TD3 : Linux « avancé »
V1.5.0

Cette œuvre est mise à disposition selon les termes de la licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Partage à l'Identique 3.0 non transposé.

Document en ligne : www.mickael-martin-nevot.com

1 Généralités

Sans mention contraire, vous vous positionnerez dans votre (sous-) répertoire `tp3` durant l'ensemble de ce TD.

N'oubliez pas de consulter le manuel à chaque fois que cela est nécessaire, en particulier à chaque découverte d'une nouvelle commande. Vous pouvez aussi faire des recherches sur le Web en prenant soin de vérifier que les informations trouvées soient correctes.

2 TD sans souris

Réalisez ce TD sans souris : seul le clavier est autorisé !

Après une période d'adaptation, vous pourriez bien être surpris du temps gagné au lieu de la gêne initialement imaginée.

Figure 1 – TD sans souris

(cc) BY-NC-SA

3 Alias

Un alias est une commande « courte » qui se comporte comme une (ligne de) commande plus longue. Utiliser des alias a comme principal avantage de gagner du temps (il y a moins de caractères à taper pour lancer une commande).

Vous connaissez déjà l'utilisation de la commande `ls -al --color=auto` : créez-en un alias nommé `llc` en utilisant la commande `alias`. Vérifiez que les deux lignes de commande produisent bien le même résultat.

Quittez le *shell* et le terminal avec la commande `exit` puis ouvrez un nouveau *shell* et essayez de nouveau d'utiliser la commande `llc` : constatez qu'elle ne fonctionne plus. En effet, toutes les modifications de configurations faites dans un *shell* ne sont valables que dans ce *shell* : lorsque vous le quittez puis en exécutez un autre, c'est la configuration par défaut qui est chargée.

4 Configuration des interpréteurs de commandes

Un *shell* Bash utilise trois fichiers de configuration (il va les chercher dans votre *home*) :

- `.bash_profile` ;
- `.bashrc` ;
- `.bash_logout`.

Au vu du rôle critique de ces fichiers, soyez toujours prudent et concentré si vous êtes amené à les modifier.

Pour rendre les alias persistants, vous devez ajouter leur déclaration (en utilisant la commande `alias` exactement comme vous venez d'apprendre à le faire) dans le fichier `.bashrc`. Soit il y a une section (souvent commentée) dédiée aux alias dans le fichier : il faut alors les placer à cet endroit (après les éventuels autres alias) ; soit il est conseillé de les ajouter à la fin du fichier.

Éditez le fichier `.bashrc` en ajoutant l'alias `llc` correspondant à la ligne de commande `ls -al --color=auto` et testez le résultat : l'alias ne semble toujours pas être pris en compte (pour le moment) ! En effet, la configuration du *shell* est chargée à son lancement : il faut donc exécuter un nouveau *shell* pour bénéficier des modifications faites dans le fichier `.bashrc` ou utiliser la commande `source` destinée à cet effet. Essayez les deux méthodes en créant d'autres alias (au moins une douzaine) de votre choix (pensez à `ll` correspondant à la ligne de commande `ls -l`, `md` pour `mkdir`, etc.).

La modification des fichiers de configuration du *shell* (et de `.bashrc` en particulier) permet en outre de personnaliser le *prompt* et l'affichage. Grâce au manuel, personnalisez conséquemment votre configuration en modifiant les fichiers `.bash_profile` et/ou `.bashrc`.

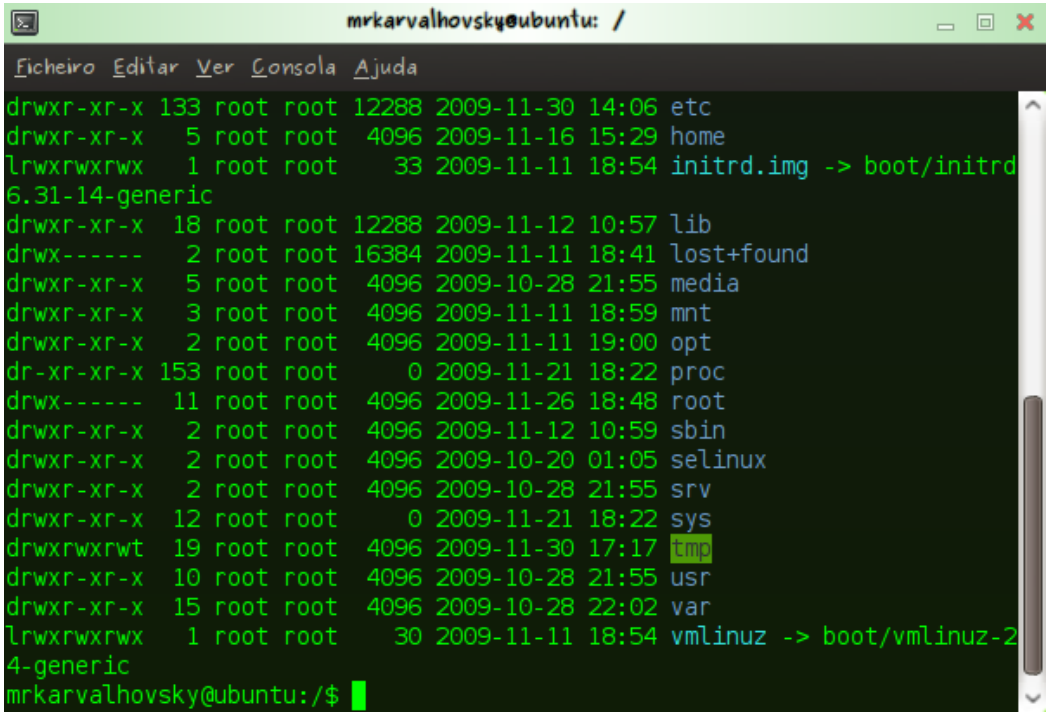

Figure 2 – Exemple de configuration de shell

5 Observation et gestion des processus

5.1 Processus : théorie et pratique

Vous connaissez déjà la différence entre programme et processus mais voici quelques approfondissements.

Un programme est une entité passive stockée sur « disque » (sur un disque dur par exemple) qui ressemble en tout point à un fichier traditionnel. À ce stade de l'explication, un programme n'est ni plus ni moins qu'une suite de *bits* (c.-à-d. des chiffres binaires). Lorsque vous exécutez un programme, il est alors chargé dans la mémoire vive (ou RAM). L'ordinateur va alors lire la première instruction du programme, la décoder et effectuer l'action correspondante. Puis il passe à l'instruction suivante et recommence… Lors de l'exécution du programme, les adresses mémoires (sortes de casiers de rangement) où est stocké votre programme, peuvent changer. Ainsi, un processus évolue, ce n'est pas le cas d'un programme.

Sous Linux, les processus sont indépendants, identifiés par un numéro (appelé PID), et toujours créés à partir d'un autre processus à deux exceptions près : les processus `idle` (PID 0, non visible) et `init` (PID 1) sont créés initialement, c.-à-d. au démarrage du système d'exploitation. Le premier sert à gérer les temps d'inactivité du processeur et le second sert à créer les premiers processus nécessaires au système (qui vont eux aussi créer des processus). Nous obtenons donc une structure arborescente où chaque processus à un père (le processus qui l'a créé) et un ou plusieurs fils (les processus qu'il a créés).

À partir d'un *shell*, un processus peut être lancé de deux façons différentes :

- au **premier plan** (ou *foreground*) : en tapant directement la ligne de commande (comme vous avez l'habitude de faire) ; par exemple : `geany` ;

(cc) BY-NC-SA

- en **arrière-plan** (ou *background* ou encore en tâche de fond) : en tapant la ligne de commande traditionnelle suivie du caractère `&` ; par exemple : geany `&`.

Lorsqu'un processus est lancé au premier plan, le *shell* devient inactif (puisqu'il est occupé à exécuter le processus) et le *prompt* ne s'affiche de nouveau qu'à l'arrêt du processus. Au contraire, si le processus est lancé en arrière-plan, le *shell* reste disponible (le *prompt* s'affiche donc de nouveau) : cette méthode permet de lancer plusieurs processus à partir du même *shell*.

Il est possible de faire passer un processus du premier plan en arrière-plan, et *vice versa* :
- si le processus tourne au **premier plan** :
  - presser `Ctrl+Z` le suspend (le processus est alors « en pause ») ;
  - presser `Ctrl+C` l'interrompt (le processus est alors arrêté brutalement) ;
- si le processus est **suspendu**, un prompt s'affiche à l'écran, alors :
  - la commande `fg` fait passer ce processus au premier plan ;
  - la commande `bg` fait passer ce processus en arrière-plan ;
- si le processus tourne en **arrière-plan** :
  - si c'est le seul processus lancé à partir du *shell*, alors la commande `fg` le passe au premier plan ;
  - sinon, il faut tout d'abord afficher la liste des processus lancés à partir de votre terminal à l'aide de la commande `jobs`, puis taper la ligne de commande : `fg numJob` (où `numJob` est le numéro correspondant à votre processus donné par la commande `jobs`).

5.2 Mise en pratique

Vous connaissez déjà la commande `ps` : déterminez à quoi correspondent ses options `-l`, `-A` et `-f`.

Lancez Geany au premier plan à partir d'un nouveau terminal (et donc d'un nouveau *shell*), faites-le passer en arrière-plan puis affichez la liste des processus liés à ce terminal. Déterminez, en utilisant la ligne de commande `ps -l`, à quelle application correspond un processus et quel est le processus père du processus correspondant à Geany. Repassez finalement Geany au premier plan.

Lancez un nouveau terminal (sans fermer le premier), puis identifiez, en utilisant la ligne de commande `ps -Af`, les PID et les propriétaires des processus correspondant à ce nouveau terminal et au terminal précédent. « Tuez » le processus correspondant au premier terminal à l'aide de la ligne de commande `kill -9 PID` (où `PID` est l'identifiant du terminal « à tuer »), puis concluez.

Déterminez le rôle de la commande `kill` (et de la commande `xkill`). Faites de même avec l'alias d'option `-9` (de forme atypique) et trouvez l'option originale correspondante. Enfin, faites de même avec l'alias d'option `-15` puis comparez les deux options.

Relancez encore un nouveau terminal puis lancez Geany en arrière-plan depuis son *shell*. Vérifiez que le processus père du processus correspondant à Geany est bien celui du *shell*. « Tuez » ce terminal : vérifiez s'il vous est toujours possible d'utiliser le logiciel Geany correspondant ; déterminez quel est son nouveau processus père et concluez.

La commande `ps` dispose de nombreuses options, essayez par exemple la ligne de commande suivante : `ps auxfw`.

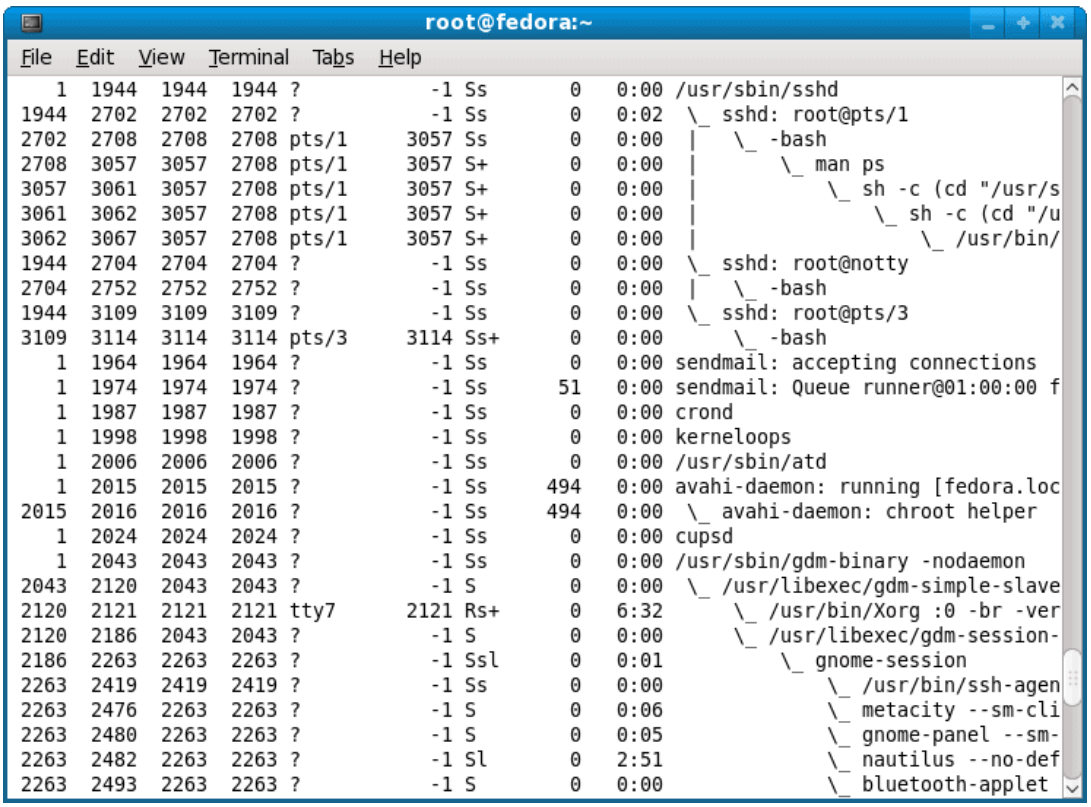

Figure 3 – Exemple d'affichage de la commande ps

Lancez plusieurs éditeurs de texte et navigateurs Web. Observez les processus créés correspondant aux diverses applications que vous venez de lancer (une actualisation de l'affichage est nécessaire à chaque nouveau processus). « Tuez » les processus ainsi créés avec la commande `xkill` (tout en étant prudent avec son utilisation).

6 Approfondissement des redirections

Vous connaissez déjà les caractères de redirection (> et >>), voici un canal de communication entre processus (cmd1 et cmd2, cmd3 étant des commandes) :

- cmd1 | cmd2 : le résultat obtenu par cmd1 est utilisé comme donnée par cmd2 (le caractère | se nomme *pipe*, ou encore tube de redirection) ; il est possible de combiner plusieurs | à la suite (par exemple : cmd1 | cmd2 | cmd3).

Tout d'abord, à l'aide d'une redirection (non écrasante), enregistrez la liste de tous les processus lancés dans le fichier `bar`. Vérifiez que tout s'est bien passé en affichant le fichier `bar`.

Déterminez le rôle des commandes `wc`, `sort`, `head` et `tail` et testez-les avec leurs différentes options. Puis :

- comptez le nombre de mots contenus dans votre fichier `/usr/share/dict/words` ;
- comptez le nombre de caractères contenus dans votre fichier `/usr/share/dict/words` ;
- comptez le nombre de lignes de `/usr/share/dict/words` ;
- triez les lignes du fichier `/usr/share/dict/words` dans l'ordre alphabétique inverse ;
- affichez (extrayez) les 20 premières lignes du fichier `/usr/share/dict/words` ;

- affichez les 5 dernières lignes du fichier `/usr/share/dict/words` ;
- affichez de la 8e à la 11e lignes du fichier `/usr/share/dict/words` ;
- afficher la pénultième ligne du fichier `/usr/share/dict/words` ;
- enregistrez dans le fichier `baz` les lignes quatre à vingt-trois du fichier `bar`, triées suivant le nom des propriétaires des processus, en utilisant uniquement les commandes `tail` et `head` ainsi que `|`.

Il est possible d'exécuter plusieurs commandes séquentiellement (les unes après les autres), en une seule ligne de commande, en utilisant le caractère `;` pour les séparer mais ceci n'est pas une redirection ou un canal de communication unidirectionnel entre processus ! En effet, les commandes sont alors indépendantes les unes des autres et ne réutilisent ni l'entrée ni la sortie d'une quelconque autre commande. Par exemple : `cat baz ; ls ; cd ~`.

7 Recherche d'une chaîne de caractères dans des fichiers textes

La commande `grep` permet de rechercher une chaîne de caractères (aussi appelée motif) dans les lignes d'un fichier texte. Cette commande incontournable est indispensable pour rechercher des informations, notamment dans les fichiers de configuration.

Déterminez quelles options utiliser pour que l'utilisation de la commande `grep` :

- donne seulement le nombre de lignes trouvées contenant le motif ;
- donne seulement le nom des fichiers dans lesquels le motif a été trouvé ;
- donne les lignes où le motif n'a pas été trouvé ;
- ne tienne pas compte de la casse ;
- impose que le motif corresponde à un mot entier sur une ligne du fichier.

Lisez dans le manuel la partie correspondant aux expressions régulières (aussi appelées expressions rationnelles ou modèles de chaînes). **Le concept d'expression régulière est différent du concept de filtrage mais a une syntaxe proche ce qui est parfois déroutant**. Déterminez quelles sont ces différences (focalisez-vous en particulier sur les caractères `.` et `?` et `*`).

Déterminez ensuite quel est le résultat des lignes de commande suivantes :

- `grep "^.o" bar` ;
- `grep -ni bash bar` ;
- `grep -ni "td$" bar` ;
- `grep --color "^.o" bar` ;
- `ps auxfw | grep user` (où `user` est votre identifiant d'utilisateur).

Enfin, déterminez les lignes de commande permettant de :

- sélectionner les lignes du fichier `bar` commençant par un « r » et terminant par un chiffre ;
- affichez uniquement les processus lancés par vous en utilisant uniquement les commandes `ps` et `grep` ainsi qu'un `|` ;
- afficher le nombre de processus lancés par vous en utilisant uniquement les commandes `ps`, `grep` et `wc` avec des `|`.

(cc) BY-NC-SA

8 Recherche de fichiers dans une arborescence

Déterminez le rôle des commandes `locate` et `find` et testez-les. Recherchez ensuite le fichier du programme `ls` et le fichier `baz`.

Simulez (grossièrement) la commande `find` à l'aide des commandes `ls` et `grep`.

9 Les variables d'environnement

L'ensemble de cet exercice doit être réalisé à partir d'un même *shell*.

9.1 Les variables

Une variable est définie par son **nom** et sa **valeur** :

- le nom d'une variable est une chaîne de caractères, par exemple : `maVar` ;
- la valeur d'une variable *shell* est obtenue en faisant précéder le nom de la variable par le symbole `$`. Par exemple `$maVar`.

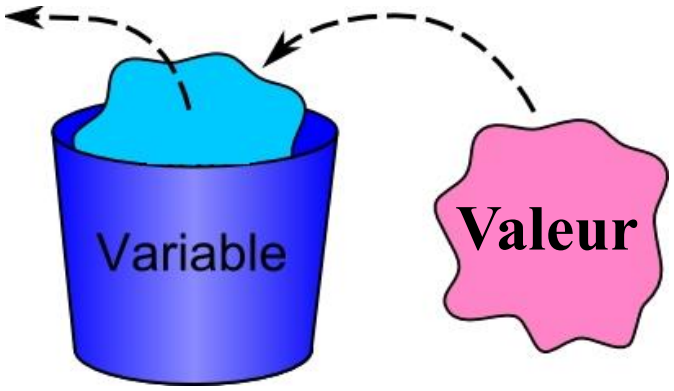

Figure 4 – Une variable

Les **variables d'environnement** sont utilisées par les différents processus d'un système d'exploitation pour communiquer entre eux.

9.2 Mise en pratique

Déterminez le rôle de la commande `echo` et des variables d'environnement `HOME`, `USER`, `DISPLAY`, `TERM`, `HOSTNAME`, `SHELL` et `PATH`. Puis affichez leurs valeurs.

Il est possible de créer de nouvelles variables d'environnement ou de modifier celles existantes. Pour cela il suffit de définir leur nouvelle valeur. Par exemple, la ligne de commande `TEMP0=hello` crée une nouvelle variable `TEMP0` (si elle n'existe pas déjà) qui a comme valeur la chaîne de caractères *« hello »* (attention, il ne doit pas y avoir d'espace autour du caractère =).

Vous connaissez l'utilité de `\`, de `"` et de `'`, faites donc la distinction entre les lignes de commande suivantes et concluez :

- `touch fichier $USER` ;
- `touch "fichier $USER"` ;
- `touch 'fichier $USER'`.

(cc) BY-NC-SA

Vous allez maintenant manipuler quelques variables (soyez vigilant : en cas de mauvaise manipulation, il vous faudra fermer le terminal avant d'en relancer un nouveau et de recommencer l'ensemble des manipulations suivantes depuis le début) :

- créez une variable `TEMP2` ayant comme valeur : *« my variable »* (attention aux espaces…) ;
- créez une variable `TEMP3` qui reçoit comme valeur celle de la variable `PATH` ;
- vérifiez que les variables `TEMP3` et `PATH` ont bien la même valeur ;
- donnez à la variable `PATH` une nouvelle valeur (par exemple *« hello »*) ; vérifiez que cela a bien fonctionné ;
- essayez maintenant d'exécuter la commande `ls` ; déterminez ce qu'il faut en déduire ;
- redonnez à la variable `PATH` sa valeur initiale (qui doit être enregistrée dans `TEMP3`) ;
- essayez de nouveau de lancer la commande `ls` ; déterminez de nouveau ce qu'il faut en déduire.