la même opération en redirigeant la sortie standard dans un (nouveau) fichier `bugabu`. Que constatez-vous ?

2. Exécuter ensuite `man gabuzomeu 2> bugabu` et comparer. Recommencer en remplaçant `2>` par `2>>`. Que constatez-vous ?
3. Exécuter la commande « `meuzobuga` » (qui n'existe probablement pas plus que le fichier précédent) en redirigeant la sortie d'erreur vers un fichier `meuzobu`, puis afficher le contenu de ce fichier : la commande « `meuzobuga` » n'existe pas, pourtant le message d'erreur a été redirigé. 🐞 Expliquer pourquoi.

« `/dev/null` » est un fichier spécial qui se comporte comme un “puits sans fond” : on peut écrire dedans tant qu'on le veut et les données sont alors perdues. Il peut par exemple servir à jeter la sortie erreur quand on n'en a pas besoin.

### Exercice 10 – le fichier `/dev/null`

1. Afficher le contenu de tous les fichiers appartenant à un sous-répertoire du répertoire `LesCowboysFringants`, et dont le nom contient au moins une majuscule. 🐞 Faire en sorte d'éliminer les messages d'erreur pour obtenir un affichage lisible.
2. 🐞 Donner une ligne de commande qui utilise les messages d'erreur de « `ls` » pour déterminer la liste des répertoires non accessibles dans l'arborescence de racine `LesCowboysFringants` (sans donc afficher les fichiers et les répertoires accessibles).

## Redirection de l'entrée

Le symbole `<` sert à rediriger le flot d'entrée. En exécutant `cmd < fic` on passe à la commande « `cmd` » le contenu du fichier « `fic` » (et pas le fichier brut).

### Exercice 11 – le symbole « `<` »

Dans le répertoire `~/Cours/IS1/TP5/EnfantsDuParadis` exécuter les commandes « `cat Garance` » et « `cat < Garance` ». Faire de même avec la commande « `wc` ». 🐞 Expliquer la différence de comportement entre « `cat` » et « `wc` ».

« `tee` » sans argument, double le flot de sortie : le flot de sortie est à la fois affiché dans le terminal et recopié dans un (ou plusieurs) fichier(s) que l'on passe en paramètre.

### Exercice 12 – copies multiples simultanées

Toujours dans le répertoire `EnfantsDuParadis`, exécuter la commande `tee Pantomime < Baptiste`. Comparer les deux fichiers « `Baptiste` » et « `Pantomime` ». Que remarque-t-on ?

🐞 Quelle ligne de commande utilisant « `tee` » permet de créer simultanément plusieurs copies du fichier `Frederick`, respectivement appelées `Lemaitre`, `Acteur`, `Jeune` ?

## Combiner les redirections : >&

On peut bien entendu faire plusieurs redirections pour la même commande, par exemple :

```
commande > sortie 2> erreur
commande > sortie < entree      (ou commande < entree > sortie)
```

On peut également rediriger simultanément les deux flots de sortie (standard ou erreur) sur le même fichier à l'aide de >& :

```
commande >& sortie_et_erreur
```

### Exercice 13 – toutes les sorties dans le même fichier

➤ À partir du répertoire ~/Cours/IS1/TP5/Shadoks afficher le contenu de tous les fichiers du sous-répertoire Personnages faisant en sorte que la sortie et la sortie erreur soient écrites dans un fichier gagabubu en les redirigeant séparément (en deux lignes de commande), et dans un fichier bubugaga en les redirigeant simultanément.

## Quelques commandes pouvant manipuler l'entrée standard

La plupart des commandes qui manipulent l'entrée standard admettent en général un argument facultatif qui est un nom de fichier sur le contenu duquel elles peuvent s'exécuter. Il peut néanmoins être intéressant de les faire travailler directement sur l'entrée standard.

« cat » sans argument, recopie sur la sortie ce qui arrive dans le flot d'entrée.

### Exercice 14 – la commande « cat »

Retourner dans le répertoire ~/Cours/IS1/TP5/EnfantsDuParadis.

1. Lancer « cat » sans argument. La ligne reste vide, le shell attend que vous lui fournissiez des données. Taper « Arrête de répéter tout ce que je dis! » suivis de la touche entrée et observer le résultat.  
Taper encore quelques lignes, puis presser la combinaison de touches *ctrl-D* au début d'une ligne vide. Cela indique au processus la fin des données à traiter.
2. Relancer « cat », taper la même phrase qu'en haut (ou des caractères de votre choix) puis, sans aller à la ligne, appuyer sur *ctrl-D*. Que se passe-t-il ?  
Taper une nouvelle ligne et, au lieu d'appuyer sur la touche *Entrée* ou *ctrl-D*, taper maintenant *ctrl-C*. ➤ Expliquer la différence de comportement.
3. Essayer l'option « -n » de *cat*, d'abord sans argument (pour que la commande lise son entrée au clavier) et après en lui passant comme argument « Lacenaire ». ➤ Que fait l'option « -n » ?



4. Chercher dans la page man ce que fait l'option « -s ». Pour la tester, il pourra être utile de la combiner avec l'option « -n ». ➤ Créer ensuite un fichier `Criminel`, lisible sans devoir utiliser la barre de défilement, à partir du fichier `Lacenaire`.

« `head` » et « `tail` » lisent sur leur entrée standard et conservent les premières lignes (pour « `head` ») et dernières (pour « `tail` »). Le nombre de lignes conservées est 10 par défaut, ou l'entier passé en argument après l'option « -n ».

### Exercice 15 – les commandes « `head` » et « `tail` »

1. ➤ Tester les commandes « `head` » et « `tail` » en leur passant comme paramètre le fichier « `Criminel` ». Afficher ensuite seulement les 6 premières lignes du même fichier.
2. Lancer la commande « `tail -n 3` ». Comme dans l'exercice précédent, le shell attend que vous fournissiez des données, que la commande traitera ligne par ligne. Taper cinq lignes de texte suivies par `ctrl-D`. ➤ Qu'affiche votre commande ? Pourquoi ?
3. Comparer avec la commande `head -n 3`. ➤ Que se passe-t-il ? Expliquer la différence.
4. ➤ À quoi sert l'option « -c » de ces commandes ? La tester.
5. Pour la commande « `tail` », le nombre passé avec les options « -n » ou « -c » peut être précédé d'un signe « + », pour indiquer qu'il est compté à partir du début et non à partir de la fin. Tester `tail -n +2` et taper cinq lignes sur le flot d'entrée, pour vérifier que vous avez bien compris. ➤ Faire de même en lui passant comme paramètre le fichier « `Criminel` ». Qu'obtient-on ?
6. Lancer la commande « `head` » en lui donnant les deux paramètres « `Garance` » et « `Baptiste` ». Que peut-on remarquer ? ➤ Déterminer quelle option de « `tail` » permet de supprimer l'affichage des noms.
7. ➤ À partir du répertoire « `/Cours/IS1/TP5/` » créer un fichier `LeGorille` à partir des 10 premières lignes de chaque fichier contenu dans le répertoire « `Brassens` ». De façon similaire, créer un fichier « `LesPassants` » en utilisant les 6 dernières lignes de chaque fichier.